

CloudView CV23 Configuration

Table of Contents

Configuration.....	10
What's New?.....	11
Workflow and Concepts.....	12
About Index Schemas and Search Logics.....	12
What is an Index Schema?.....	12
What is a Search Logic?.....	13
Modifying the Index Schema and Search Logics.....	13
When do I need to Re-Index Documents?.....	14
The Indexing and Search Processes.....	15
About the Indexing Process.....	15
About the Search Process.....	17
Configuring CloudView with the Data Model.....	20
About the Data Model.....	20
What is the Data Model Expansion.....	21
What is generated by the Data Model Expansion.....	22
Controlling the Data Model Expansion.....	23
Taking Control over Generated Index Fields.....	25
Working with Data Model Classes.....	26
Can I Delete the Default Document Class?.....	26
Working with Multiple Classes.....	26
Impact of Multiple Classes on Performance.....	26
Using Properties to Configure Document Metas.....	26
Data Types and Semantic Types for Properties.....	27
Indexing Options for All Properties.....	27
Indexing Options for Alphanumeric Properties.....	29
Indexing Options for Numerical Properties.....	30
Indexing Options for Date Properties.....	30
Indexing Options for Geographical Properties.....	33
Indexing Options for Measure Properties.....	33
Creating Dynamic Properties.....	34
Add a Dynamic Property.....	34
About Storing and Displaying Dynamic Property Fields.....	35
Store Properties in a Parent Class Dynamic Property.....	37
Search Dynamic Property Fields.....	37
Creating Multivalued Properties.....	38
Tools to Create a Data Model from Your Corpus.....	39
Create a Data Model from Sample Documents.....	39
Store Unprocessed metas.....	39
Configuring Data Processing.....	41
Understanding and Using the Analysis Pipeline.....	41
About Data Processing.....	41
The Analysis Pipeline Sequence of Processors.....	45
Use Multiple Pipelines with Conditions.....	46
Use a Single Pipeline with Groups of Processors.....	47
Multiple Pipelines vs. Single Pipeline with Groups.....	48
Configuring the Analysis Pipeline Manually.....	48
Testing your Analysis Pipeline Behavior.....	51
Test the Analysis Pipeline with an Indexed Document.....	51
Test the Analysis Pipeline with a New Custom Document.....	56
Display Document Processing Information.....	57
Test the Semantic Processing of your Analysis Pipeline.....	59
More About Semantic Analysis.....	61
When does Semantic Analysis take Place?.....	62
Set Up Semantic Analysis?.....	62
Index-Time Semantic Analysis.....	63
Other Documentation about Semantic Analysis.....	63

Tokenizing Text.....	63
Using Native Tokenizers.....	64
Using Basis Tech Tokenizer.....	69
About Creating Additional Tokenization Configurations.....	72
Customizing the Tokenization Config.....	73
About Decomposing.....	74
Creating and Deploying Semantic Resources.....	76
Create a Resource File from the Administration Console.....	76
Manage Resources in cvadmin.....	76
Managing Semantic Annotations.....	82
Manage Annotations with the Annotation Manager.....	82
Manage Annotations with Custom Code.....	85
Configuring Form Indexing.....	87
Use Form indexing for Over-Indexing Acronyms.....	87
Set Weight.....	87
Configuring Search Queries.....	90
User Query Language (UQL).....	90
The Different Types of Search in UQL.....	91
Reserved Characters in UQL.....	99
Operands.....	100
Operators by Priority.....	102
More About INNERJOIN.....	105
Exalead Low-Level Query Language (ELLQL).....	107
Why Use ELLQL?.....	107
ELLQL vs UQL.....	107
ELLQL Syntax.....	107
Filtering Search Results in ELLQL.....	108
Defining Query Templates.....	108
Query Template Syntax.....	108
Reserved Named Queries.....	109
Use Case.....	109
Using Prefix Handlers.....	109
The Different Types of Prefix Handlers.....	110
Specify a Tokenization Configuration for Prefix Handlers.....	112
Configuring Query Expansion.....	113
Query Tree and Query Expansion.....	114
Query Expansion Features.....	116
Enable query expansion.....	117
Stemming.....	119
Lemmatization.....	120
Phonetization.....	122
Approximation.....	123
Normalization Exceptions.....	126
Synonyms.....	126
Japanese Synonyms.....	129
Configuring Dictionaries.....	129
About Dictionaries.....	130
Setting Up a Dictionary.....	131
Compacting and Building Dictionaries.....	132
Clearing Dictionaries.....	133
Adding 'Did You Mean?' Spell-Check.....	134
About Spell-Check.....	134
Setting Up Spell-Check.....	136
Adding Search Suggestions.....	138
About Search Suggestions.....	139
Create a Suggest Dictionary.....	142
Enable the Suggest in the Mashup UI.....	146
Use the Suggest Via the Search API.....	146
Export Suggest Dictionary Content to an XML File.....	147
Dispatch a Query to Several Suggest Dictionaries.....	148
Performance Considerations and Options for Search Suggest.....	152
Adding Related Terms.....	153
About Related Terms.....	153
Configure Related Terms and Similar Documents Detection.....	155
Configuring and Using Similarity Measures.....	159

Configure the Index for Similarity Queries.....	159
Use the #attrsimilar Function in the Search API.....	161
Code Samples to Create Similarity Query Prefix Handlers.....	164
Configuring Geographic Search.....	168
About Geographic Points.....	168
Create a Geographic Point.....	169
Search a Geographic Point.....	170
Calculate Distances in Virtual Fields.....	171
Use Geolocation Based on Place Detection.....	171
Adding a Query Cache.....	173
About Query Cache.....	174
Create and Manage a Query Cache.....	174
Configuring Search Results.....	176
Defining Search Results Content.....	176
Configure the Search Result Summary.....	176
Configure Value Selection for Metas.....	177
Configuring the Highlighting of Search Terms.....	178
Creating Facets to Refine Search Results.....	181
About facets.....	182
Create Facets.....	184
Numerical Range facets.....	185
Date Facets.....	187
Configure Date Facets.....	190
Multidimension Facets.....	193
Geographic Facets.....	195
Create Value Facets for Nonhierarchical Metas.....	198
Create Aggregations for Facets.....	198
Exclusive vs. Disjunctive Refinements.....	200
Calculating Results On-The-Fly with Virtual Fields.....	201
When to Use Virtual Fields.....	201
Performance Considerations.....	201
Virtual Field Syntax.....	202
Specifying a Timezone for Date Time Metas.....	202
Specify a Timezone in the Output Format.....	202
Convert Date Time Values to a Specific Timezone.....	202
Specify a Timezone at Search Time.....	203
Ranking and Sorting Search Results.....	204
About Ranking.....	204
Sorting.....	210
Collapsing/ Grouping Search Results.....	211
About Grouping.....	212
Setting Up Grouping.....	215
Setting the Limits of Search Results.....	216
Managing Saved Configurations.....	217
About Saved Configurations.....	217
How Applying Configuration Works.....	217
Apply Configuration Process.....	217
Comparing Configuration Versions.....	218
Rolling Back to a Previous Configuration.....	218
Editing the Configuration Manually.....	219
Edit a File in the API Console.....	219
Edit the Configuration Files Directly.....	220
Apply Changes in the Command Line.....	220
Apply changes when Exalead CloudView has stopped.....	220
Troubleshooting.....	222
Troubleshooting Document Analysis.....	222
Identify the Cause of the Index Crash.....	222
Unexpected Search Behavior.....	223
Analyzing User Queries with Reporters.....	226
About Reporters.....	227
Output Reporting Data to CSV Files.....	228

Output Reporting Data to a JDBC Database.....	230
Output Reporting Data to the Internal SQLite Database.....	231
Index Reporting Data as a Data Source.....	232
Available Fields for the Reporting Publishers.....	232
Performance Considerations.....	239
About Exalead CloudView Sizing.....	239
How Project Requirements Impact Sizing.....	239
Disk Requirements.....	240
RAM Sizing Formula.....	240
The Impact of the Data Model on Performance.....	241
How Property Options Impact Performance.....	241
How Classes Impact Performance.....	245
Dealing with Hierarchical Dimensions.....	245
Appendix - Configure Document Processors.....	247
Chunk Operations.....	247
Copy Context Chunks.....	247
Multi-Context Encoder.....	247
New Chunk.....	248
Remove Contexts.....	248
Rename Context for Chunks.....	248
Rename Unmapped Contexts.....	248
Replace Values.....	248
Value Selector.....	248
Normalization.....	249
Date Formatter.....	249
Numerical Formatter.....	250
Public URL Processor.....	250
Units of Measurement Normalizer.....	250
Numerical Operations.....	251
Double to Long.....	251
Fixed Range Numerical Partitioning.....	251
Forced Range Numerical Partitioning.....	251
Math Document Processor.....	252
Text to Num.....	252
Text Extraction.....	252
HTML Relevant Content Extractor.....	252
MIME Detector.....	253
Mime Type Setter.....	254
Semantic Web Document Processor.....	254
Standard Parts Merger.....	254
Text Extractor (All Mime Types).....	254
Text Extractor (text, html, exalead).....	255
Xpath Extractor.....	255
Xpath Fragment Extractor.....	256
Text Operations.....	256
Concatenate Values.....	256
Content Cleanup.....	256
Language Detector.....	256
Language Setter.....	257
Print Values.....	257
Replace Regexp.....	257
Split Values.....	258
String Hash.....	258
String Transform.....	258
Custom.....	258
Custom Document Processor.....	258
Java Document Processor.....	258
Remote HTTP Transformer.....	259
Other.....	259
Debug Processor.....	259
Discard Document Processor.....	259
Document Processor Group.....	259
Format Checker Date.....	259
Infer File Extension.....	260
Insert Current Date.....	260
Precomputed Thumbnails Document Processor.....	260

Random DocumentChunks Generator (Uniform Distribution).....	260
Random DocumentChunks Generator (Zipf Distribution).....	260
Real-Time Alerting.....	260
Semantic Pipe.....	260
Similar String to Part Converter.....	261
Storage Service Document Processor.....	261
UTF8 Checker.....	261
Appendix - Configure Semantic Processors.....	262
About Semantic Processors.....	262
Acronym Detector.....	263
Chunker.....	263
Compound Words Splitter.....	264
Example.....	264
When to Use.....	264
Dependencies.....	265
Fast Rules Matcher (Rule-Based).....	265
When to Use.....	265
Dependencies.....	266
Rule Nodes.....	266
Sample Fast Rules XML Files.....	266
Supported Queries.....	267
Rule Syntax.....	267
Create the Fast Rules Resource File.....	269
Map the Annotation to a Category Facet.....	269
Lemmatizer.....	270
When to Use.....	270
Configure Lemmatization Manually.....	270
Named Entities Matcher.....	270
When to Use.....	271
Which Entities are extracted?.....	271
Filtering Options.....	271
Named Entities Classes and Subclasses.....	272
Extract Your Own Named Entities.....	277
Set Block Lists and Allow Lists for Named Entities Extraction.....	278
NGram Extractor.....	279
Normalizer.....	280
Ontology Matcher (Resource-Based).....	280
Dependencies.....	281
Rules for Ontology Matching.....	281
Sample Ontology Matcher XML File.....	281
Ontology Rules Syntax.....	282
Multilevel Ontology Example.....	283
Create the Ontology Matcher Resource File.....	284
Map an Annotation to a Category Facet.....	285
Phonetizer.....	285
When to Use.....	285
Phonetize a Field Created from a Data Model Property.....	286
Configure Phonetization Manually.....	286
Proximity.....	287
How Is the Best Match Selected?.....	287
Configure the Proximity Processor.....	287
Related Terms.....	288
Required Settings.....	289
Optional Settings.....	289
Search-Time Configuration.....	290
Rules Matcher (Rule-Based).....	290
Dependencies.....	290
Basics of Creating Rules.....	291
Sample Rules Matcher XML File.....	291
Rules Syntax.....	294
Rules Best Practices.....	298
Caveats.....	298
Limitations.....	299
Create a Rules Matcher Resource File.....	299
Map the Annotation to a Category Facet.....	300

Semantic Extractor.....	300
Entities and Attributes.....	301
Rule Attributes.....	303
Dependencies.....	303
Sample Semantic Extractor XML File.....	304
Entities Syntax.....	305
Rules Syntax.....	307
Macros.....	309
Create the Semantic Extractor Resource File.....	310
Map the Annotation to a Category Facet.....	311
Semantic Query Analysis.....	311
Configure Semantic Query Analysis.....	311
Example 1: Define "Cheap" for an E-Commerce Site.....	312
Example 2: Define "Cheap" for Different Products.....	314
Snowball Stemmer.....	315
When to Use.....	315
Configure Stemming Manually.....	315
Part of Speech Tagger.....	315
How to use.....	315
When to use.....	315
Appendix - Semantic Resources Reference.....	316
Ontology.....	316
OInclude.....	316
Pkg.....	317
Entry.....	317
Form.....	318
FastRulesDefinition.....	319
Category.....	319
Rule.....	320
DateFormat.....	320
LemmaDictionary.....	321
Lemma.....	321
Inflected.....	322
NormalizationOverwrites.....	322
NormalizationOverwrite.....	322
NormalizationAlternatives.....	323
NormalizationAlternative.....	323
NormalizationExceptions.....	324
NormalizationException.....	324
RegexMatches.....	324
RegexMatch.....	325
SemanticExtractorConfig.....	326
Entity.....	327
TextEntity.....	327
BooleanEntity.....	329
IntegerEntity.....	330
FloatingPointEntity.....	332
RangeEntity.....	333
RegexEntity.....	334
Define.....	336
Include.....	336

Rule.....	336
Synonyms.....	337
SynonymSet.....	339
Synonym.....	339
TRules.....	340
Seq.....	341
Iter.....	342
Star.....	344
Plus.....	346
Opt.....	347
Sub.....	349
Or.....	351
Near.....	353
Noblank.....	354
PatternRef.....	356
And.....	358
Not.....	359
Nor.....	361
TokenKind.....	363
Paragraph.....	364
Sentence.....	366
Dash.....	368
Punct.....	369
Digits.....	371
Alnum.....	372
Alpha.....	374
TokenLanguage.....	376
AnyToken.....	377
TokenRegexp.....	379
Word.....	381
Annotation.....	382
Ctx.....	384
AnnotationRegexp.....	386
TRule.....	388
MatchAnnotation.....	390
TInclude.....	390
TImport.....	392
Remove.....	394
Copy.....	394
KeepLongestLeftMost.....	394
AnnotationProcessed.....	395
KeepLeftMostLongest.....	396
KeepFirst.....	396
SelectMostFrequentValue.....	397
SelectMostFrequentAnnotation.....	397

SelectByContexts.....	398
StringValue.....	398
Appendix - ELLQL Language.....	401
ELLQL Language Features.....	401
Structure of the Language.....	401
Options.....	401
Simple Operators.....	402
Fields Search.....	402
Specials.....	410
Delimiters.....	410
Compound Operators.....	410
Unary Operators.....	411
Binary Operators.....	412
Nary Operators.....	414
Proximity Operators.....	415
Appendix - Search API Parameters.....	417
The search Command.....	417
Global Parameters.....	417
Sorting and Grouping Parameters.....	420
User Query.....	425
UQL Interpretation.....	427
Limits Parameters.....	428
Hit Meta Parameters.....	429
Faceting Parameters.....	434
Dynamic Search Target.....	444
Textual Relevance Parameters.....	445
Unranked search mode.....	446
Search Logic Editing.....	446
Misc.....	449
The fetch, preview and thumbnail Commands.....	450
About Thumbnails.....	451
Global Parameters.....	451
Fetch Parameters.....	452
Preview Parameters.....	452
Thumbnail Parameters.....	452
The Search Results.....	453
The spellcheck Command.....	454
The suggest Command.....	454
The security Command.....	455
The expansion Command.....	456
The introspection Command.....	456
Appendix - Virtual Field Expression Syntax.....	457
What Is a Virtual Field Expression.....	457
Expression Types.....	457
Numerical Operators.....	458
Built-ins.....	459
General Functions.....	459
Mathematic Functions.....	459
Geographic Functions.....	460
Category Functions.....	461
Time Manipulation Functions.....	461
String Functions.....	464
Multivalued fields Manipulation Functions.....	464
Dynamic Fields Manipulation Functions.....	465
Type Casting.....	465
Special Functions.....	466
Ranking Elements.....	466

Configuration

This guide provides detailed explanations of core Exalead CloudView features, such as the data model and the analysis pipeline. It also introduces advanced functionalities such as virtual fields, dynamic facet, and geographic search.

Audience

The purpose of this guide is to help consultants, developers, or system Administrator who have previous experience setting up Exalead CloudView, or have already followed the Exalead CloudView Getting Started Guide.

Further Reading

You might need to refer to the following guides:

Guide	for more details on
Installation & Administration	installing the product and administrative tasks typically performed on production systems.
Mashup Builder	building the front-end of your search application.
XML Configuration Reference	very specific product options.

What's New?

There are no enhancements in this release.

Workflow and Concepts

This chapter describes the global product workflow and how indexing and search work.

To develop a search application in Exalead CloudView, you must define what data to include in the index schema. Then you must configure one or more search logics to control how to present the documents as search results.

[About Index Schemas and Search Logics](#)

[The Indexing and Search Processes](#)

About Index Schemas and Search Logics

What is an Index Schema?

Index schemas define the structure of the Exalead CloudView index, which constitutes of index fields and categories.

Index Fields

Index Fields help searching and displaying data in the hit content of the search results.

The following index field types are available:

- **Alphanumerical**: stores words or text chunks.
- **Numerical**: stores both integer and decimal numbers. Decimal fields are stored with a fixed precision, which you can specify.

Numerical fields support the following operations:

- Equality
- Inequality
- Larger than, smaller than
- Range search ("from X to Y")
- **Point**: stores location data. For more information, see [Configuring Geographic Search](#).
- **Date**

Categories

Categories store static facet values. These values display in the **Refinements** panel of the search results as well as in hit content. Static facet allow users to narrow their search results by focusing on a certain aspect of the results, such as a particular country or product line.

In the `categories` field, category values are stored in a tree. The root of the tree is called `Top`.

For example, for a given document, the `categories` field could contain the following categories:

```
Top/  
Top/Source  
Top/Source/files  
Top/Language  
Top/Language/fr  
Top/Detected Entities/People/Mr. Jones  
Top/Detected Entities/People/Mrs. Singh
```

You can use categories for navigation over static categories. To enable this navigation, Exalead CloudView stores the retrievable part of the category in RAM.

What is a Search Logic?

Search logics enable you to define:

- Which index fields or category facets to use for queries, using prefix handlers.
- What types of semantic interpretation to perform on queries, using semantic expansion modules.
- Which index fields or facets to return in the search results, using hit meta and facets.

You can create multiple search logics, which enables you to provide querying options and hit content specific for the destination search application.

Modifying the Index Schema and Search Logics

The simplest way to modify both the index schema and the search logic is to create classes and properties in the data model. For more information, see [Using Properties to Configure Document Metas](#).

For advanced configurations, you can directly modify the analysis pipeline and search logic. See the following:

- [Controlling the Data Model Expansion](#)
- [More About Semantic Analysis](#)
- [Configuring Query Expansion](#)

When do I need to Re-Index Documents?

The table below indicates when to re-index according to the sections modified in the Administration Console:

Modifications made on...	Needs re-indexing?
Index	
Connectors	Yes
Data Processing	Yes
Data Model	Yes Note: when you add a property to the data model, you only need to re-index to search or retrieve this new property.
Linguistics	Yes
Tuning	No Changes made in Analyze and Commit require a restart of Exalead CloudView, or at least the indexing server process, to be taken into account. Changes made in Compact , do not require re-indexing, a full compact is enough to clean index slots.
Search	
Search Logics	No
Security Sources	No
Search API	No
Suggest	No
Reporting	No
Deployment	
Roles	No
Build Groups	Yes
Plug-ins	No
Resources	No

The Indexing and Search Processes

The Exalead CloudView index is generational. When indexing documents, they are divided into batches known as jobs.

Each time you index a job, it creates a new generation of the index. Exalead CloudView stores this new generation of the index in a data structure called a slot. Each new slot is appended to the original index.

Once you commit the latest generation (slot) to the index, the index replicas are updated.

At search time, Exalead CloudView searches in all these slots of all the index replicas, and merges the results to return the final result set.

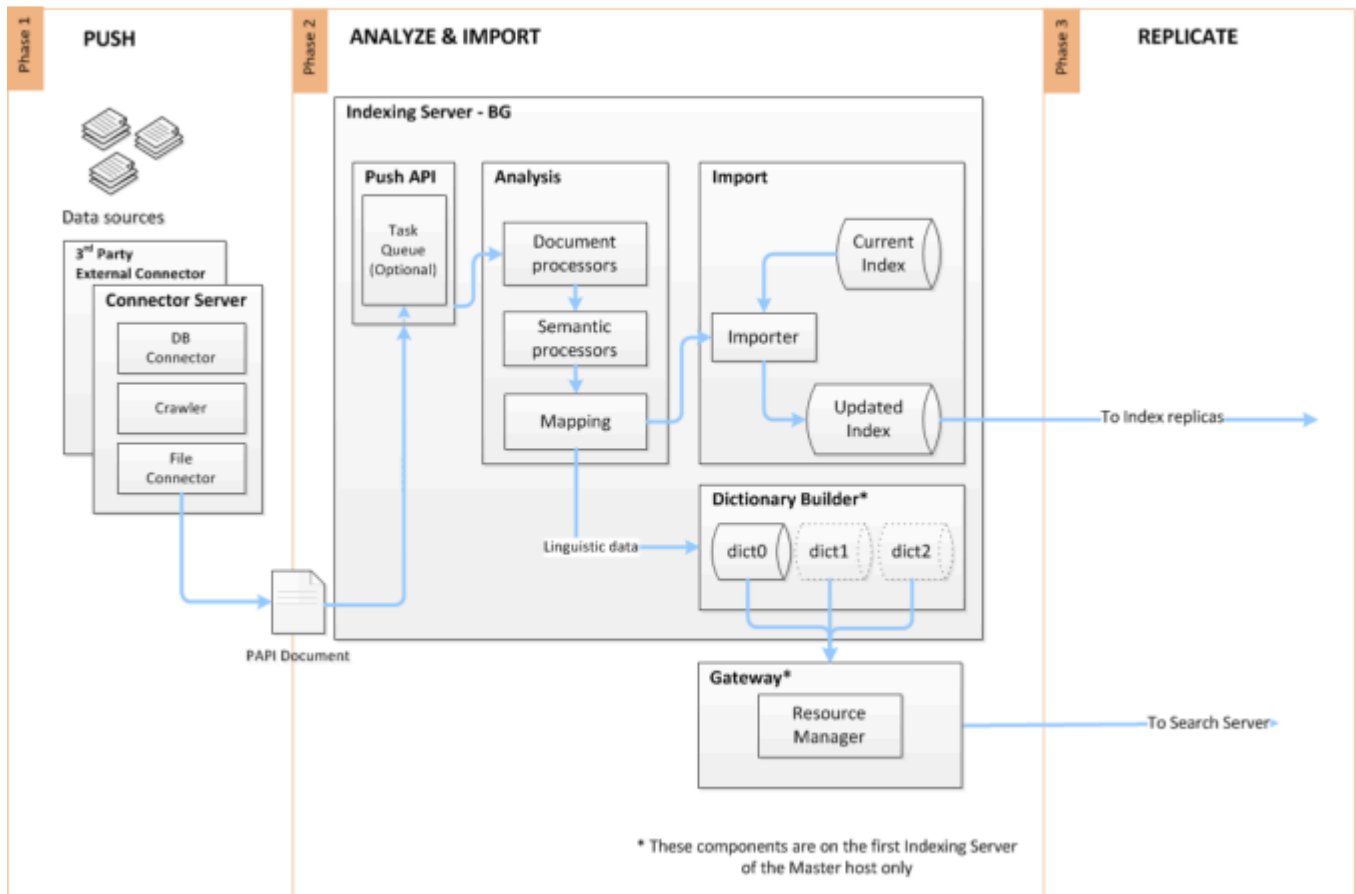
From time to time, slots are compacted to create an index with fewer slots, for more efficient searching.

About the Indexing Process

Indexing is the process of scanning a document to create an index and to store its metas (content such as fields) into a collection.

The following diagram explains the process to update an index and its replicas.

Indexing Process



Phase 1: Push

The connectors retrieve documents from data sources, convert them into PAPI documents, and send these documents to the Indexing Server process.

For each document, the Indexing Server assigns:

- a Document Identifier (DID)
- an index slice

Important: You can add the Consolidation Phase between Phase 1 and Phase 2. It allows you to transform and aggregate documents coming from different sources before pushing them to the Indexing Server. For more details on consolidation, see the Exalead CloudView Consolidation Server Guide.

Phase 2: Analysis

Analysis is the process of formatting content and extracting information from documents pushed by connectors before storing them in the index.

In Exalead CloudView, an indexing job starts as soon as it receives a document. The analysis pipeline processes it immediately, using several threads for better performance.

Processors play an important role during the analysis phase. The document and semantic processors parse each document in the job to perform text extraction, semantic processing, custom operations, and mapping.

Commit Triggers define the conditions that prompts the saving of the analysis to the index.

When you commit, the results of the analysis create:

- An import to the index, which merges the data computed during the analysis with the data present in the index. This results in a new generation of the index. The new data resides in a new, separate slot in the index.
- Semantic annotations (linguistic statistical data) about the corpus to the dictionary builder of the indexing server process. This data can be used for query expansion and index-time semantic processing.

The index is now committed to disk.

Phase 3: Replicate

After the new index data is committed, the new index generation is replicated on all index slices in the deployment. Once fully replicated, the new documents are available for searching.

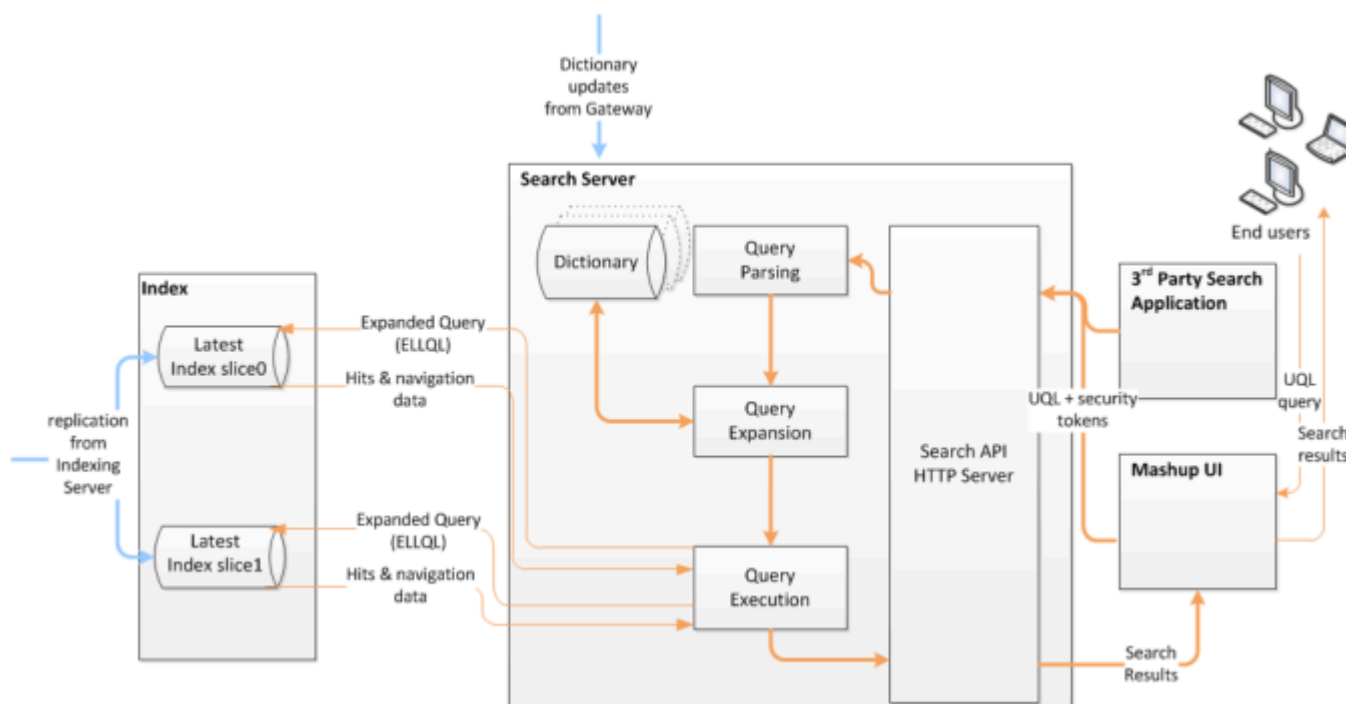
Once the dictionary builder has received new semantic annotations, it updates the dictionary (or dictionaries, if you configured multiple ones) on the search server.

About the Search Process

Search is triggered when end users (or a third-party application) submit a query to Exalead CloudView.

The following diagram shows how Exalead CloudView parses and expands queries before searching for matches on the index replicas.

Search Process



The user enters a query in UQL (User Query Language). The Mashup UI (or a custom search API application) forwards the query to the Search API. If you configured security, the query includes the security tokens.

Query Parsing

Parsing involves checking whether the query includes words and operators. If there are no operators in the original query, this step inserts the default operator (AND) between multiple words, unless the words are enclosed in quotations.

Query Expansion

To linguistically expand this query, Exalead CloudView consults the dictionary to check for words available in the corpus, then sends a fully expanded query to the index slices.

The fully expanded query breaks down into a more granular query language known as ELLQL (Exalead Lower-level Query Language) so the index slices can understand it. During the step, the ELLQL query includes the security tokens so that the index slices can verify whether you have access to the matching documents.

Query Execution

Query execution constitutes of:

- Searching for the most relevant matches in all index slices.
- Generating the navigation data (facets).

All slices receive the query because each slice only contains a portion of the corpus. The hits from each slice are merged in the search server before returning all the matching hits to the user.

Configuring CloudView with the Data Model

This chapter describes Data Model properties and expansion.

[About the Data Model](#)

[What is the Data Model Expansion](#)

[Working with Data Model Classes](#)

[Using Properties to Configure Document Metas](#)

[Creating Dynamic Properties](#)

[Creating Multivalued Properties](#)

[Tools to Create a Data Model from Your Corpus](#)

About the Data Model

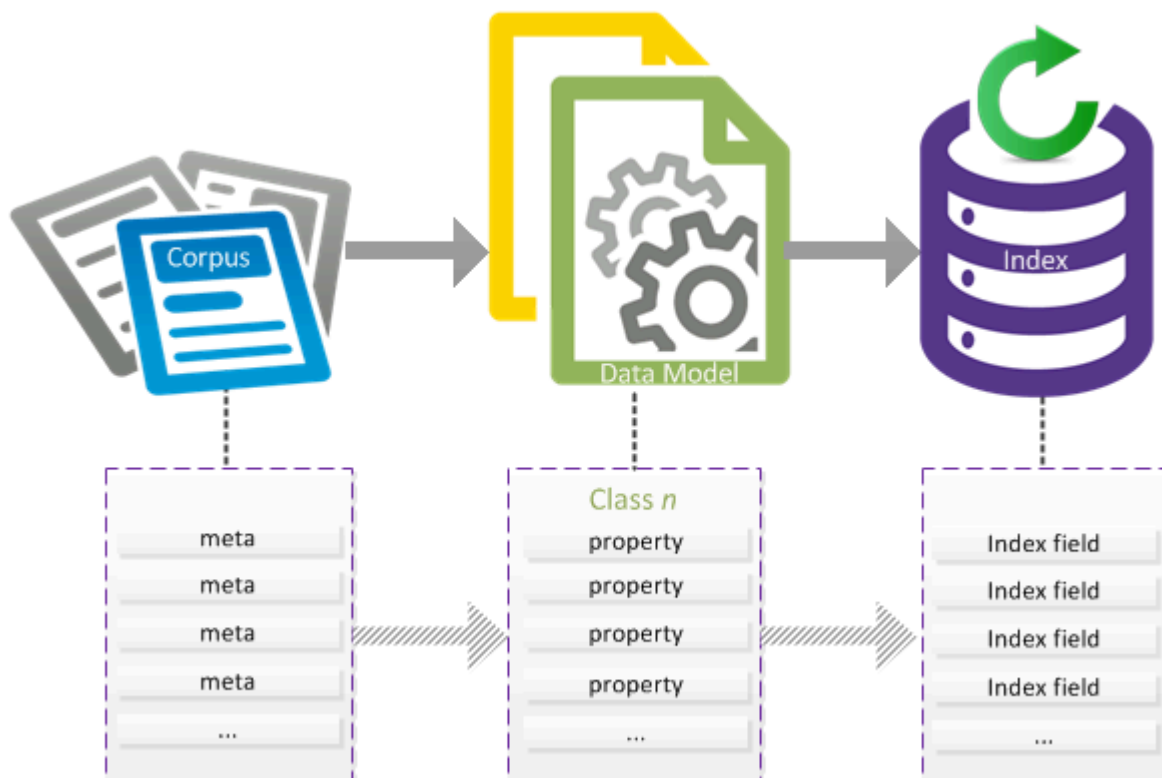
The **Data Model** defines how documents are stored in the Exalead CloudView index. It provides a way to map the relevant document metas available in your corpus to specific properties grouped by classes. The data model defines how to store data in the index, and how to use it at search time.

Classes allow you to group lists of properties to clearly structure your data model, according to the types of documents present in your corpus. They are similar to Business Logic views.

Note: A document can only belong to one class.

Properties define the configuration of the document metas you want to index. During indexing, Exalead CloudView converts properties into dedicated index fields, and optionally hit display fields or facets, depending on your configuration.

Transformation of Document Metas into Index Fields via the Data Model



Exalead CloudView is installed with a **default_model** data model containing a default `document` class but you can create your own data models and classes.

For example, when indexing an information system for a bookstore, we could have source system data for authors and for books. To separate these, we could create a `book` class and an `author` class, and each consists of a specific set of properties.

- The book class could have properties for document metas like: `title`, `author`, `year`, `publication`, `ISBN`, etc.
- The Author class could have properties for document metas like: `last name`, `first name`, `date of birth`, `biography`, etc.

Important: All changes made to the data model require clearing the index and re-indexing data to avoid inconsistencies.

What is the Data Model Expansion

Data model expansion is the automatic generation of index schema elements (index fields, categories, processing, and mappings) as well as search logic elements (prefix handlers, hit meta, and facets) when defining a property in a data model class.

What is generated by the Data Model Expansion

Controlling the Data Model Expansion

Taking Control over Generated Index Fields

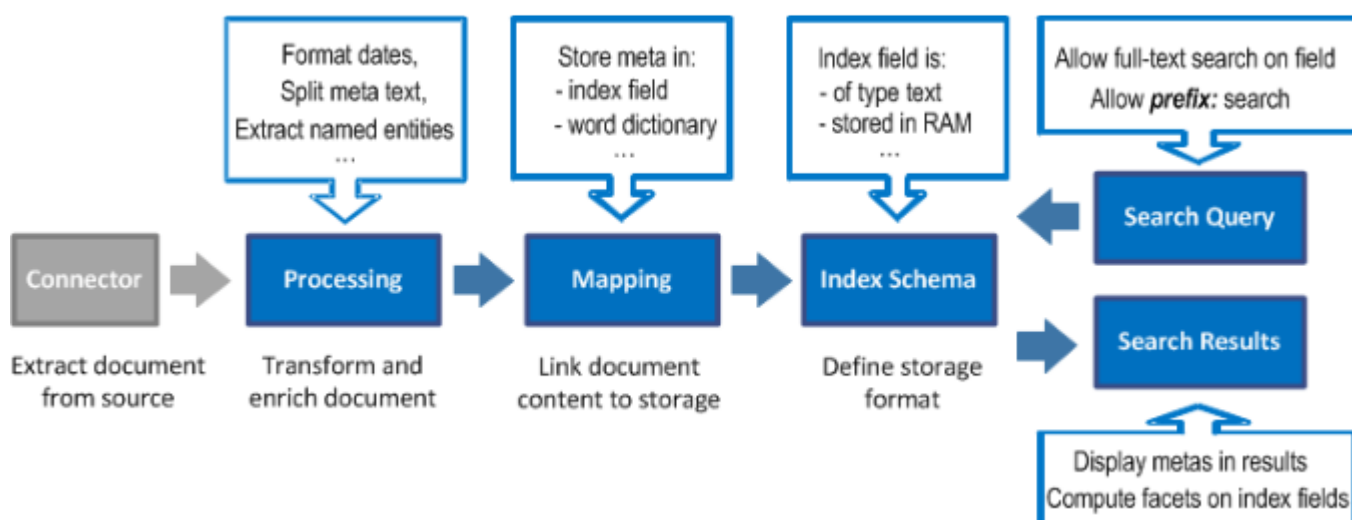
What is generated by the Data Model Expansion

When you apply a configuration, the high-level view provided by the Data Model is expanded into the multiple index and search elements.

The properties that you define generate:

- Normalization and semantic processing instructions (for alphanumeric fields or facets only). These instructions are contained in semantic processors that are automatically configured and added to the analysis pipeline.
- New dedicated index fields and category facets mapping document metas in the index schema.
- Storage settings of index fields in the index schema.
- Instructions saying whether to allow full-text search on index fields and search with prefix handlers. For example, the query `title: Recipes` with the `title: prefix` handler search for documents having `Recipes` in their titles).
- The corresponding hit meta and hit facets to display in the search results, and prefix handlers in the search logic.

Typical Data Model Expansion



For example, creating an alphanumeric property called `name` in a `customer` class expands into the following:

Phase	generates...	in...
Data Processing	A <code>customer_name</code> output context that you can manage to transform/normalize your data before indexation.	Data Processing > Document Processors/ Semantic Processors
Mapping	<p>A mapping from the <code>customer_name</code> output context to the <code>customer_name</code> field.</p> <p>If <code>sales</code> is not the default class, the output context name would instead be <code>customer_name</code>.</p> <p>If the <code>customer_name</code> property is to be both an index field and a facet, the data model creates two <code>customer_name</code> output contexts, with one mapped to the category, the other mapped to the index field.</p>	Data Processing > Mappings
Index schema	An index field of type text called <code>customer_name</code> .	Data model > Advanced Schema
Search Query	A prefix handler called <code>customer_name</code> that targets the <code>customer_name</code> index field.	Search Logics > Query Language
Search Results	A facet named <code>customer_name</code> .	Search Logics > Facets
	A hit meta named <code>customer_name</code> . At search time, this meta is automatically renamed to <code>name</code> so the class name (<code>customer</code>) does not appear in the hit content.	Search Logics > Hit Content

Controlling the Data Model Expansion

The Data Model simplifies the setup of the most common features for index fields and category facets. But you may need to control the default behavior of the Data Model expansion, or customize specific elements only.

Controlling the Overall Data Model Expansion

1. In the Administration Console, go to **Index > Data Model**.
2. Expand **Default expansion control**.

Default Expansion Control Options

Option	Description
Generate analysis config	Creates the required mappings and document processors for each property. Only clear this option to take complete control over analysis expansion.
Generate index schema	Creates the required dedicated index fields for each property set as Dedicated field . Only clear this option to take complete control over index schema expansion.
Generate facet	Creates a category facet in the search logics for each property set as Category facet . Only clear this option to take complete control over facet config expansion.
Generate hit meta	Creates the required hit metas in the search logics for each property set as retrievable . Only clear this option to take complete control over meta expansion.
Generate prefix	Creates the required prefix handlers in the search logics for each property set as searchable with prefix . Only clear this option to take complete control over query prefix handler expansion.
Analysis pipelines	Specifies the analysis pipelines (as a comma-separated list) for which mappings and documents processors are generated. If empty, they are generated for all pipelines.
Search logics	Specifies the search logics (as a comma-separated list) for which prefix handlers, facets, and hit metas are generated. If empty, they are generated for all search logics.

Customize a Specific Data Model Property

1. In the Administration Console, go to **Index > Data Model**.
2. Select the class for which you want to customize properties.
3. Click the property name to display its configuration details and expand the **Expansion control** section.

- a. To make options available, select **Customize default expansion**.
- b. You can now modify the configuration options for the selected property only. For more information, see [Controlling the Data Model Expansion](#).

Note: These options depend on the property configuration.

4. Click **Apply**.

Fully Override the Data Model Expansion for a Property

In some cases, you actually want to fully remove the generated elements. For example, you may want to remove a generated facet, because you want to store it but not display it at a given time.

1. In the Administration Console, go to **Index > Data Model**.
2. Select a property, then expand **Expansion control**.
3. Clear the appropriate **Generate...** option.

Customize a Specific Search Logic Element

1. In the Administration Console, go to **Search > Search Logics** tab that stores the element to customize.
2. Click an element name to display its configuration details.
 - The options are unavailable.
 - **Generated by Data Model** displays **Yes**.
3. To make these options available, click **Customize**.
 - The configuration options can now be modified.
 - To undo your changes, click **Restore**.
4. Click **Apply**.

Taking Control over Generated Index Fields

Sometimes, you may want to take control over index fields generated from data model properties or add new fields to store specific elements of your corpus.

This can be done by editing or adding fields in the **Data Model > Advanced Schema** tab.

For example, you can add binary retrievable fields to store binary data in the index. This can be useful if you want a dedicated thumbnail or preview generation and make it available directly inside the document. This also allows you to optimize the index storage.

1. In the Administration Console, go to **Data Model > Advanced schema**.
2. Click **Add field**.

- a. For **Name**, enter a descriptive name, for example, `thumbnails`
 - b. For **Type**, select **Binary**.
3. Click **Apply**.

Working with Data Model Classes

Can I Delete the Default `Document` Class?

For production deployments, it is best to delete the default `Document` class if you do not use it. See [How Classes Impact Performance](#).

Working with Multiple Classes

You can define multiple classes in the Data Model. You can also alter this by custom analysis code.

Classes can be independent, or hierarchized. In a hierarchy, a class inherits all of its parent's properties.

For example:

```
class A
  foo
  bar
class B: parentClass=A
  gizmo
```

This means you can search for `B_gizmo`, `B_foo`, `B_bar`, `A_foo`, or `A_bar`.

You can specify which class in the Data Model is the default class. When searching for properties from this class, you do not need to prefix the search by the class name. For example, if `B` was the default class, searching for `gizmo` returns the same result as searching for `B_gizmo`.

Impact of Multiple Classes on Performance

Adding multiple classes presents some limitations and has an impact on resource consumption. See [How Classes Impact Performance](#).

Using Properties to Configure Document Metas

A property helps to configure a document meta. It defines how to process it and store it as a dedicated index field. It is a functional layer over the technical configuration.

For example, if you want to search for the content of a specific meta, the property is going to store the meta as a dedicated index field and configure the search to enable you to do so.

Data Types and Semantic Types for Properties

Properties in a data model class have a **Data type**, such as alphanumeric, numeric, or geographic. The type, as well as other settings, determines what kind of elements you automatically create from the property when you scan your data sources.

Semantic types are assigned to all alphanumeric properties in a data model class, to determine the type of semantic processing applied to these properties at index time.

You can create the following types of properties:

- Alphanumeric
- Numerical: integer, unsigned integer, and double
- Date: date only and date-time
- Geographic: GPS point and XY point
- Measure

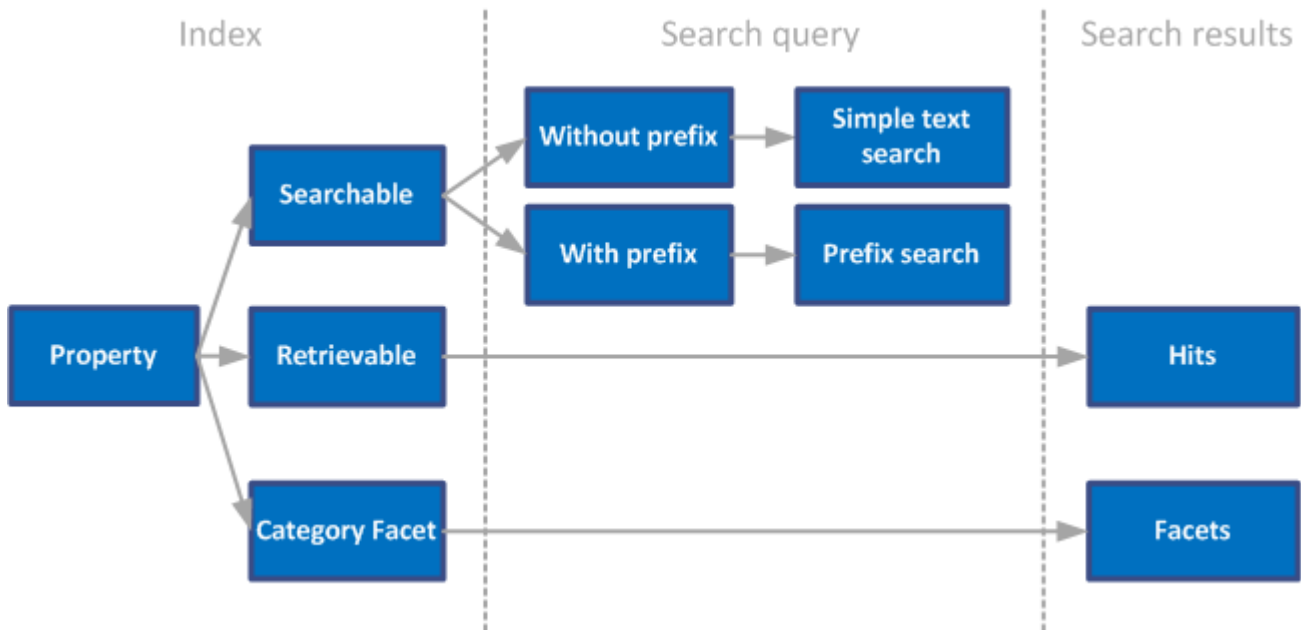
Indexing Options for All Properties

When creating a data model property, you can choose one of the following field types:

- a dedicated field only
- a category facet only (as opposed to virtual facets. For more information, see [Creating Facets to Refine Search Results.](#))
- both dedicated field and category facet
- none

This section explains the options available depending on the property type. The following schema represents it.

Indexing Options for Data Model Properties



Note: For an example explaining what is exactly created in the index schema and search logic, see [What is the Data Model Expansion](#).

Indexing Options for Index Fields

The most common indexing method is to assign a dedicated index field to the property. The property can then be:

- **Searchable** which means that user queries can be applied to this field. If the property is searchable with a prefix:
 - If this class is the default class, you can search for the property with a prefix format of `property:value`.
 - Otherwise, you can search for the property with a prefix format of `classname_property:value`.
- **Retrievable**, which means that the field can display in the search results. A hit meta with the property name and value appears in hit content.

Indexing Option for Facets

The property can also be stored as a **Category facet**.

- A category is created with the value of the property.
- Faceted navigation is automatically created for this property. In that case, it is possible to search for the value of the property.
 - If the property belongs to the default class, you can search for the property with a prefix format of `property:value`.

- Otherwise, you can search for the property with a prefix format of `classname_property:value`.

Note: If the property is both in a facet and in a dedicated field, search is handled through the field. The facet is then only used for faceted navigation.

Indexing Options for Dedicated Fields and Facets

The property can be made **searchable without prefix**, which means in addition to the indexing determined by other options, the property is indexed as **searchable** in the field called **text**.

This allows users to search this property without specifying a prefix in the query.

When this option is selected, the **Relevance class** select box displays, to let you adjust the ranking level of the index field. The value can be a standard ranking key from -1 to 8 (8 being the highest rank) or a custom value with a positive integer higher than 8.

Note: You can modify the ranking of search results afterward. For more information, see [Ranking and Sorting Search Results](#).

Not in a Dedicated Field or Facet

If the property is not in a dedicated index field or in a facet, or searchable in text, it can still be stored and retrieved, but not searched for.

Advanced Options

All properties have advanced parameters. For more details, see "AdvancedParams" in the Exalead CloudView XML Configuration Reference Guide.

Indexing Options for Alphanumeric Properties

When adding a new alphanumeric property, you must assign it a **semantic type**. It defines the global semantic processing to be applied when indexing document metas. You can then customize this semantic processing at the Analysis pipeline level (**Index > Data Processing > Semantic Processors**).

Exalead CloudView includes default semantic types, but you can modify them or create additional ones in **Data Model > Semantic Types**.

Default Semantic Types

Semantic type	Description
text	<p>Apply this semantic type to document metas containing several words or sentences.</p> <p>For example, you can apply this semantic type for a meta with a value like:</p> <p>"Deep Blue was a chess-playing computer. It is known for being the first piece of artificial intelligence to win both a chess game and a chess match against a reigning world champion under regular time controls."</p> <p>You must do it if you want to perform extra processing treatment in the analysis pipeline.</p>
metadata	<p>Apply this semantic type to document metas containing a few words that do not require language detection and spell-checking.</p> <p>For example, if we have the meta value <code>Deep Blue</code></p> <p>Searching for <code>deep blue</code> in lowercase would work, as well as searching for <code>deep</code> only or for <code>blue</code> only.</p>
identifier	<p>Apply this semantic type to document metas containing IDs.</p> <p>With this semantic type, document metas are searched in their normalized forms, but they cannot be tokenized and the language cannot be detected.</p> <p>For example, if we have the meta value <code>Deep Blue</code></p> <p>Searching for <code>"deep blue"</code> (with quotes) would work, but searching for <code>deep</code> only or <code>blue</code> only would not work.</p>
url	<p>Apply this semantic type to document metas containing web addresses.</p>

Indexing Options for Numerical Properties

When storing a numerical property as a facet, only equality search is possible. Range search is only possible in a dedicated index field. For more information, see [Numerical Range facets](#).

When storing a numerical property as a dedicated index field, **searchable with prefix** includes searching using the range operators.

Indexing Options for Date Properties

When adding a date property, you must define the **input format** that triggers the indexing of dates. You can leave the field empty for an automatic detection of standard formats or specify the date format to detect in the document corpus.

Exalead CloudView uses the UNIX date syntax to specify date and time formats for date fields in the data model. The same syntax is also used in the search logic for date metas displayed in the hit content, and dynamic date facets.

Note: There is also a time **output format** which can be set for each meta to display in the search results hits. Its default value is `%m/%d/%Y %H:%M:%S`. See **Search Logics > Hit Content > Metas** and expand, for example, the `lastmodifieddate` meta **Time format** operation.

Note: If a timezone is detected inside the date time value, it allows you to convert and store this value in UTC format. For more information, see [Specifying a Timezone for Date Time Metas](#).

UNIX Date Syntax

Specifier	Description	Values/Example
Day		
%a	weekday, abbreviated	Mon
%A	weekday, full	Monday
%d	day of the month (dd), zero filled	08
%e	day of the month (dd)	8
%j	day of year, zero filled	001-366
%u	day of week starting with Monday (1), that is, 7 (for Sunday) mtwtfss	
%w	day of week starting with Sunday (0), that is, 0 (for Sunday) smtwtfs	
Week		
%U	week number Sunday as first day of week	00–53
%W	week number Monday as first day of week	01–53
%V	week of the year	01–53
Month		
%m	mm month	08

Specifier	Description	Values/Example
%h	Mon	Aug
%b	Mon, locale's abbreviated	Aug
%B	locale's full month, variable length	August
Year		
%y	yy two-digit year	00–99
%Y	ccyy year	2014
%g	2-digit year corresponding to the %V week number	
%G	4-digit year corresponding to the %V week number	
Century		
%C	cc century	00–99
Date		
%D	mm/dd/yy	08/20/14
%x	locale's date representation (mm/dd/yy)	08/20/2014
%F	%Y-%m-%d	2014-08-20
Hours		
%l	hour (12 hour)	5
%I	hour (12 hour) zero filled	05
%k	hour (24 hour)	17
%H	hour (24 hour) zero padded	17
%p	locale's capitalize AM or PM (blank in many locales)	PM
%P	locale's lowercase am or pm	pm
Minutes		
%M	MM minutes	18
Seconds		

Specifier	Description	Values/Example
%S	seconds since 00:00:00 1970-01-01 UTC (UNIX epoch)	1345483096
%S	SS second	00–60 (The 60 is required to accommodate a leap second)
Time		
%r	hours, minutes, seconds (12-hour clock)	05:18:16 PM
%R	hours, minutes (24-hour clock)	17:18
%T	hours, minutes, seconds (24-hour clock)	17:18:16
%X	locale's time representation	11:07:26 AM
Date and Time		
%C	locale's date and time	Sat Nov 04 12:02:33 EST 1989

Indexing Options for Geographical Properties

Exalead CloudView supports two types of geographic coordinates:

- **GPS fields**, also called WGS84, store pairs of latitudes and longitudes separated by commas. They are expressed in decimal format, with an accuracy of 6 decimal places. Example: 37.818667, -122.478383 is a valid meta for GPS fields.
- **XY fields**, also called Meter, store pairs of integers separated by commas. As no unit is defined, you can consider the unit as meters, miles, or whatever unit you need. Example: 125, 8215 is a valid meta for XY fields.

You can select **Use separate metas for each coordinate** if latitude/ X and Longitude/ Y come from two different metas in your data source. Exalead CloudView concatenates the two coordinates to store them in a single index field.

Indexing Options for Measure Properties

To handle a meta-data that contains a measure, add a property in your data model with a **Measure** data type. Then specify the **unit symbol** of your measure; this works as indexing unit and also as default input unit.

By default, the expansion control configuration automatically specifies all required elements for the good indexing & search of this new property:

- a **Units of Measurement Normalizer** document processor in the **Data Processing > MODEL_NAME > Document Processors**.
- a measurement prefix handler with the name of your measure property in **Search Logics > SEARCH_LOGIC > Query Language**.

Creating Dynamic Properties

A dynamic property allows you to map one or several metas to the same index field, based on the meta name.

This is useful when you do not know all the metas available in your data source. It also reduces the number of fields you need in the index.

For example, use dynamic properties to:

- Map all meta names that start with the word `item` to an `item` field, when indexing a corpus of store inventory.
- Map any meta name that includes the word `price` to a `price` facet, when indexing sales data.

Note: Dynamic properties can generate alphanumeric fields, numerical fields, date/time fields, or category facets. When using date/time dynamic properties, the validation of the date/time format is performed according to the format specified for this dynamic property.

Add a Dynamic Property

About Storing and Displaying Dynamic Property Fields

Store Properties in a Parent Class Dynamic Property

Search Dynamic Property Fields

Add a Dynamic Property

The following procedure describes how to configure a dynamic property for a data model class.

1. In the Administration Console, go to **Index > Data Model**.
2. Under **Classes**, select the class (if the property is not part of the default class).
3. Under **Dynamic properties for the <Classname> class** (below), click **Add dynamic property**.
 - a. Select a **Name** and **Data type**.
 - b. Select a **Semantic type**. This defines the type of semantic processing for the property.
 - c. Select a **Field type**.
4. Define **Matching rules**:

For **Mode**, select a matching method:

- **Exact**: includes the specified meta name only.
- **Prefix**: includes meta names that begin with this string.

Note: You can select the **Unprefix** check box if you want to remove the prefix from search queries.

- **Suffix**: includes meta names that end with this string.
- **Substring**: includes meta names that contain this string.
- **Pattern**: includes meta names that match this regular expression.

For **Pattern**, specify the string or regular expression.

5. Click **Accept**.
6. (Optional) To display meta values under their meta name in hit results or the Refinements panel, select **Store meta names**.

Note: Selecting this option impacts how you search with dynamic property prefix handlers. See [Search When Storing Meta Names](#).

7. (Optional) Select or clear the remaining options as required.
8. Click **Apply**.
9. Go to **Home > Indexing**, click **Clear**, and then select the build groups that are modified by this property.
10. Scan your documents.

About Storing and Displaying Dynamic Property Fields

The way in which index fields and category facets are stored is determined by the **Store meta names** option, which also determines how these fields can be retrieved and searched.

For details on searching, see [Search Dynamic Property Fields](#).

If Store Meta Names is not Selected

If **Store meta names** is not selected for a dynamic property, the resulting dedicated index field (or category facet or output context) is the same as any other alphanumeric or numerical field created with standard properties. It contains only meta values.

Without Stored Meta Names

Note: The `store` index field includes both the `store_city` and `store_country` metas, and displays the values for both together.

11 units of Nicer swimming suit - bought by Jacob Patton				Download	Preview
category	swimming-suit	color	multiple		
description	Don't miss this before beach time.	firstname	Jacob		
lastname	Patton	quantity	11		
unit_price	34.99	name	Nicer swimming suit		
store	Hong-Kong, China				
id: id=11&					

If Store Meta Names is Selected

If **Store meta names** is selected, by contrast, the resulting index field (or category facet or output context) is indeed different from standard fields: for each value, it also stores the associated meta name. This allows you to search and retrieve specific metas within the field.

With Stored Meta Names

Note: The `Store` index field displays the `store_city` and `store_country` metas separately.

11 units of Nicer swimming suit - bought by Jacob Patton				Download	Preview
store_city	Hong-Kong	store_country	China		
category	swimming-suit	color	multiple		
description	Don't miss this before beach time.	firstname	Jacob		
lastname	Patton	quantity	11		
unit_price	34.99	name	Nicer swimming suit		
id: id=11&					

Calculate Facets for Dynamic Property Fields

Dynamic properties, are not stored in the `Categories` index field with a `/top/foo/bar` tree structure like standard facets. To calculate their values, use virtual expressions, with the following syntax:

`#extract(dynamicField, "<meta name>")` to return the value of "meta name" in the dynamic field.

Store Properties in a Parent Class Dynamic Property

For big projects with a lot of fields, you can mutualize dynamic fields if they can be used by different classes.

The idea is to store the properties of child classes into a single dynamic property specified for a parent class. For example, we could configure our data model to have:

parent_class with:

- dynprop_num_ram
- dynprop_alpha_search
- ...
- common_prop1
- common_prop2

and child_class1 with:

- prop1 --> in dynprop_num_ram (parent_class)
- prop2 --> in dynprop_alpha_search (parent_class)
- ...
- prop3 (dedicated to this class)

1. In the Administration Console, go to **Index > Data Model**.
2. Select a property, then expand **Other advanced options**.
3. In **Store in dynamic property**, enter the name of the dynamic property in which you want to store the current property.
4. To avoid getting duplicate values for your children meta at search time:
 - a. Expand **Expansion control**.
 - b. Select **Customize default expansion**.
 - c. Clear **Generate hit meta**.
5. Click **Apply**.

Search Dynamic Property Fields

The **Store meta names** option also determines how you search the resulting index field or category facet using prefixes.

Search When Storing Meta Names

1. Include the meta name in the prefix handler, in this format:

- If the property belongs to the default class:
`<property_name>:<meta_name>:<query>`.
- If the property does not belong to the default class:
`<classname_property_name>:<meta_name>:<query>`.

For example: `item:item_description:skirt`

Searches for `skirt` in the `item_description` meta of the `item` index field.

Search When not Storing Meta Names

1. Use only the property name for the prefix handler, in this format:
 - If the property belongs to the default class: `<property_name>:<query>`.
 - If the property does not belong to the default class:
`<classname_property_name>:<query>`.

For example: `item:skirt`

Searches for `skirt` in the `item` index field, which includes the metas `item_description`, `item_name`, and `item_details`.

Search When the Dynamic Property is in a Parent Class

1. To target a property stored in a dynamic property of a parent class, you must specify the parent class and the child class in your prefix handler:
`<parentclassname_dynamic_property_name>:<childclassname_meta_name>:<query>`.
The following syntax does not work:
`<parentclassname_dynamic_property_name>:<meta_name>:<query>`.

Creating Multivalued Properties

You can set all types of data model properties as multivalued. This is useful when you need to index several values in a single index field.

For example, you want to store several email addresses in an `email_address` field or phone numbers in a `phone_numbers` field for persons:

1. In the Administration Console, go to **Index > Data Model**.
2. Expand a property to view its configuration options and expand **Other advanced options**.
3. Select **Multivalued** and click **Apply**.

Tools to Create a Data Model from Your Corpus

A common challenge when creating a Data Model is to know which metas are in your corpus. There are several ways you can explore your available metas using the Exalead CloudView Data Model.

Create a Data Model from Sample Documents

To save time when creating properties, you can use the **Trace all metas** option.

Important: Every time you scan your sources, this option saves all these metas to an internal database.

Recommendation: Disable **Trace all metas** after you have generated the properties you need.

1. In the Administration Console, go to **Index > Data Model**.
2. Under **Data model options**, select **Trace all metas**
3. Click **Apply**.
4. Go to the **Home** page, click **Scan** for the connector.

This saves all scanned metas to an internal database.

5. Go back to **Index > Data Model > Classes**, click **Add properties from traced metas**.

This enables you to select multiple metas to save as properties and optionally, index fields or category facets.

For an example of creating properties from traced metas, see "Create a new class in the data model" in the Exalead CloudView Getting Started Guide.

6. Click **Generate properties**.
7. Click **Apply**.
8. As you have changed the Index Schema by adding new properties, you need to clear the documents in the build group and re-index your data:
 - a. On the **Home** page, under **Indexing**, click **Clear**. Wait for the index to clear its documents.
 - b. Under **Connectors**, click **Scan** next to your connector.

Store Unprocessed metas

You can save unanalyzed metas to a single csv-encoded output context called `metas`, and then display them in hit content.

Important: This is a legacy option. Save unanalyzed metas using a dynamic property to store them in a dedicated index field (see [Creating Dynamic Properties](#)).

Recommendation: It is more convenient to search for each meta individually.

1. In the Administration Console, go to **Index > Data Model**.
2. Under **Data model options**, select **Store all unprocessed metas**.
3. Click **Apply**.

These metas are retrievable only. You cannot search them. To search on one of these metas, you must create a Data Model property for it.

Configuring Data Processing

Data Processing refers to the processing performed on documents, starting from the pushing of documents in the Push API to the storing of the resulting output as fields and categories into the index.

In most situations, you can use the data model to create index fields and handle data processing. For more complex processing needs, however, you may need to create fields or customize processing by directly modifying the document analysis pipeline and index schema (see [Taking Control over Generated Index Fields](#)).

[Understanding and Using the Analysis Pipeline](#)

[Testing your Analysis Pipeline Behavior](#)

[More About Semantic Analysis](#)

[Tokenizing Text](#)

[Creating and Deploying Semantic Resources](#)

[Managing Semantic Annotations](#)

[Configuring Form Indexing](#)

Understanding and Using the Analysis Pipeline

This section describes the main phases involved for data processing in the analysis pipeline, how to use pipeline conditions and how to configure the pipeline manually.

[About Data Processing](#)

[The Analysis Pipeline Sequence of Processors](#)

[Use Multiple Pipelines with Conditions](#)

[Use a Single Pipeline with Groups of Processors](#)

[Multiple Pipelines vs. Single Pipeline with Groups](#)

[Configuring the Analysis Pipeline Manually](#)

About Data Processing

About the Overall Document Lifecycle

This section explains the document lifecycle through the various components of the Indexing Server.

In the Push API Server

When connectors send documents to the Indexing Server, they first arrive into the Push API server (PAPI server) of the Indexing Server.

When you push a document to the Push API Server, it contains:

- document URI
- document stamp to indicate the version
- meta data
- parts containing the bytes from the document
- directives for data extraction

In the Analysis Pipeline

Documents are then pushed to the analysis pipeline of the Indexing server.

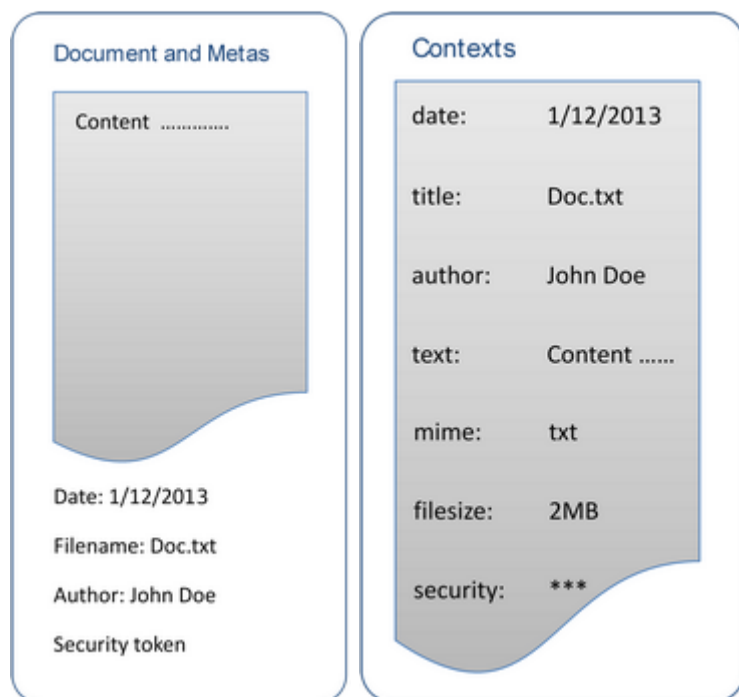
The first step is to transform the content of the document parts and the metadata items into internal items that the document and semantic analysis processors can process. These internal items are called document chunks.

Note:

- Each document chunk is tagged as a context which name is case-sensitive.
- Created contexts depend on the extraction process. By default, it creates the `text` and `title` contexts for each part, and a context for each metadata item.

The following example shows the mapping of a document with one part and metadata items to several contexts.

Document and Metadata Mapped to Contexts



Once the document is represented as contexts with one or more document chunks, the analysis processors can process it. The processors can perform one of the following:

- create new contexts
- transform existing contexts

If you want to perform semantic analysis, you must tokenize the context. The semantic processor can create annotations for the tokens. The following example shows a possible representation of the document after analysis.

Contexts and Annotations after Document and Semantic Processing

Contexts		Annotations	
date:	1/12/2013		
title:	Doc.txt		
author:	John Doe		
text:	Content	NE:people:	John Doe Jane Smith
security:	***	NE:location:	England
mime:	txt		
language :	en		
filesize:	2MB		

In the Index

The final phase is the mapping of the contexts and the annotations to index fields so that the document may be used for search.

When a document matches a query the results contain hit fields, metadata (that can be different from the Push API metadata), categories, and related terms.

Document Mapped to Index Fields

Index Fields	
1/12/2013	lastmodifieddate categories:Top/Date/Lastmodified
Doc.txt	title
John Doe	categories:Top/Person/Author
Content	text
John Doe	categories:Top/Person
Jane Smith	
England	categories:Top/Location/
***	security:Top
txt	categories:Top/Mime
en	categories:Top/Language
2	file_size

Focus on Data Processing Phases in the Analysis Pipeline

Data processing involves the following phases in the Exalead CloudView analysis pipeline.

#	Phase
1	Document Processing provides a set of analysis filters that are able to modify the content of the documents for indexing. For example, the document's language and MIME type are automatically detected, and it is possible to use the result of this detection as a document category.
2	Contexts appear at the end of the document processing phase. You can find new or transformed content that can be mapped in the final phase to fields in the index.
3	Semantic Processing provides a set of semantic processors to detect related terms, perform semantic query processing, categorization thesaurus, extractions, and ontology matching.
4	Annotations provide additional information about a piece of text in the form of names, attributes, descriptions, and so on. They are attached to documents by semantic processors during analysis. To search on an annotation, map to an index field. To use for a facet, map to a category.
5	Use Content and Annotation Mapping to send data from multiple metas of the documents to the same field, or to send a given part of the document to multiple index fields, with multiple options. You can find in the index fields the heterogeneous content and annotations gathered from the connectors and created during the document analysis

The Analysis Pipeline Sequence of Processors

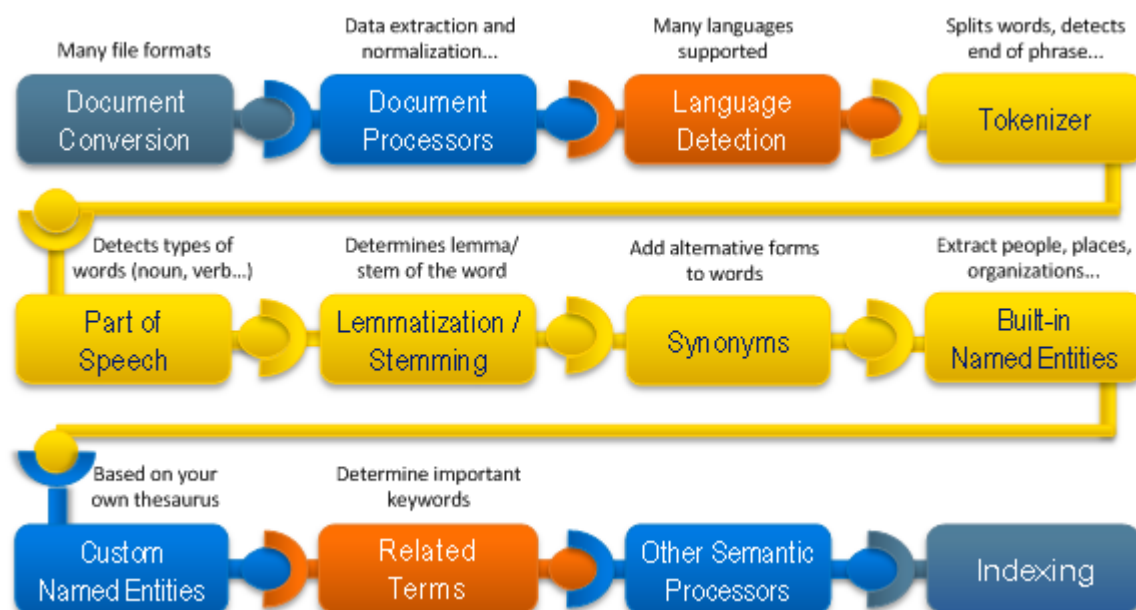
The Analysis pipeline defines a sequence of processors executed on a document before it is added to the index. These processors are:

- document processors, used for transforming document meta and content,
- and semantic processors, used for extracting structured information from unstructured document content.

The analysis pipeline processes documents one by one in a specified sequence determined by the order of the document and semantic processors. For each input document, the pipeline outputs exactly one document.

You can access the analysis pipeline in the Administration Console by going to **Index > Data processing > Analysis pipelines**, and selecting the pipeline name.

The following figure shows the typical analysis pipeline workflow.



Use Multiple Pipelines with Conditions

In the data processing configuration, you can have several analysis pipelines. Multiple pipelines allow for different processing depending on the document. Each pipeline has an associated condition, to determine which pipeline processes the document.

The order of document pipelines in the data processing configuration is important. As soon as a document pipeline is found with a valid accept condition, the document is dispatched to this pipeline.

1. Go to **Index > Data processing > Analysis pipelines**.
2. Expand the **Condition** section of an analysis pipeline, for example **ap0**.
3. Click **Add condition**.
4. Set the analysis pipeline condition.

Condition applies to...	If...
Metadata	<p>The document metadata name equals or matches the specified name, and if its value:</p> <ul style="list-style-type: none"> • Exists • Equals the specified value • Contains the specified value • Matches the specified value • Does not exist • Is not equal to the specified value

Condition applies to...	If...
	<ul style="list-style-type: none"> Does not contain the specified value Does not match the specified value
Custom Directive	The document contains the specified custom directive set to the specified value.
Data model class	<p>The document belongs to the data model class specified in your connector. This is controlled by the Connector > Configuration > Store documents in data model class parameter set on your connector.</p> <p>If your connector does not specify a default data model class, this condition does not work.</p>
Document source	The document comes from the specified connector source.
URL match	The document URL does or does not match the specified regular expression.

5. Test that your condition works as expected. See [Testing your Analysis Pipeline Behavior](#).
6. Add as many conditions as required.
7. Click **Save**.

Use a Single Pipeline with Groups of Processors

To avoid multiplying pipelines and therefore simplify the administration of the data processing configuration, you can also choose to group processors sharing a same trigger condition (typically on document source) within a same pipeline.

Using a single pipeline with collapsible groups of processors and sets of conditions is much easier to control and maintain.

Important: Some processors have dependencies, so be careful to make consistent groups including these dependencies. For example, the **html relevant content extractor** depends on the **native text extractor**, and both must therefore be included in the group to get a correct processing.

1. Go to **Index > Data processing > Pipeline Name > Document Processors**.
2. Drag the **Other > Document Processor Group** processor to the pipeline.
This processor acts as a container for other document processors.
3. Drag other document processors into the **Document Processor Group** processor.

You can sort these processors as required using the up and down arrows.

Note: To create subgroups, drag a **Document Processor Group** processor within the **Document Processor Group** container.

4. Optionally, you can add triggering condition on the:
 - **Document Processor Group** processor so that its list of processors is executed when the specified condition is met (if any).
 - Processors contained in the **Document Processor Group** processor.

Note:  Icons flag processors with conditions.

Important: Document processing performance is linear depending on the number of conditions used in the pipeline. Use **Equals** conditions instead of **Matches** conditions in the **Add condition** dialog box.

5. Click **Save**.

Multiple Pipelines vs. Single Pipeline with Groups

Use a single pipeline when you have several document sources sharing common document processing and mappings.

For example, several filesystem sources with the same data model classes (title, text, Author, etc.).

Use multiple pipelines when indexing document sources that do not share the same data model classes.

Note: Each analysis pipeline has a fixed memory consumption regardless the number of processors. Having 2 pipelines with 10 processors each, uses therefore more memory than having 1 pipeline with 20 processors.

Configuring the Analysis Pipeline Manually

This section outlines the high-level procedure to configure an analysis pipeline with both document and semantic processors, and to display the output in the search logic.

To map data to index fields and categories, you must define:

1. The document processors to generate output contexts, or metas, based on the name specified in the **Output to** box.
2. The semantic processors to generate annotations.

3. Map all contexts and annotations to index fields or categories.

Before performing the mapping, you must create the target index fields under **Index > Data Model > Advanced Schema**.

You can then configure in **Index > Data Processing > Mappings**, a list of:

- metas and parts mapping document contexts to index fields
- annotations mapping document annotations to index fields

Important: All changes made to the analysis pipeline require clearing the index and reindexing data to avoid inconsistencies.

Map Document Processors Manually

1. Add document processors to the list of current processors, and define the output contexts (or metas) for them.
 - a. Go to **Index > Data processing > pipeline name > Document Processors**.
 - b. Drag the appropriate processor to the list.
 - c. Configure the processor.
2. Create index fields to store the contexts.
 - a. Go to **Index > Data Model > Advanced Schema**.
 - b. Click **Add Field**.
3. Define mappings for the new context generated by the document processors.
 - a. Go to **Index > Data processing > pipeline name > Document Processors**.
 - b. Select the **Mappings** subtab.
 - c. Click **Add mapping source** and create a mapping with the same name as the output context.
 - d. Click **Add mapping target** and select either **Index field** or **Category field** (for facet).
 - e. Under **Details**, configure indexing options for the index field or category field. The following table describes the most important ones.

Option	Description
Searchable	Preindexes the data for efficient lookup. This allows users to search for documents based on the available values in this index field.
Retrievable	Displays the content of the field in the hit content of the search results.
Indexing options	For each target field, you can configure the form of words to be indexed. Choose one of these: <ul style="list-style-type: none"> • index exact form

Option	Description
	<ul style="list-style-type: none"> index lowercase form index normalized form (this removes accents and specifies the contents to lowercase) <p>You can also choose to index the position of separators to enable search within a string. Select this option for use with a "split" type prefix handler.</p>

Map Semantic Processors Manually

- Drag semantic processors to the list of current processors
 - Go to **Index > Data processing > pipeline name > Semantic Processors**.
 - Drag the appropriate processor to the list.
 - Configure the processor.
- Define mappings for the new annotations generated by the semantic processors.
 - On the **Mappings** subtab, select **Add mapping source**.
 - To create annotation mappings, follow [Map Document Processors Manually](#) > Step 3.
- To display category or index fields in the search results:
 - To display categories as facet in the search results, add a facet to your search logic.
 - To display index fields in the hit content of the search results, create a hit meta (or add the field to an existing hit meta) in the search logic.
- Click **Apply**.
- Clear the index and reindex your data.

Set Mapping Limits for Index Fields

You can configure a list of field retrieval limits to define the size limits for the index fields.

- Go to **Index > Data Processing > Mapping Limits**.
- In **Max indexed words**, specify the maximum number of words that are searchable for the field.

For example, if you index 500 words in this field, but set a limit to 100, then only the first 100 words are searchable.
- In **Max size stored for display**, specify the maximum number of bytes that are retrieved for the field.

Testing your Analysis Pipeline Behavior

To reduce index configuration time or for debugging purposes, you can submit a document to the analysis pipeline and see how it is modified.

You can track changes for:

- an indexed document by creating a connector and running a full scan.
- a new custom document pushed to the index.

For each document, you can display specific document processing information.

[Test the Analysis Pipeline with an Indexed Document](#)

[Test the Analysis Pipeline with a New Custom Document](#)

[Display Document Processing Information](#)

[Test the Semantic Processing of your Analysis Pipeline](#)

Test the Analysis Pipeline with an Indexed Document

In this example, we index the PDF guides of the Exalead CloudView documentation. We change the Author name displayed in the hit content of search results. Instead of 'EXALEAD R&D', we want to display 'EXALEAD'.

Add a Files Connector and Index the Documentation

You can add a new connector that includes the Exalead CloudView functional and reference product documentation. For this example, we keep most of the default parameters.

1. Open the Administration Console: `http://<HOSTNAME>:<BASEPORT+1>/admin`
2. Go to **Index > Connectors** and click **Add connector**.
3. Complete the fields as follows:
 - a. For **Name**, type `Functional_guides`
 - b. For **Creation mode**, select **new**.
 - c. For **Type**, select **Files**.
 - d. Click **Accept**.

The **Functional_guides** connector page is created.

Note: When creating connectors, always use intuitive names as by default, they appear as navigation facet in the Mashup UI.

4. On this connector **Configuration** page, go to the **Filesystem paths** section and enter the path:
 - for UNIX platforms, /<INSTALLDIR>/docs/pdf
 - for Windows platforms, <INSTALLDIR>\docs\pdf
5. Click **Apply**.
6. Go to the **Home** page and under the connectors list, click **Scan** next to the **Functional_guides** connector.

This triggers the indexing.

7. Go to the Mashup UI: <http://<HOSTNAME>:<BASEPORT>/mashup-ui> and enter #all in the search field.

Search results are displayed.

Note: for each hit the Author meta displays **Exalead R&D**.

Hit with Author Meta Displaying Exalead R&D

Results › 1-10 of 11
Sort by Relevance Date Size

EXALEAD CloudView Getting Started Guide

Download
Preview

CloudView R2015x GETTING STARTED GUIDE This tutorial explains how to create a search application that demonstrates CloudView's most popular index and search capabilities.

File path /data/ [redacted] /cloudview-dev_br_V6R2015x.dev.71080-linux-x64/docs/pdf/CloudView_GetStartedGuide_EN_R2015x.pdf

File size 4452116

File name CloudView_GetStartedGuide_EN_R2015x.pdf

Source Functional_guides

Author Exalead R&D

Language English

Last modification 2015 > 03 > 13

Data model class document

File extension pdf


Document type PDF document

id: /%2Fdata%2F [redacted] %2Fcloudview-dev_br_V6R2015x.dev.71080-linux-x64%2Fdocs%2Fpdf/CloudView_GetStartedGuide_EN_R2015x.pdf



Define the New Author Name

Once PDFs are indexed, the first thing you need to define is the new Author name to be displayed.

1. In the Administration Console, go to **Index > Data Processing > Edit > Pipeline name (e.g. ap0) > Document Processors**.
2. In **Processor types**, search for the **Replace Values** processor and add it above the **rename_extracted_author** processor.
3. Click the  icon and rename this new processor: **rename_exalead**.
4. Define the following parameters:
 - a. **Input from:** `extracted_author`
 - b. **String to replace:** `EXALEAD R&D`
 - c. **Replacement string:** `EXALEAD`
5. Click **Apply**.

Test the Document Meta Processing

You can process a document to see how the modification is taken into account.

1. In the Administration Console, go to **Index > Data Processing > Test**.
2. In **Select an indexed document**, choose:
 - a. your connector's name in **Source:** `Functional_guides`
 - b. the document to be processed in **Document URI:**
`CloudView_GetStartedGuide_EN_R2015x.pdf`
3. Select your analysis pipeline (by default `ap0`).
4. Click **Process**.

The list of generated metas and annotations is displayed on the right. You can see that the Author has been properly renamed 'EXALEAD'. Metas have all been newly generated, that is why they are displayed in green.

Analysis pipeline ap0

Output document

	Name ↕	Value	Operation
M	analysisdate	2015/03/30-13:43:53	Added
M	author	EXALEAD	Added
M	dataModelClass	document	Added
M	dataModelClassHierarchy	document	Added
M	docsrc	pdf <input type="checkbox"/> (2 values)	Added
M	extracted_comments	CloudView installation and configuration for new users	Added
M	extracted_copyright	© Dassault Systèmes,... © Dassault Systèmes,... (2 values)	Added
M	extracted_creationDate	Sun, 13 Mar 2015 17:... Fri, 13 Mar 2015 17:... (2 values)	Added
M	extracted_creator	EXALEAD R & D FrameMaker 12.0.4 (2 values)	Added
M	extracted_height	842	Added
M	extracted_keywords	EXALEAD CloudView EXALEAD CloudView (2 values)	Added
M	extracted_mime	application/pdf	Added
M	extracted_modificationDate	Sun, 13 Mar 2015 17:... Fri, 13 Mar 2015 16:... (2 values)	Added
M	extracted_numpages	91	Added
M	extracted_numproducedpages	91	Added
M	extracted_pixel-aspect-ratio	1.41460300393132 (2 values)	Added

⏪ ⏩ Page 1 of 5 ⏪ ⏩

Now you can take a closer look at what happened in the analysis pipeline.

Display the Analysis Pipeline Details

1. Look for the `extracted_author` meta using the **Filter metas** search filter on the right.
2. Select the analysis pipeline name (for example `ap0`) to expand it.

Only document and semantic processors using the `extracted_author` meta are active. You can see the two processors involved when renaming and extracting the Author.

Analysis pipelines

ap0 - Analysis pipeline

Document processors (20)

mime_detector - MIME Detector
 native_textextractor - Text Extractor (text,html,exalead)
 convert_textextractor - Text Extractor (all mime types)
 dataModelClassResolver - DataModelClassResolver
 context_metas_renamer - ReplaceContextNames
 select_numpages - Value selector
 standard_parts_merger - Standard Parts Merger
 html_relevant_content_extractor - HTML Relevant Content Extrac
 select_title - Value selector
 current_date - Insert Current Date
 rename_exalead - Replace Values
IF rename_extracted_author - Rename Context for Chunks
 split_file_path - Split Values
 replace_extracted_mime - Replace Regexp
 concat_source_path - Concatenate Values
 document_lastmodifieddate_formatter_lastmodifieddate - Fon
 document_pageurl_urlproc_pageurl - Public Url Processor
 concat_source_crawler_path - Concatenate Values
 csv_encoder_metas - Multi-Context Encoder
 language_detector - Language Detector

Semantic processors (1)

Notice the green icon before the **rename_exalead** processor. It means that the condition defined for this processor is met.

3. Select the **rename_exalead** processor.

You can see that the `extracted_author` meta is 'EXALEAD'. This meta has been modified, that is why it is displayed in blue.

Document processor rename_exalead

Output document

Name	Value	Operation
 <code>extracted_author</code>	EXALEAD	Modified

Processing time: 14µs

4. Select the **rename_extracted_author** processor.

You can see that the `extracted_author` meta has been removed. The final name is still 'EXALEAD'.

Document processor rename_extracted_author

Output document


Name	Value	Operation
M extracted_author		Removed

Processing time: 1042µs

- Go to the **Home** page and click **Clear documents** from the **Functional_guides** connector.
- Click **Scan**.
- Once the scan is complete, go back to the Mashup UI page and refresh the view.

The Author meta now displays **EXALEAD**.

EXALEAD CloudView Getting Started Guide
Download
Preview

CloudView R2015x GETTING STARTED GUIDE This tutorial explains how to create a search application that demonstrates CloudView's most popular index and search capabilities.


File path	/data/functional_guides/cloudview-dev_br_V6R2015x.dev.71080-linux-x64/docs/pdf/CloudView_GetStartedGuide_EN_R2015x.pdf	File name	CloudView_GetStartedGuide_EN_R2015x.pdf
File size	4452116		
Source	Functional_guides	Data model class	document
Author	EXALEAD	File extension	pdf
Language	English	Document type	PDF document
Last modification	2015 > 03 > 13		

id: /%2Fdata%2Ffunctional_guides%2Fcloudview-dev_br_V6R2015x.dev.71080-linux-x64%2Fdocs%2Fpdf/CloudView_GetStartedGuide_EN_R2015x.pdf

Test the Analysis Pipeline with a New Custom Document

You can push new test documents to the index.

- In the Administration Console, go to **Index > Data Processing > Test**.
- Select **Create a custom document** and click **Edit document**.

The **Edit** dialog box is displayed.

3. Enter the unique document identifier in **URI** and the time **stamp** (optional).

For example, `myuri` and `2015/03/28-08:00:00`.

4. In the **Metas** section, set the meta details for the document in **meta name** and **value**.

For example, enter `department` as meta name and `marketing` as value.

5. Click **Add meta** to create new metas.
6. Click **Upload file** to select your document.

Your document is displayed with the name `master`. Click this name to display:

- the **filename**. For example, `doc`.
- the **encoding** type. For example, `UTF-8`.
- the **mimeHint**. For example, `text/richtext`.



7. Click **Close**.
8. Click **Process**.

You can now test the analysis pipeline using the displayed document processing information.

Display Document Processing Information

Display Meta Processing

The following elements are displayed for each document processor and semantic processor:

- Meta operation statuses: Added, Modified, Removed.
- Processor conditions:
 - met 
 - not met 
- Metas excluded from the index (**Index mappings**)

Analysis pipelines

ap0 - Analysis pipeline

Document processors (19)

mime_detector - MIME Detector
native_textextractor - Text Extractor (text,html,exalead)
convert_textextractor - Text Extractor (all mime types)
dataModelClassResolver - DataModelClassResolver
context metas renamer - ReplaceContextNames
select_numpages - Value selector
standard_parts_merger - Standard Parts Merger
html_relevant_content_extractor - HTML Relevant Content Extrac
select_title - Value selector
current_date - Insert Current Date
if rename_extracted_author - Rename Context for Chunks
split_file_path - Split Values
replace_extracted_mime - Replace Regex
concat_source_path - Concatenate Values
document_lastmodifieddate_formatter_lastmodifieddate - Dat
document_pageurl_urlproc_pageurl - Public Url Processor
concat_source_crawler_path - Concatenate Values
csv_encoder_metas - Multi-Context Encoder
language_detector - Language Detector

Semantic processors (0)

Index mappings

Document processor rename_extracted_author

Output document

Name	Value	Operation
author	EXALEAD R&D	Added
extracted_author		Removed
analysisdate	2014/01/31-08:52:17	
dataModelClass	document	
dataModelClassHierarchy	document	
displayurl	/data/ user /icloudview-dev_trunk.dev.59845-linux-x64/docs	
docsrc	pdf (2 values)	
extracted_comments	CloudView installation and configuration for new users.	
extracted_copyright	? Dassault Syst?mes,... ? Dassault Syst?mes,... (2 values)	
extracted_creationDate	Sun, 27 Nov 2013 14: ... Wed, 27 Nov 2013 14: ... (2 values)	
extracted_creator	EXALEAD R & D FrameMaker 11.0.2 (2 values)	
extracted_height	842	
extracted_keywords	Administration Conso... Administration Conso... (2 values)	
extracted_mime	application/pdf	
extracted_modificationDate	Sun, 27 Nov 2013 14: ... Wed, 27 Nov 2013 13: ... (2 values)	
extracted_numpages	80	

Page 1 of 4

Display Annotation Processing

When clicking a meta name for a semantic processor, you can display:

- Annotations
- Tags, including
 - Available forms
 - Count

Display Internal Properties

You can also display parts and directives.

Details of the master part related to the **mime_detector** document processor:

Document processor mime_detector

Part master

[Back to the list](#)Content length: 4.2MB ([download](#))

Detected MIME: application/pdf

Detected encoding: utf-8

Part directives

Name	Value
filename	CloudView_GetStartedGuide_EN_R2015x.pdf
encodingHint	UTF-8
mimeHint	application/pdf

Disable Processors and Options

You can change document and semantic processors anytime in the **Edit** tab. For example, you may need to disable processors or disable the document cache.

Disable a Processor

1. In the **Edit** tab, select your processor.
2. Select the **Disable processor** check box.
3. Display the **Test** tab.
4. The processor is grayed and indicated as `(disabled)`.

Disable the Document Cache

If document cache is enabled for your build group in **Deployment > Build groups**, the **Use document cache** option is automatically enabled when testing your analysis pipeline. You can disable it to test the latest modified version of your document.

For more information, see "Document cache" in the Exalead CloudView Administration Guide.

Test the Semantic Processing of your Analysis Pipeline

When you add document or semantic processors to your pipeline, you may want to see its output.

To do so, you can use the `semantic annotate` function of the `cvdebug` command-line tool, located in the `<DATADIR>/bin` directory.

```
cvconsole cvdebug > semantic annotate [args]
```

Where possible arguments `[args]` are:

- `[buildGroup]` – Build group name (default: `bg0`)
- `[context]` – Context of the chunk (type: `STRING`)
- `[language]` – ISO code of the language (type: `ISO_CODE`)
- `[pipeline]` – Analysis pipeline to use (default: `ap0`)
- `[value]` – (Required) Text to process, standard input if missing (type: `STRING`)

Example:

Consider that our analysis configuration contains only one pipeline. This pipeline contains a single semantic processor, the Named Entities Matcher. This processor provides Named Entities annotations.

We start the `semantic annotate` function to test the Named Entities Matcher with the following textual input.

```
cvconsole cvdebug > semantic annotate value="Bill Keller and Barack Obama" language=en
```

Applying this command gives the following XML output for the first three tokens.

```
<AnnotatedToken token="Bill" kind="ALPHA" lang="en" offset="0">
  <Annotation displayForm="bill" displayKind="lowercase" tag="LOWERCASE" nbTokens="1"
  <Annotation displayForm="bill" displayKind="normalized" tag="NORMALIZE" nbTokens="1"
  <Annotation displayForm="" displayKind="exact" tag="exalead.nlp.firstnames" nbToken
trustLevel="0" />
  <Annotation displayForm="BILL" displayKind="exact" tag="exalead.loose.nlp.firstname
trustLevel="0" />
  <Annotation displayForm="person" displayKind="exact" tag="NE" nbTokens="3" trustLev
  <Annotation displayForm="2" displayKind="exact" tag="sub" nbTokens="1" trustLevel="
  <Annotation displayForm="Bill Keller" displayKind="exact" tag="NE.person" nbTokens=
</AnnotatedToken><AnnotatedToken token=" " kind="SEP_SPACE" lang="en" offset="4">
</AnnotatedToken><AnnotatedToken token="Keller" kind="ALPHA" lang="en" offset="5">
  <Annotation displayForm="keller" displayKind="lowercase" tag="LOWERCASE" nbTokens="
  <Annotation displayForm="keller" displayKind="normalized" tag="NORMALIZE" nbTokens=
  <Annotation displayForm="" displayKind="exact" tag="exalead.nlp.firstnames" nbToken
  <Annotation displayForm="KELLER" displayKind="exact" tag="exalead.loose.nlp.firstna
trustLevel="0" />
  <Annotation displayForm="3" displayKind="exact" tag="sub" nbTokens="1" trustLevel="
</AnnotatedToken><AnnotatedToken token="and" kind="ALPHA" lang="en" offset="12">
  <Annotation displayForm="and" displayKind="lowercase" tag="LOWERCASE" nbTokens="1"
  <Annotation displayForm="and" displayKind="normalized" tag="NORMALIZE" nbTokens="1"
</AnnotatedToken><AnnotatedToken token=" " kind="SEP_SPACE" lang="en" offset="15">
</AnnotatedToken><AnnotatedToken token="B" kind="ALPHA" lang="en" offset="16">
  <Annotation displayForm="b" displayKind="lowercase" tag="LOWERCASE" nbTokens="1" tr
  <Annotation displayForm="b" displayKind="normalized" tag="NORMALIZE" nbTokens="1" t
  <Annotation displayForm="Barack Obama" displayKind="exact" tag="exalead.people" nbT
trustLevel="100" />
  <Annotation displayForm="famousperson" displayKind="exact" tag="NE" nbTokens="4" tr
  <Annotation displayForm="1" displayKind="exact" tag="sub" nbTokens="4" trustLevel="
  <Annotation displayForm="Barack Obama" displayKind="exact" tag="NE.famousperson" nb
trustLevel="100" />
</AnnotatedToken><AnnotatedToken token="." kind="PUNCT" lang="en" offset="17">
  <Annotation displayForm="PUNCT" displayKind="exact" tag="tagger" nbTokens="1" trust
</AnnotatedToken><AnnotatedToken token=" " kind="SEP_SPACE" lang="en" offset="18">
</AnnotatedToken><AnnotatedToken token="Obama" kind="ALPHA" lang="en" offset="19">
  <Annotation displayForm="obama" displayKind="lowercase" tag="LOWERCASE" nbTokens="1"
  <Annotation displayForm="obama" displayKind="normalized" tag="NORMALIZE" nbTokens="
  <Annotation displayForm="Obama" displayKind="exact" tag="exalead.loose.world" nbTok
trustLevel="0" />
  <Annotation displayForm="Obama" displayKind="exact" tag="exalead.loose.world.subnat
nbTokens="1" trustLevel="0" />
  <Annotation displayForm="Obama" displayKind="exact" tag="exalead.loose.world.subnat
nbTokens="1" trustLevel="0" />
</AnnotatedToken>
```

Note: For details about the XML tags, see [Appendix - Semantic Resources Reference](#). Keep in mind that this XML output is a serialization of the underlying JAVA objects manipulated by the semantic pipeline.

This is how the XML processes the textual input:

- The pipeline processes each token ("Bill", " ", "Keller", " ", "B", ".", " ", "Obama") separately. We then obtain as many `AnnotatedToken` nodes as the number of tokens contained in the textual input.
- Each token goes through the pipeline and each processor generates one or many `Annotation` java objects that are appended to the `AnnotatedToken` object.

Focus on the two main attributes of an `Annotation`, `tag` and `displayForm`, which you can consider as (a key, value) describing the content of the `Annotation`.

- `tag` – the name of the annotation. For example, "Bill" is labeled as a first name using the tag `"exalead.nlp.firstnames"`
- `displayForm` – the value of the annotation (may be empty). This attribute is very useful for normalization purposes. For example, in a sentence containing "Barack Obama", "B. Obama", "Obama Barack", all these 3 N-grams may be annotated as `"exalead.people"` and with the same `displayForm` `"Barack Obama"`.
- The Named Entities Matcher processor matches the "Bill" token since "Bill Keller" (the display form) is tagged by `NE.person`. We notice here that the `Annotation` object has an `nbToken` attribute set to 3. This reveals the way processors work:
 - When treating a token (here "Bill"), the processor checks in its resources if the token matches with the beginning of a `displayForm`. If it is the case, the generated `Annotation` includes information about the number of tokens involved, for example, 3 for "Bill Keller".

Note: That processors work forward and never backward. They consider the tokens following the current one but not the previous ones.

Once the pipeline has produced these annotations, they may be mapped to produce as many index fields or categories as required.

More About Semantic Analysis

To be able to provide relevant search results when the user's query is incomplete, misspelled, or imprecise, Exalead CloudView performs a semantic analysis of documents as well as the queries themselves. This generates word matching operators and fuzzy matching options.

For example:

- Did you mean? Spell check: "exalaed" prompts "Did you mean: exalead?"
- Approximation: "exalaed" matches "exalead"
- Phonetic spelling: "exaleed" matches "exalead"
- Word truncations: "exal*" matches "Exalead", "exalid" and "exalted"
- Regular expressions: "/exa.ead/" matches "exalead" and "exahead"

When does Semantic Analysis take Place?

Depending on the semantic feature, the analysis takes place either at indexing-time, or at search-time.

- Index-time: analyzes documents before indexing, using semantic processors. Anytime you modify semantic processors, you must always reindex your documents before the change appears in your application.
- Search-time: analyzes the user's search request, known as Query Expansion, which essentially adds additional search terms to the user's original query. For example, if phonetic query expansion is enabled, the query "exaleed" would be expanded to "exaleed" OR "exalead".

This section explains how to perform index-time semantic analysis by configuring semantic processors.

For information on search-time semantic analysis, see [Configuring Query Expansion](#).

Set Up Semantic Analysis?

Begin your semantic configuration using the semantic types delivered in the default Data model. With semantic types, you can configure index-time options such as:

- language detection
- tokenization
- basic indexing form or kind (normalized, exact, or lowercase)
- extractions of phonetized forms and spell-check ngrams

These are examples of the basic building blocks of semantic analysis that allow you to set up more advanced semantics. For example, using the Rules Matcher processor or the Semantic Extractor processor.

For more information on semantic types, see [Indexing Options for Alphanumeric Properties](#).

Index-Time Semantic Analysis

A semantic processor adds semantic information to text during analysis. These are annotations that you can map to fields and categories (index-time facet) in the index.

The annotations are named based on the type of semantic processor and its configuration.

Note: Because this analysis occurs at index-time, you must reindex your documents after enabling or modifying these features.

These are the main semantic processors available in Exalead CloudView:

- **Related terms** flag-related concepts in your corpus. Related terms typically display as navigation facet in your search application.
- **Named entities** flag people, places, organizations, or events in your corpus. Named entities typically display as navigation facet in your search application.
- **Phonetizers** creates a phonetic version for each word in your corpus and stores them in the dictionary. Phonetic processing significantly improves the effectiveness of spell check and enables phonetic search (soundslike: exaleed). This processing is language-dependant.
- **Rules-based matching and annotations** are provided through semantic processors such as the Rules Matcher, Fast Rules, the Ontology Matcher, and Semantic Extraction.
- **Ngram Extractors** calculate probability of word occurrences or phrases within the corpus. This significantly improves the effectiveness of spell-check at search-time.

Other Documentation about Semantic Analysis

- For a detailed reference of the processor parameters, see the semantic processors descriptions in the "Search" section of the CloudView XML Configuration Reference Guide.
- For a detailed reference of the format of a semantic processor's resource file, see [Appendix - Semantic Resources Reference](#).

Tokenizing Text

Tokenization is the process of splitting up a segment of text into smaller pieces, or tokens. Tokens can be broadly described as words, but it is more accurate to say that a token is a sequence of characters grouped together for useful semantic processing.

Since tokenization is a required processing step for all searchable alphanumeric text, it is set up automatically as part of the Exalead CloudView installation. This setup is known as the default tokenization config, `tok0`.

A tokenization configuration specifies which tokenizers to use when Exalead CloudView analyzes incoming documents at index-time. It also specifies how to tokenize queries at search-time.

Note: You can test the result of the tokenization process in **Index > Data Processing > Test**.

Using Native Tokenizers

Using Basis Tech Tokenizer

About Creating Additional Tokenization Configurations

Customizing the Tokenization Config

About Decompounding

Using Native Tokenizers

Cloudview tokenizes many languages natively. This is the Standard support.

Standard tokenizer

This tokenizer is set up by default when you install Exalead CloudView. It breaks down text into tokens whenever it encounters a space or a punctuation mark. It recognizes all known punctuation marks for all languages.

You can have either:

- One Standard tokenizer, to process all languages without a dedicated tokenizer (this is the default setup).
- Multiple Standard tokenizers to process one or several specified languages for which you want to guarantee a certain tokenization behavior.

Character Overrides

When Exalead CloudView tokenizes text, it examines each character and checks the character override rules for any special processing instructions for this particular character.

Type	Description
token (default)	Processes the specified character as a normal alphanumeric character.
ignore	Does not process the specified character during indexing, and does not include it when building the query tree at search time.
punct	Processes the specified character as a punctuation mark. Important: The underscore character is not considered as a punctuation mark by default. For example, <code>j o h n _ d o e</code> is considered as a single token. To consider

Type	Description
	the underscore as a punctuation mark, specify it in the characters to override (<code>_</code>). For example, <code>john_doe</code> is sliced into three tokens.
sentence	The specified character denotes the end of a sentence.
separator	Processes the specified character as a separator.

Pattern Overrides

When Exalead CloudView tokenizes text, it searches for any patterns between the current location to the end of the document, based on the pattern override rules. For each matching pattern, it applies the corresponding processing instruction to this particular pattern.

Specify patterns using PERL 5 regular expressions.

Type	Description
token (default)	<p>Processes the specified pattern as a normal alphanumeric pattern.</p> <p>The following override patterns are used for standard tokenization:</p> <p><code>[[:alnum:]] [&] [[:alnum:]]</code> – for example, <code>M&M</code> or <code>3&c</code> are considered as one token.</p> <p><code>[[:alnum:]]* [.] (? i : net)</code> – none, one or several alphanumeric characters followed by <code>.net</code> or <code>.Net</code> or <code>.NET</code> or <code>.nEt</code> or <code>.nET</code> or <code>.neT</code> is considered as a single token, for example, <code>.Net</code>, <code>ASP.NET</code></p> <p><code>[[:alnum:]]+ [+] +</code> – for example, <code>C++</code> is considered as one token.</p> <p><code>[[:alnum:]]+ #</code> – one or several alphanumeric characters followed by a sharp sign are considered as a single token, for example, <code>M#</code>, <code>free#</code>, <code>2u#</code></p> <p>Example: to avoid slicing compound words separated by hyphens into 3 tokens and get only one token, you can set the following pattern:</p> <p><code>[[:alnum:]]+ [-] [[:alnum:]]+</code></p>
ignore	Does not process the specified pattern during indexing, and does not include it when building the query tree at search time.
punct	Processes the specified pattern as a punctuation mark.
sentence	The specified pattern denotes the end of a sentence.
separator	Processes the specified pattern as a separator.

Disagglutination Options

Selecting these options for German, Norwegian, or Dutch means once the tokenizer has produced a token during indexing, it checks if the token is a compound word. If it is, it adds annotations for each part of the compound word. This way, searching for one part of the compound word can match the whole word.

For more details on decompounding, see [About Decompounding](#).

Concatenation Options

Selecting these options ensures words that include numbers are processed as a single token. For example:

- `Windows7` is processed as a single token if `concatAlphaNum` is true. Otherwise, it is processed as two tokens.
- `9Mile` is tokenized as a single token if `concatNumAlpha` is true. Otherwise, it is processed as two tokens.

Normal separator rules for tokenization still apply: if there is a separator between numbers and other letters, the numbers and letters are processed as two separate tokens.

Transliteration Option

You can activate the `transliteration` option in the Exalead CloudView XML configuration (it is not available in the Administration Console), for transliterating Unicode Latin Extended B characters. Several characters are converted to their closest Latin equivalent to facilitate query typing. This is useful when you want to match documents in another charset from the one you use for searching.

For example, you may want to search for words containing "Ł" using the closest Latin character "L" even though it does not match phonetically.

As for now, the supported transliterations are the following:

- `00D0 = Ð -> 'd'`
- `0110 = Đ -> 'd'`
- `00F0 = ð -> 'd'`
- `0111 = đ -> 'd'`
- `00D8 = Ø -> 'o'`
- `00F8 = ø -> 'o'`
- `0126 = Ĥ -> 'h'`
- `0127 = ĥ -> 'h'`

- 0131 = ι -> 'i'
- 0141 = \mathfrak{L} -> 'l'
- 0142 = \mathfrak{t} -> 'l'
- 0166 = \mathfrak{T} -> 't'
- 0167 = \mathfrak{t} -> 't'
- 0192 = f -> 'f'

Japanese Tokenizer

This tokenizer is set up by default. You can have only one Japanese tokenizer per tokenization config.

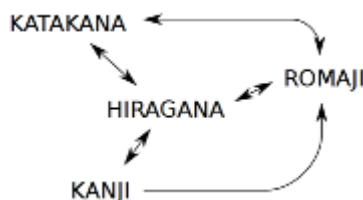
The 4 alphabets used in Japanese texts (kanji, hiragana, katakana, romaji) are implemented by this tokenizer.

Japanese Text Normalization

The text normalization process for Japanese is more complex than for European languages:

- normalization of half-width katakana to full-width, applying NFKC (Normalization Form Compatibility Composition)
- normalization of full-width roman characters to standard latin alphabet
- in addition to each normalized token, indexing of the corresponding hiragana form. At query time, the same processing is applied. It allows to match this normal form with the alphabet used in the documents and the query.
- over-indexing of the okurigana-free forms of kanji

This processor supports the following transcriptions:



The default configuration uses only tokens and their hiragana transcriptions. You can use any processor in the semantic pipe to exploit other transcriptions produced for linguistic purposes. To annotate text with parts of speech to flag nouns, verbs, and adjectives, select the option **Add morphology** in **Linguistics > Tokenizations > your tokenization > Japanese**.

Use of Recall

Since tokenizing Japanese text is a difficult task, the processor may produce different outputs for the same input. Some of them may contain errors.

For example, in the following cases, queries may fail and skip documents (thus increasing silence):

- the context is different.

A text may be tokenized in the context of a document in a different way than it is in a query because the context is missing.

- the alphabet is different.

For example, tokenizing kanji may produce a different output than tokenizing the equivalent hiragana.

- the document has been processed by a nonjapanese tokenizer at indexing time.

For example, the tokenizer for European languages produces single-character tokens from Japanese texts, which do not match correct Japanese tokenization.

To maximize recall and avoid errors, a character-based over-indexing is enabled by default. This over-indexing reduces search silence by trying to match character sequences independently of tokenization, in addition to the tokenized query. However, to avoid an unreasonable amount of noise in top search results, this character-based query expansion has a much lower contribution to the document score than the tokenized part.

If too many results are displayed, you can disable this recall by changing the value of the `favor` option from `recall` to `precision` in the Japanese tokenizer configuration (in `<DATADIR>/config/Linguistic.xml`).

Japanese Spell-Check Settings

To compute spell-check suggestions for Japanese, you need to change default settings in **Search > Search logics > Your search logic > Query Expansion**. See [Set Up Spell-Check for CJK \(Chinese-Japanese-Korean\)](#) for more details.

If you have purchased Extended Languages, you can remove this tokenizer and instead set up a Basis Tech tokenizer for Japanese tokenization.

The main differences are:

- Exalead tokenizer is based on the open-source JUMAN Japanese processor. BasisTech is based on proprietary technology. You may find differences in tokenization results when switching from one system to another.
- Exalead tokenizer shows better processing performances on average.

- BasisTech tokenizer shows a higher coverage when extracting named entities.

Chinese Tokenizer

This tokenizer is specified by default. You can only have one Chinese tokenizer per tokenization config.

It includes the option to annotate the text with simplified Chinese transliterations.

The Chinese tokenizer retrieves first (and with a higher score) the sets of ideograms that correspond to real words. It also tokenizes ideogram by ideogram to support arbitrary ideogram combination, but this is less relevant in terms of meaning.

Note: If you have purchased Extended Languages, you can remove this tokenizer and instead set up a Basis Tech tokenizer for Chinese tokenization.

Using Basis Tech Tokenizer

Exalead CloudView also offers Extended Languages, provided by a third-party linguistic platform, Basis Technology's Rosette. Basis Tech is only available if you have installed the Extended Languages add-on.

You can define a Basis Tech tokenizer for a specific language only. Basis Tech tokenizers generally offer richer semantic processing for Asian, Middle-eastern, and African languages than the native Exalead CloudView tokenizers.

Note: For the complete list of supported languages, see the "CloudView Supported Languages" datasheet.

Install the Extended Languages Add-On

This gives you access to the Basis Tech tokenizer. This add-on is available as a separate license.

1. See "Install add-ons" in the Exalead CloudView Administration Guide.
 - a. From the Administration Console, go to **Help > License**.
 - b. Check that the **Extended Languages** feature is available.

Enable Extended Languages

Create one Basis Tech tokenizer for each language to be tokenized.

Important: There are additional steps for setting up Basis Tech tokenizers for Chinese, Japanese, German, Dutch, or Norwegian. See steps 5 and 6.

Before enabling Extended Languages, keep in mind the following:

- While providing more in-depth text analysis for certain languages, Basis Technology's analysis typically requires more RAM.
- Named Entities extraction for languages with Extended Languages enabled are determined by Basis Technology's tokenization rules instead of Exalead CloudView's:

Basis Technology uses a tokenization based on the Unicode standard word boundaries definition () with a few customizations.

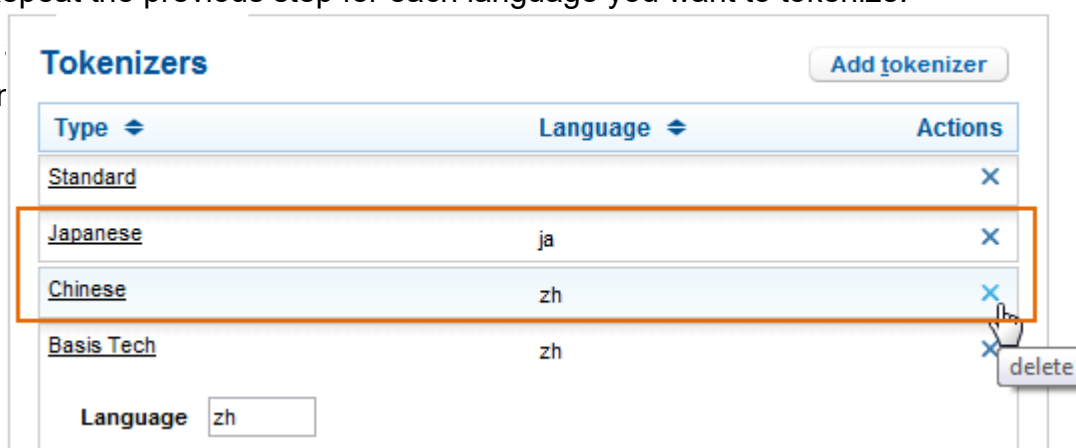
The main difference with Exalead CloudView's tokenization is that Basis Tech tokenization does not split text on certain punctuation when they are not preceded and followed by blank spaces.

- colons: Basis Tech keeps together 12:30pm. Standard tokenization produces three tokens 12 + : + 30pm
- periods: I.B.M. or 12.33 both produce a single token
- apostrophes: a single token for can't or 1970's

This may affect your downstream processing in the semantic pipeline. For example, when an ontology has been compiled with Exalead CloudView's default tokenization, terms made of those characters do not match.

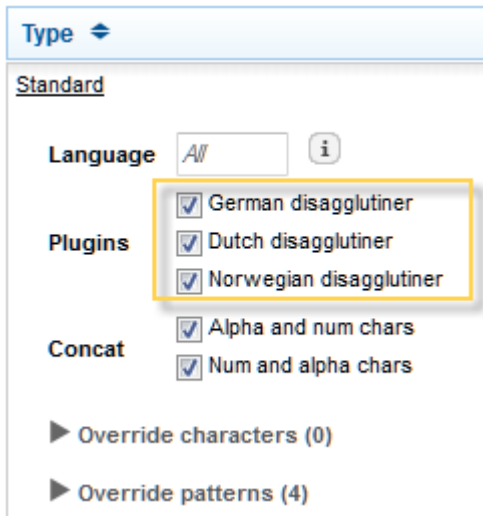
1. In the Administration Console, go to **Index > Linguistics > Tokenizations**.
2. Click your tokenization config (for example, **tok0**).
3. In **Tokenizers**, click **Add tokenizer**.
 - a. For **Type**, select **Basis Tech**, and then click **Accept**.
 - b. For **Language**, select the language to tokenize.
4. Repeat the previous step for each language you want to tokenize.

5. If you are adding Japanese or Chinese tokenizers, you may want to add Japanese or Chinese tokenizers.



6. If adding Basic Tech tokenizers for German, Dutch, or Norwegian:
 - a. Click the **Standard** tokenizer.

- b. Clear the **German**, **Dutch**, or **Norwegian disagglutiner** option.



7. Click **Apply**.
8. Reindex.

Enable Lemmatization with Basis Tech

With Exalead CloudView's native tokenization, we only index the original words. At query time this adds an OR expansion for lemmatization.

For example: `q=alsacien` is expanded to `q=alsacien OR alsacienne OR alsaciens OR alsaciennes`.

When using BasisTech to tokenize a language, you must index the lemmatized form of the word in addition to the original form. Indeed, Basis Tech does not include the resource that associates a form to all its expansions.

1. In **Data Processing > Semantic Processors**, add a **Lemmatizer** semantic processor to your analysis pipeline.
For more information, see [Configure Lemmatization Manually](#).
2. Go to **Index > Linguistics > Tokenizations**.
3. Click your tokenization config (for example, **tok0**).
4. On the **Advanced** tab, add the following tag and matching mode:
 - a. **Tag**: lemma
 - b. **Matching Mode**: 2
5. Click **Apply**.
6. Reindex.

All `lemma` semantic annotations, which store the lemmatized form of the token, are indexed with a `kind=2`, meaning they are indexed in the same way as the original normalized form.

About Creating Additional Tokenization Configurations

A tokenization configuration specifies which tokenizers to use when Exalead CloudView analyzes incoming documents at index-time. It also specifies how to tokenize queries at search-time.

By default, Exalead CloudView uses `tok0` as the tokenization configuration for converting text into tokens. However, if you create additional tokenization configs, you must specify them explicitly in **Data Model > Semantic Types** and **Data Processing**.

When do you Need to Create a New Tokenization

You may want to create a new tokenization config to define:

- A certain character as a separator, or a character as NOT being a separator. See [Character Overrides](#).
- A specific pattern to be a word only, instead of a word with separators. For example, to make sure `C++` is always indexed as the token `C++`. See [Pattern Overrides](#).

Specify a tokenization config for a specific index mapping or semantic type, when you know that a certain meta contains characters that need to be interpreted differently than other alphanumeric metas. Typical examples are identifiers like user IDs, model numbers and product codes.

Index-Time and Search-Time Tokenization

When you specify a new tokenization config for document analysis at index-time, you must also specify this same tokenization config for interpreting queries at search-time.

You can specify a default tokenization config for a search logic, as well as an "exception" tokenization config for specific prefix handlers. For more information, see [Specify a Tokenization Configuration for Prefix Handlers](#).

Indexing is both morphological and ngram-based. This gives Exalead CloudView a kind of fallback mechanism. For example, "cjk" (that is, Chinese Japanese Korean) in **Linguistics > Tokenizations > Advanced > Form indexing** triggers the indexing of isolated characters but we still index words made of several characters output by tokenizers. This way, in case of different tokenizations between indexing and querying for the same text (either because of a different context or because of a tokenization error), we can still find documents containing the sequence of characters.

Which Tokenization Config takes Precedence?

A tokenization config that is defined for a specific:

- semantic type overrides the one defined for its corresponding analysis pipeline.

- meta mapping overrides the one defined for its corresponding semantic type.

For example:

- The analysis pipeline `ap0` uses the default `tok0` tokenization config. Any text that is processed by processors added directly to this analysis pipeline is tokenized with `tok0`.
- The **text** semantic type uses `tok1`, and the `text`, `location`, and `product_name` metas in the data model are set up to use this semantic type. In other words, `tok1` overrides `tok0` for these three metas.
- However, the `product_name` meta has special tokenization requirements, so its index mapping only uses a third config, `tok2`, which overrides `tok1`.

Customizing the Tokenization Config

You can modify the default tokenization config, or create additional ones.

Create or Edit a Tokenization Config

1. In the Administration Console, go to **Index > Linguistics > Tokenizations**.
 - Click an existing tokenization config to edit.
 - Click **Add tokenization config** to create a new one.
2. Set up the tokenizers as described in [Using Native Tokenizers](#).
3. When finished, click **Apply**.
4. Reindex your data.

Specify Another Tokenization Config for an Analysis Pipeline

1. In the Administration Console, go to **Index > Analysis**, then select your analysis pipeline (for example, `ap0`).
2. From the **Tokenization config** list, select a different configuration.
3. Click **Apply**.
4. Reindex your data.

This tokenization config is only used on metas processed by document or semantic processors that were manually configured in the analysis pipeline.

Specify Another Tokenization Config in the Data Model

1. In the Administration Console, go to **Index > Data Model > Semantic Types** tab.
2. Expand the semantic type you want to modify, or create a new one.
3. In the semantic type:

- a. Make sure that the **Tokenize** option is selected.
- b. From the **Tokenization config** list, select a different configuration.
4. Click **Apply**.
5. Reindex your data.

All properties using this semantic type tokenize the corresponding meta using this tokenization configuration, regardless of the tokenization config specified for its associated analysis meta.

Specify Another Tokenization Config for an Index Mapping

1. In the Administration Console, go to **Index > Data processing**, then select your analysis pipeline (for example, `ap0`).
2. Select the **Mappings** tab.
3. Under the **Mapping sources** list, click the mapping you want to modify.
4. From the **Tokenization config** list, select a different configuration.

Note: If this mapping was produced by a data model, you need to click **Customize** before you can modify any mapping settings.

5. Click **Apply**.
6. Reindex your data.

The meta for this mapping is always be tokenized using the tokenization config specified here, regardless of the tokenization config specified for its analysis pipeline or for its semantic type (if any).

About Decomponding

Decomponding is splitting up a word into components so that when searching for one of these components, the whole compound is found. For instance, we want "Lastwagen" or "Fahrer" to match "Lastwagenfahrer".

To achieve this, Exalead CloudView splits compound words into at most two components using a dictionary. Three issues may occur during the process:

- Since the dictionary cannot cover the integrality of a language (very specific/technical words may not be known from the algorithm), the compound may not be split because its components are not found in the dictionary.
- When there are more than one way to split the word, the most likely one is selected but this may not be the expected one.

- Some compound words should not be decomponded. The algorithm uses statistics to figure it out but some words may end up decomponded though it doesn't make much sense ("Handschu", "Volkswagen").

In these cases, the user can enrich the dictionary with his own rules by defining a list of words which should not be decomponded and a list of explicit decompondings taking precedence over the Exalead CloudView-provided resource.

Custom Resource Creation

The user dictionary is a UTF-8 text file (`user.txt`) in the directory `KITDIR/resource/all-arch/subtokenizer/ID` where `KITDIR` is the root of CloudView unzipped kit directory and `ID` is one of following language identifiers: `de` (german), `nl` (dutch), `no` (norwegian).

The file format must be as follows (lines failing to match this format are ignored):

- Lines starting with `#` are comments (ignored)
- Lines containing one word define uncompoundable words
- Lines containing three words explicitly define how the 1st word of the line must be decomponded into the 2nd and the 3rd words

Note that words are matched case-insensitively but accents matter ("Kuchen" is not "Küchen").

```
# this is an ignored comment
# this line states that Volkswagen shouldn't be split:
volkswagen
# this line forces decomponding of Lastwagenfahrer into Lastwagen+Fahrer:
lastwagenfahrer lastwagen fahrer
```

Applying Changes

When you specify a new tokenization config for document analysis at index-time, you must also specify the same tokenization config for interpreting queries at search-time.

After editing the user dictionary:

- the indexingserver and the searchserver must be restarted for the modifications to be taken into account
- documents have to be reindexed

If issues should appear, turn the logging level of the indexingserver to debug and restart it. Search for `[subtokenizer]` in the logs to filter relevant lines.

Creating and Deploying Semantic Resources

Semantic processing often requires resources such as thesauruses, synonyms or block lists, at both index and search time.

You can distribute semantic resource files either by:

- Creating the resource in the Administration Console.
- Using the resource manager through the `cvadmin` command. The following resources must be compiled and published using `cvadmin`: XML-compliant, CSV-compliant, and custom.

Create a Resource File from the Administration Console

Manage Resources in `cvadmin`

Create a Resource File from the Administration Console

You can create resource files directly from the Administration Console. Once created, the resource is automatically added in the background to the Resource Manager, which automatically compiles and deploys the resources across all hosts.

This workflow uses an Ontology Matcher as an example, however the process is similar for other semantic resources.

1. In the Administration Console, go to **Index > Data processing > Pipeline name > Semantic Processors**.
2. Add an **Ontology Matcher** processor to the pipeline.
3. In the **Resource directory** of the processor, click **Create new**.

Once the resource is added the **Resource directory** path is set to `resourcemanager://indexing/<RESOURCE_NAME>`

4. Click **Apply**. Apply changes before you can edit the resource file in the Business Console.
5. Click **Edit** to edit the resource file in the Business Console.

See "Adding Ontology Resources" in the Business Console.

Manage Resources in `cvadmin`

The Resource Manager allows you to edit linguistic and semantic resources without having to manually compile or publish them.

The resource manager publishes semantic resource files to specified roles, which are organized into groups to update interrelated resource files in unison.

The Resource Manager:

- compiles resources
- assigns versions to resources
- publishes resources, including to multiple hosts
- converts resource formats, such as from XLS to XML

This section explains how to use the Resource Manager by taking the Ontology Matcher semantic processor as an example. The procedure is the same, however, for all resource types.

Edit ResourceManager.xml

The first step is to edit `<DATADIR>/config/ResourceManager.xml`. In this file, you define groups of resources and which roles to publish these resources to.

Grouping resource files enables you to keep dependant resources together (for example, if a Rules Matcher depends on the Ontology Matcher's results). This ensures consistent updates since the group is published as a unit.

The default configuration includes the following resource groups:

- indexing, which targets the analyzers
- search, which targets the searcher

Let us assume you want to include an OntologyMatcher in our semantic pipeline. We would then include an ontology resource in the indexing ResourceGroup as shown in the following ResourceManager.xml example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<res:ResourceManagerConfig resourceDir="data:///build/resources" version="0"
xmlns:res="exa:com.exalead.mercury.mami.resources.v10" xmlns="exa:exa.bee"
xmlns:config="exa:exa.bee.config">
  <res:ResourceGroup waitOnSync="false" roles="Analyzer,Searcher,Dictionary" name="co
    <res:Resource type="rt-blacklist" name="rt-blacklist"/>
    <res:Resource type="rt-whitelist" name="rt-whitelist"/>
    <res:Resource type="categories-blacklist" name="synthesis-blacklist"/>
    <res:Resource type="facets-blacklist" name="facets-blacklist"/>
    <res:Resource type="stopwords" name="stopwords"/>
    <res:Resource type="spellcheck-blacklist" name="spellcheck-blacklist"/>
    <res:Resource type="spellcheck-whitelist" name="spellcheck-whitelist"/>
  </res:ResourceGroup>
  <res:ResourceGroup waitOnSync="false" roles="Analyzer" name="indexing">
    <res:SemanticResource type="ontology" name="onto" tokenizationConfig="tok0"/>
    <res:SemanticResource type="rules" name="rules" tokenizationConfig="tok0"/>
  </res:ResourceGroup>
  <res:ResourceGroup waitOnSync="false" roles="Searcher" name="search" />
```

```
</res:ResourceManagerConfig>
```

Once you have edited your `ResourceManager.xml` file, we need to apply the configuration by using the API Console (select **Manage**, then search for the `applyConfiguration` method and click **Send**).

Upload, Compile, and Publish the Ontology

We now have a new configuration and we have created our own resource file called `ontology.xml`.

We now need to upload this file to the Resource Manager. To do so, we use the `cvadmin` command-line tool. First check that the resources have been properly added:

```
cvconsole cvadmin> resources list-resources group=group_name_of_your_resource
```

We can see our resource in the list. We can now upload our resource file:

```
cvconsole cvadmin> resources upload path=<path/to/your/file/ontology.xml> resource=<n
[publish=true]
```

The name of our resource is the one we defined in the `ResourceManager.xml` file.

Note: By using the optional argument `publish=true`, the file is uploaded, compiled (if required) and published in one command. If we wanted to separate these actions, after uploading our file, we would use:

```
cvconsole cvadmin> resources publish group=<name_of_the_group>
```

Note: .

This publishes all the resources in the specified group. If the last upload date is later than the last compile date, then this also proceeds to a compilation.

Errors may come up because the resource file (here `ontology.xml`) is not correct. Since the Resource Manager is part of the gateway, we can check the gateway's log file for causes of a compilation failure.

The resources published are available under `resourcemanager://group_name/resource_name`

Supported Resource Types

Resource	Type value	RESOURCE_NAME (for samples)
Annotation Manager	<code>type="annotation-manager"</code>	n/a
Ontology Matcher	<code>type="ontology"</code>	n/a
Sentiment Analyzer	<code>type="sentiment"</code>	n/a

Resource	Type value	RESOURCE_NAME (for samples)
Rules Matcher	type="rules"	n/a
Fast Rules	type="fastrules"	n/a
Semantic Extractor	type="semantic-extractor"	n/a
Lemmatizer	type="lemmas"	n/a
Synonyms expansion module	type="synonyms"	n/a
Related terms blacklist	type="rt-blacklist"	rt-b10
Suggest allow list	type="suggest-whitelist"	suggest-w10
Suggest block list	type="suggest-blacklist"	suggest-b10
Query block list	type="query-blacklist"	query-b10
Spellcheck block list (See Setting Up Spellcheck Resources in dictionary.xml)	type="spellcheck-blacklist"	spellcheck-b10
Spellcheck allow list (See Setting Up Spellcheck Resources in dictionary.xml)	type="spellcheck-whitelist"	spellcheck-w10
CSV-compliant	type="csvraw" (no compilation, no validation)	n/a
Custom	type="raw" (no compilation, no validation)	n/a
XML-compliant	type="xmlraw" (no compilation, no validation)	n/a

Setting Up Spellcheck Resources in dictionary.xml

For spellcheck block list and allow list resources, you need to perform an extra step. After declaring the resources, you must also edit the `dictionary.xml`.

1. Declare the two resources handling the allow and block listing. In this example, we add them to the "content-filtering" group of the Resource Manager.

```
<res:ResourceManagerConfig resourceDir="data:///build/resources" version="13389679"
xmlns:res="exa:com.exalead.mercury.mami.resources.v10" xmlns="exa:exa.bee"
xmlns:config="exa:exa.bee.config">
  <res:ResourceGroup waitOnSync="false" roles="Analyzer,Searcher,Dictionary" name="
    <res:Resource type="spellcheck-blacklist" name="spellcheck-bl0"/>
    <res:Resource type="spellcheck-whitelist" name="spellcheck-wl0"/>
  </res:ResourceGroup>
</res:ResourceManagerConfig>
```

2. Since the global dictionary is in charge of spell checking, you need to edit the `dictionary.xml` configuration file to bind the resources.

```
<DictionaryConfig xmlns="exa:com.exalead.mercury.mami.linguistic.v10" version="13389679"
maxWords="126000000" maxRelatedTerms="10000000"
minNbDocsForRelatedTerm="10" maxWordCooccurrencesForSpellcheck="1260000000" automa
maxInputsBeforeBuild="0"
tokenizationConfig="tok0" preAllocatedInterpretationPipelines="2" preAllocatedSpel
<!-- specify resource names declared for the "content-filtering" group -->
spellCheckBlacklistResource="spellcheck-bl0" spellCheckWhitelistResource="spellche
<!-- [...] -->
</DictionaryConfig>
```

3. Apply the configuration.
4. Create the allow list and block list CSV files. For sample formats, see [Access Sample Block List and Allow List Formats](#).
5. Upload, compile, and publish your resources to the Resource Manager using the `publish=true` option.

```
cvconsole cvadmin> resources upload path=/path/to/spellcheck_blacklist.csv resource
publish=true
cvconsole cvadmin> resources upload path=/path/to/spellcheck_whitelist.csv resource
publish=true
```

Access Sample Block List and Allow List Formats

These resource formats are available using the `resources get-sample` command of `cvadmin`, where `RESOURCE_NAME` is the name of the sample. In `cvadmin`, enter:

```
cvconsole cvadmin> resources get-sample resource=RESOURCE_NAME
```

See [Supported Resource Types](#) for `RESOURCE_NAMES`. For example:

```
cvconsole cvadmin> resources get-sample resource=spellcheck-bl0
```

Returns:

```
# language, level, expression "en", "normalized", "exalead"
```


Convert Semantic Resources to Other Formats

Several `cvadmin` and `cvdebug` commands are available to convert resources to different formats. You can go to `<DATADIR>/bin/` to start them.

To output/convert	Start the command...
a compiled ontology resource to XML	<code>cvconsole cvdebug > linguistic dump-ontology path="<PATH TO COMPILED ONTOLOGY>"</code>
an ontology resource from XML to XLS	<p><code>cvconsole cvadmin > linguistic convert-ontology-from-xml-to-xls input=/tmp/onto.xml output=/tmp/out.xls</code></p> <p>For example, the following ontology:</p> <pre><Ontology xmlns="exa:com.exalead.mot.components.ontology"> <Pkg path="top.custom"> <Entry display="Mickey Mouse"> <Form value="Mickey M." level="normalized"/> <Form value="Mouse M." level="exact" /> </Entry> </Pkg> </Ontology></pre> <p>generates an <code>out.XLS</code> file with a single sheet, named "<code>top.custom</code>" that contains:</p> <ul style="list-style-type: none"> • Display: Mickey Mouse, Form: Mickey M., Lang: xx, Level: normalized • Display: Mickey Mouse, Form: Mouse M., Lang: xx, Level: exact
an ontology resource from XLS to XML	<code>cvconsole cvadmin> linguistic convert-ontology-from-xls-to-xml input=/tmp/out.xls output=/tmp/out.xml</code>
an ontology resource from SKOS to XML	<code>cvconsole cvadmin> linguistic convert-ontology-from-skos-to-xml input=/tmp/out.xls output=/tmp/out.xml</code>
a suggest resource from SKOS to XML	<code>cvconsole cvadmin> linguistic convert-suggest-from-skos-to-xml input=/tmp/out.xls output=/tmp/out.xml</code>
a synonym resource from SKOS to XML	<code>cvconsole cvadmin> linguistic convert-synonyms-from-skos-to-xml input=/tmp/out.xls output=/tmp/out.xml</code>

Managing Semantic Annotations

This section explains how to manage semantic annotation with the Annotation Manager or with custom code.

Manage Annotations with the Annotation Manager

The Annotation Manager allows you to perform several operations on annotations under the right conditions. You can use it to copy, select, and remove annotations.

The Annotation Manager configuration consists of a list of operations.

Important: There is no define order of the execution of operations. If you really care about operation ordering, you must add several annotation managers to the semantic pipe.

Copy Annotation

You can copy a source annotation along with its display form, display kind, and trust level to a target annotation.

Option	Example
Copy without condition	For example, to ignore the distinction between famous and nonfamous people. <code><Copy annotation="NE.famousperson" target="NE.person"/></code>
Copy with a condition	For example, to copy famous people to the nonfamous people annotation, unless they are block listed. <code><Copy annotation="NE.famouspeople" target="NE.people" unless="blocklisted"/></code>

Remove Annotation

You can remove the occurrences of an annotation under the right conditions.

Option	Example
Remove without condition	For example, to remove all end-of-sentences: <code><Remove annotation="sbreak" /></code>
Remove an annotation if it	For example, to remove a person's name annotation when it spreads over two sentences: <code><Remove annotation="NE.person" ifOverlapWith="sbreak" /></code>

Option	Example
overlaps with another one	
Remove an annotation if the annotated text span matches that of another one	<p>For example, to remove a person's name annotation when the text is block listed:</p> <pre><Remove annotation="NE.person" ifMatchWith="blocklist.person" /></pre> <p>An ontology matcher upstream or any other semantic processor can set the annotation <code>blocklist.person</code>. Both annotations must start and end exactly on the same tokens.</p>
Remove an annotation if the annotated text span and display form match those of another one	<p>For example, we want to implement a block list with a fine granularity:</p> <pre><Remove annotation="title.approx" ifMatchWith="blocklist.title" displayFormsMustMatch="true"/></pre> <p>If an ontology containing a <code>title</code> package matches <code>professor</code> on the text processor using approximation</p> <pre><pkg path="title"> <Entry> <Form value="professor" /> </Entry> </pkg></pre> <p>... the annotation is removed if the annotation (<code>blocklist.title</code>, <code>"professor"</code>) occurs at the very same place, thus block listing the specific approximation.</p>
Keep the first occurrence of an annotation and remove all others	<p>For example, to keep only the first organization occurrence in title and text:</p> <pre><KeepFirst annotation="NE.organization" contexts="title,text"/></pre>
Keep the longest leftmost of a set of overlapping annotations and remove all others	<pre><KeepLongestLeftMost annotations="NE.person,NE.place,NE.organization" interTags="false"/></pre> <p>With <code>interTags</code> set to <code>false</code>, one annotation per tag is kept.</p>

Select Annotation

You can select the most frequent annotations and store the results as document annotations.

Option	Example
Select the most frequent values in a document for a given annotation	<p>For example, we want to select the 5 places that occur the most in a document and store them in <code>selectedPlaces</code> document annotations.</p> <pre><SelectMostFrequentValue annotation="NE.place" documentAnnotation="selectedPlace" howMany="5" truncate="true"/></pre> <p>If there are more than 5 most frequent places, the resulting list is arbitrarily truncated since <code>truncate="true"</code> guarantees that no more than 5 annotations are ever reported.</p>
Select the most frequent annotation in a document among a list	<pre><SelectMostFrequentAnnotation annotations="NE.organization,NE.place,NE.person" documentAnnotation="selectedAnnotation"/></pre> <p>The most frequent annotation is used to output a <code>selectedAnnotation</code> document annotation whose value is one of the annotations from the list.</p>
Select annotations depending on an index field (context) priority	<p>For example, we want to select an annotation from the "title, text" contexts, by first looking within the <code>title</code> context and then, if the annotation is not found, looking within the <code>text</code> context:</p> <pre><SelectByContexts annotation="NE.person" contexts="title,text" documentAnnotation="selectedAnnotation" firstOnly="false"/></pre> <p>With <code>firstOnly</code> set to <code>false</code>, all occurrences of <code>NE.person</code> annotations in the specified contexts are reported.</p>

Use Regular Expressions

You can use regular expressions for all annotation parameters.

Set `enableRegexp` to `true` in the `<AnnotationManager>` object (default is `false`).

Example of Annotation Manager XML Configuration File

```
<AnnotationManager xmlns="exa:com.exalead.linguistic.v10">
  <Copy annotation="NE.famousperson" target="NE.person"/>
  <Copy annotation="NE.famousperson" target="NE.person" unless="blocklist"/>
  <Remove annotation="NE.famousperson" ifOverlapWith="sbreak" />
  <Remove annotation="NE.person" ifOverlapWith="sbreak" />
  <Remove annotation="NE.person" ifMatchWith="blocklist.person" />
  <Remove annotation="title.approx" ifMatchWith="blocklist.title" displayFormsMustMat
</AnnotationManager>
```

Manage Annotations with Custom Code

In most semantic-oriented projects, you need to manipulate (filter, combine, replace, etc.) the semantic annotations set by your semantic processors before sending them to the index.

The easiest way to do this is to add semantic processors into the Document Processor pipeline, transforming the annotations into metas (also known as chunks). Then you can manipulate them using either:

- custom java code via the **JavaDocumentProcessor**
- standard document processors such **ReplaceValues** or **ConcatenateValues**.

Example: Index Term Occurrences in a Document

Let us say you want to send to the index the number of times a term is matched in the document from an existing list of terms.

Recommendation: Use the **OntologyMatcher** to detect all terms. Go through it using a **SemanticPipeDocumentProcessor**. Convert the semantic annotations into metas (or chunks) and use custom java code to count them.

Create a List of Terms

This ontology annotates each term of the list with the "myterms" annotation.

```
<Ontology xmlns="exa:com.exalead.mot.components.ontology">
  <Pkg path="myterms">
    <Entry display="Term 1">
      <Form level="normalized" />
    </Entry>
    <Entry display="Term 2">
      <Form level="normalized" />
    </Entry>
    <Entry display="Term 3">
      <Form level="normalized" />
    </Entry>
    <!-- [...] -->
  </Pkg>
</Ontology>
```

Modify the Analysis Pipeline

Add the following configuration to the end of your document processor Analysis Pipeline.

Each "myterms" semantic annotation is converted into a meta (or chunk), that the Document Processors can manipulate.

The XML representation of the **SemanticPipeDocumentProcessor** configuration looks like:

```
<SemanticPipeDocumentProcessor
  annotations="myterms" // Comma-separated semantic annotations that will be converted into metas
  topLevelAnnotationsOnly="false" // only convert document annotations?
  disabled="false" name="SemanticPipeDocumentProcessor.0">
  <OntologyMatcher resourceDir="/path/to/myterms.bin" disabled="false" name="Ontology
</SemanticPipeDocumentProcessor>
```

In the **JavaDocumentProcessor**, count the number of metas (or chunks) named "myterms". Add the count to a new "nbTerms" meta (the mapping of this "nbTerms" meta to the index is not detailed here).

```
import com.exalead.pdoc.ProcessableDocument;
import com.exalead.pdoc.analysis.DocumentProcessingContext;
import com.exalead.pdoc.analysis.StandardDocumentProcessor;
import com.exalead.pdoc.Meta;
public class JavaDocumentProcessorTemplate extends StandardDocumentProcessor {
    @Override
    public void process(DocumentProcessingContext context, ProcessableDocument document)
        throws Exception {
        int count = 0;
        for (Meta m : document.getMetas("myterms")) {
            ++count;
        }
        document.addMeta("nbTerms", String.valueOf(count));
    }
}
```

Configuring Form Indexing

Form indexing configuration defines pairs of semantic annotations and matching modes (or index levels). Here, semantic annotation values are indexed at the defined matching mode. The matching mode is an arbitrary integer required to access inverted lists (word, level), which gives access to the word positions in all the documents.

Three matching modes have a predefined meaning: 0 is exact, 1 is lowercase, 2 is normalized. The rest is up to the user.

For example, there is a hidden form indexing configuration (NORMALIZE, 2) that defines that the normalizer's `NORMALIZE` annotations must be indexed at level 2. Then at query time, if `normalized` is the prefix handler matching mode, these annotations permit access to the index and to look for the requested words.

Use Form indexing for Over-Indexing Acronyms

The form indexing customization helps over-indexing. For example, we want that the query `NASA` matches occurrences of `NASA` and `N.A.S.A.`. That is to say, each time `N.A.S.A.` appears in a document, we want to over-index it with `NASA`.

1. Add an acronym detector in the analysis pipe.
 - a. Go to **Data Processing > Semantic Processors**.
 - b. Drag the **Acronym Detector** in the analysis pipeline.
2. Add a form indexing (`acronym, 2`) so that the acronym detector's annotations are indexed at level 2.
 - a. Go to **Index > Linguistics > Tokenizations > Advanced**.
 - b. Click **Add form**.
 - c. For **Tag**, enter `acronym` and for **Matching Mode**, enter 2 (normalized).

Since our prefix handler targets matching mode 2, any query word can match any over-indexed value coming from the acronym detector.

Set Weight

To set a distance (or weight) in Form indexing configuration, you may specify an additional **Trust level** in **Index > Linguistics > Tokenizations > Advanced**. This attribute ranges from 1 to 100, 100 being the highest and default weight. The query expander uses it to compute a weight for expansion.

1. Let us say that in the `Linguistic.xml` file, the `trustLevel` parameter corresponds to a weight of 50.

```
<ling:LinguisticConfig version="0" xmlns:bee="exa:exa.bee" xmlns="exa:com.exalead.
xmlns:config="exa:exa.bee.config">
  <ling:TokenizationConfig name="tok0">
    <ling:StandardTokenizer concatNumAlpha="true" concatAlphaNum="true">
      <ling:BasisTechTokenizationCompatibility languages="en,de,fr,sv,es,it,nl,pt,
hu,pl,sk,sl,sr"/>
      <ling:GermanDisagglutiner/>
      <ling:DutchDisagglutiner/>
      <ling:NorwegianDisagglutiner/>
      <ling:charOverrides/>
      <ling:patternOverrides>
        <ling:StandardTokenizerOverride toOverride="[:alnum:]]&[:alnum:]]"
        <ling:StandardTokenizerOverride toOverride="[:alnum:]]*.(?i:net)" type=
        <ling:StandardTokenizerOverride toOverride="[:alnum:]]+["+ type="token"
        <ling:StandardTokenizerOverride toOverride="[:alnum:]]+#" type="token"/>
      </ling:patternOverrides>
    </ling:StandardTokenizer>
    <ling:JapaneseTokenizer addMorphology="false" addRomanji="true"/>
    <ling:ChineseTokenizer addSimplified="false"/>
    <ling:BasisTechTokenizer language="ko"/>
    <ling:FormIndexingConfig>
      <ling:Form tag="SubTokenizerLowercase" indexKind="1"/>
      <ling:Form tag="SubTokenizerNormalize" indexKind="2"/>
      <ling:Form tag="SubTokenizerConcatLowercase" indexKind="1"/>
      <ling:Form tag="SubTokenizerConcatNormalize" indexKind="2"/>
      <ling:Form tag="cjk" indexKind="2"/>
      <ling:Form tag="lemma" indexKind="3"/>
      <ling:Form tag="jafactorized" indexKind="42"/>
      <ling:Form tag="jaexpanded" indexKind="43"/>
      <ling:Form tag="jaromanji" indexKind="44"/>
      <ling:Form tag="jaradicalfactor" indexKind="45"/>
      <ling:Form tag="jaradicalexpand" indexKind="46"/>
      <ling:Form tag="compound" indexKind="2" trustLevel="50"/>
    </ling:FormIndexingConfig>
    <ling:NormalizerConfig useGermanExceptions="false" disableBasisTechNormalizerF
useNormalizationExceptions="true" transliteration="true"/>
  </ling:TokenizationConfig>
</ling:LinguisticConfig>
```

2. In the resulting expansion, compounding has a weight of 0.5 using BasisTech Korean tokenizer:

```
http://localhost:10010/search-api/search?q=###&l=ko
#query{nbdocs=0, text_relevance.expr="@term.score * @proximity + @b", proximity.ma
term.score=RANK_TFIDF}
(#or{*.policy=MAX}
```



```
(#alphanum{source="MOT",seqid=0,groupid=0,k=2}(text,"###")  
#alphanum{w=0.5,source="MOT",seqid=0,groupid=0,k=2}(text,"##")  
#alphanum{w=0.5,source="MOT",seqid=0,groupid=0,k=2}(text,"#") ) )"
```

Configuring Search Queries

Describes how to configure the behavior of the search fields.

This chapter explains how to specify all the elements impacting the behavior of your application search fields. It covers the UQL and ELLQL query languages used to make user queries, search-time semantic analysis, query expansion, search assistance tools like search suggestions, related terms, etc.

[User Query Language \(UQL\)](#)

[Exalead Low-Level Query Language \(ELLQL\)](#)

[Defining Query Templates](#)

[Using Prefix Handlers](#)

[Configuring Query Expansion](#)

[Configuring Dictionaries](#)

[Adding 'Did You Mean?' Spell-Check](#)

[Adding Search Suggestions](#)

[Adding Related Terms](#)

[Configuring and Using Similarity Measures](#)

[Configuring Geographic Search](#)

[Adding a Query Cache](#)

User Query Language (UQL)

User Query Language (UQL) serves for real user queries.

It allows you to make simple or rich queries using various query operators, such as Boolean operators (AND, OR, NOT), word sequence operators (NEAR, NEXT, BEFORE, AFTER), score operators (MAX, MIN), etc., and also prefix handlers to focus on specific metas.

[The Different Types of Search in UQL](#)

[Reserved Characters in UQL](#)

[Operands](#)

[Operators by Priority](#)

[More About INNERJOIN](#)

The Different Types of Search in UQL

This section describes the different types of search that you can make through UQL.

Important: Errors occur when you make queries using a single word in capital letters that is also an UQL operator or an operand. For example, if you search for `AND` you get an error like `[code=360142] Error while processing CloudView SearchAPI request... as` `AND` is an UQL operator. It works if you search for `and` in lowercase.

List of UQL operators that you cannot search alone in capital letters: `AFTER`, `AND`, `BEFORE`, `BOR`, `BUTNOT`, `FUZZYAND`, `NEAR`, `NEXT`, `NOT`, `OPT`, `OR`, `SPLIT`, `TO`, `XOR`.

Search by Exact Phrase

Operator	" " (quotation marks)
Purpose	<p>You can get more results than expected if you enter a search phrase (that is, two or more search terms meant to appear together), but do not enclose the phrase with quotation marks.</p> <p>To search for documents on 2018 sales, typically people would enter: <code>2018 sales</code></p> <p>In this case, the search results would include any document that contains both <code>2015</code> and <code>sales</code>, but not necessarily next to each other.</p>
Example	To search for documents containing the exact phrase <code>2018 sales</code> , use quotation marks: <code>"2018 sales"</code>

Search by Exact Words

Operator	+
Purpose	<p>You can override the matching behavior using the <code>+</code> (plus) operator to search for exact words only. It is typically useful to search for:</p> <ul style="list-style-type: none"> link words (<code>the</code>, <code>a</code>, <code>of</code>, <code>or</code>, and) that are ignored by default, the plural of a word. <p>This operator is useful for building very specific queries.</p> <p>You can also prepend words by <code>+</code> in your query to search for the exact forms of these words only. For example, with the query <code>foo +bar</code>, <code>foo</code> has the standard semantic expansion (like lemmatization if activated) but not <code>bar</code>, which returns the exact form only (that is, <code>bar</code>).</p>

Search with Logical Expressions

Operators	OR, AND, NOT, XOR, BOR
Purpose	Searches for documents containing: <ul style="list-style-type: none"> • OR: either one search term OR another • AND: one search term AND another search term • NOT: one search term BUT NOT another search term • XOR: either one search term OR another BUT NOT both • BOR: either one search term OR another. Only use it for a fast OR on many documents (no expansion, no ranking).
Example	Use OR to specify a list of similar terms that may occur in the document you are looking for. <code>(movie star) OR (celebrities)</code> searches for documents containing either <code>movie star</code> or <code>celebrities</code> .

Search with Excluded Words

Operators	NOT, -XX, BUTNOT
Purpose	Excludes documents containing a specific word or phrase from the search with a - (minus sign) or a NOT operator before the word to exclude.
Example	<p><code>new -york OR new NOT york</code> searches for documents containing <code>new</code> but not <code>york</code>.</p> <p>Note: NOT and - are unary operators and depend on the implicit default operator AND. The expressions <code>new -york OR new NOT york</code> are therefore interpreted as <code>new AND NOT york</code>.</p> <p>You can also use the BUTNOT operator:</p> <p><code>"Martin Luther" BUTNOT "Martin Luther King"</code> matches if there is at least an instance of <code>Martin Luther</code> not followed by <code>King</code>.</p>

Search with Prefix Handlers

Operators	Use prefix handlers: see the list of prefix handlers defined in Search logics > Query Language .
Purpose	Refine your queries by targeting specific index fields with default prefix handlers like <code>text:</code> , <code>title:</code> , etc.

	<p>You can also</p> <ul style="list-style-type: none"> • specify aliases for these prefix handlers. For a list of aliases, see that prefix handler's Alias field, in Search logics > Query Language. • search by category values • search numerical fields by a range of values • define custom prefix handlers to go further than the index field level, and trigger very specific search. <p>For more details, see Using Prefix Handlers.</p>
Example	<p>Search with a default prefix handler: <code>title:foo</code> searches for <code>foo</code> in document titles.</p> <p>Search with an alias: for the prefix handler <code>document_file_size</code>, you have the following aliases by default: <code>file_size</code>, <code>imap_mail_file_size</code>, <code>nntp_post_file_size</code>, <code>ldap_record_file_size</code></p> <p>Search by category values: <code>categories:fileattributes/extension/PDF</code></p> <p>Search a numerical range of values: <code>NumericalPrefixHandler:[100 TO 200]</code></p> <p>Custom prefix handler: for a similarity search, we could enter a query like: <code>similar:(ID1, ID2, ID3)</code> where ID1, ID2, ID3 are the IDs of related terms, to search for all the documents having a part or all of these related terms.</p>

Phonetic Search

Operators	<p>Prefix handler <code>soundslike</code>:</p> <p>You must create this prefix handler beforehand. For more details, see Using Prefix Handlers.</p>
Purpose	<p>Finds documents using the phonetic spelling of search terms.</p> <p>Important: The language used for the query is important and must match the language specified in your Mashup UI configuration. If none is specified, Exalead CloudView uses the web browser's preferred language.</p>
Example	<p>To find a coworker with a name that sounds like <code>Brona</code>, enter: <code>soundslike:brona</code> to return results such as <code>Bronagh</code> and <code>Branagh</code>.</p>

Search with Approximate Spelling

Operators	Prefix handler <code>spelllike</code> :
-----------	---

Purpose	Finds documents that do not exactly match the search terms. This is useful if uncertain of the correct spelling, or there are several accepted spellings for a search term.
Example	Searching for <code>spellslike:organise</code> also returns documents containing <code>organize</code> .

Search by Date

Operators	Prefix handlers <code>date:</code> , <code>document_lastmodifieddate:</code> , <code>document_before:</code> , <code>document_after:</code>
Purpose	<p>Retrieves documents based on a given date, or date range.</p> <p>By default, the input format is detected automatically. If you need to define a custom format, update the Input format field for your prefix handler in Search Logics > Query Language.</p> <p>What you must know:</p> <ul style="list-style-type: none"> We support the date formats: RFC 822, RFC 850, asctime, ISO 8601, and date format <code>YYYY/MM/DD-HH:MM:SS</code> (<code>DD/MM/YYYY</code> is NOT supported) Operators are <code>=</code>, <code>==</code>, <code><=</code>, <code><</code>, <code>>=</code>, <code>></code>, <code>!=</code> and <code>:</code> The default timezone is GMT. Quotes are required in search queries when there is at least a blank space in the date. For example, <code>myDatePrefixHandler="12/15/2018 15:23:22 GMT+02"</code>
Supported formats	<ul style="list-style-type: none"> <code>Sun, 06 Nov 1994 08:49:37 GMT</code> <p>RFC 850:</p> <ul style="list-style-type: none"> <code>Sunday, 06-Nov-94 08:49:37 GMT</code> <code>Fri Nov 21 11:18:47 CET 2014</code> <p>asctime:</p> <ul style="list-style-type: none"> <code>Sun Nov 6 08:49:37 1994</code> <p>RFC 822:</p> <ul style="list-style-type: none"> <code>Fri, 21 Nov 2014 16:59:27 MET DST</code> <code>Fri, 21 Nov 2014 17:59:14 EET</code> <code>Fri, 21 Nov 2014 15:59:16 +0000 (UTC)</code> <code>Fri, 21 Nov 2014 16:59:42 MET</code> <code>Fri, 21 Nov 2014 15:58:04 +0000 (UTC)</code> <code>Fri, 21 Nov 2014 07:58:28 -0800</code>

American date format:

- 12/23/2014 15:23:22
- 12/23/2014 15:23:22 GMT+02
- 09/23/2014 08:52:59 [+00:00]
- 2014/12/23 15:23:22
- 2014/01/23-22:11:37
- 2014/12/23
- 2014/12

ISO 8601 samples (ISO works with / or – separators):

- 2014-03-12 15:23:22
- 2014-03-12
- 2014-03
- 2014
- 2014-12-06T15:31Z
- 2014-12-06T15:31:17+00:00
- **Week numbers like 2016-W18-1T09:49:38Z are NOT supported**

Example Letâ€™s say that we give the modified alias to the document_lastmodifieddate prefix handler. We could have:

- modified="11/23/2018 10:18:02 GMT+01" for a fully explicit date query
- modified="2018/11/23 10:18:02+00:00" for a fully explicit date query
- modified="2018/11/23 10:18:02" for a date query with the default GMT time zone interpreted implicitly.
- modified="11/23/2018 10:18" for a query with an implicit range of 1 minute.
- modified=2018/11/23 for a query with an implicit range of one day.
- modified=2018/11 for a query with an implicit range of 1 month.
- modified=2018 for a query with an implicit range of 1 year.
- modified<"11/23/2018 10:18:02 GMT+01" for all documents before the explicit date.
- modified<"2018/11/23T10:18:02+01:00" for all documents before the explicit date.

- `modified<=11/23/2018` for all documents until the end of the 11/23/2014 day.
- `modified<=2018` for all documents until the end of the 31/12/2018 day.
- `modified:[2014/12/23 TO "2018/01/21-22:11:37 GMT+01"]` to search documents in a specific date range. This range notation is inclusive, and works with numerical values too.

We can also restrict a search query according to a document's last modification or creation date:

- `"movie star" AND date >= 2018/05/21` finds documents containing movie star modified after May 21, 2014.
- `and "movie star" AND date <= 2018/05/21` finds documents on movie star modified before May 21, 2018.

Search by Size

Operator	Prefix handler <code>file_size:</code>
Purpose	Searches based on file size in bytes.
Example	<ul style="list-style-type: none"> • <code>file_size:1024</code> returns documents with a file size of 1 KB. • <code>file_size>=1024</code> returns documents with a file size larger than 1 KB.

Search by Language

Operator	Prefix handler <code>language:XX</code>
Purpose	<p>Limits your search to the documents of a specific language using the <code>language:XX</code> prefix handler (where XX can be EN, FR, DE, etc.).</p> <p>This is useful when you need to search using a term that you can find in many languages, but has different meanings from one language to another.</p>
Example	<code>"Tour de France" language:en</code> searches for English-language documents about the Tour de France.

Search in URL

Operator	Prefix handler <code>inurl:</code>
Purpose	Includes all web pages with URLs containing the search keywords. Unlike <code>site:</code> , this is a full text search of the URL text.

Example	<code>inurl:example</code> returns: <ul style="list-style-type: none"> <code>http://www.example.com/</code> <code>http://www.exalead.com/blog/another_cloudview_example/</code>
----------------	---

Search for URL

Operator	Prefix handler <code>url:</code>
Purpose	Searches for pages with the same normalized URLs. You do not need to include the leading <code>http://</code> , <code>https://</code> , <code>www.</code> , and trailing slashes in the query.
Example	<code>url:example</code> returns: <ul style="list-style-type: none"> <code>http://www.example.com/</code> <code>http://www.exalead.com/blog/another_cloudview_example/</code>

Search Site Content

Operator	Prefix handler <code>site:</code>
Purpose	Returns all documents on a site. Only expect results for documents with a <code>publicurl</code> meta, such as those pushed by the Crawler and the Feed Fetcher connectors. The leading <code>"http://"</code> or <code>"https://"</code> and <code>"www."</code> , and trailing slashes are optional in the query.
Example	<code>site:example.com</code> always returns the same documents as <code>site:http://www.example.com/</code>

Search with Optional Terms

Operator	<code>OPT</code>
Purpose	Specifies an optional word to include in the search. Use it to specify several terms without limiting the scope of the search.
Example	<code>cow OPT mad</code> searches for documents containing <code>cow</code> that preferably also include <code>mad</code> .

Search by Word Proximity

Operators	<code>NEAR</code> , <code>NEXT</code> , <code>AFTER</code> , <code>BEFORE</code>
------------------	--

Purpose	<p>Find documents where search terms are in proximity of one another. By default the maximum distance between terms is 16 words.</p> <p>Edit this value using the Search > Search Logics > Query Language > Default distance for proximity operators property.</p>
Example	<p>"movie star" AFTER hollywood searches for documents where movie star appears soon after hollywood.</p> <p>Note: "movie star" is equivalent to movie NEXT star, the NEXT operator having a distance of 1 with the following word.</p> <p>You can also specify the maximum distance of the words by using NEAR/x, AFTER/x, and BEFORE/x. For example:</p> <ul style="list-style-type: none"> "movie star" NEAR/5 hollywood searches for documents where movie star appears within 5 words of hollywood, and "movie star" BEFORE/5 hollywood searches for documents where movie star appears within 5 words before hollywood. <p>Important:</p> <ul style="list-style-type: none"> You cannot use proximity operators with expressions whose "position" cannot be computed. For example, the query music NEAR (Madonna AND mp3) does not work, because the expression Madonna AND mp3 cannot be associated with a single word position value. Some queries using proximity operators may fail with a No occurrence for query message when you want to open the preview of Office documents. This issue is linked to a format conversion limitation.

Prefix Search

Operator	*
Purpose	Searches using the beginning of a word to find a proper noun using its short form, or its linguistic root.
Example	Jenn* searches for documents containing words starting with Jenn, such as Jennifer, Jennie, Jenni, and Jenna.

Pattern Search

Operator	<p>Regular expression patterns based on Perl 5.</p> <p>You must open and close patterns with a / (slash) character.</p>
----------	---

Purpose	Searches using the beginning of a word to find a proper noun using its short form, or its linguistic root.
Example	<ul style="list-style-type: none"> <code>/s.ren..pi.y/</code> searches for documents with words that match the pattern <code>S . R EN .. PI . Y</code> and would find documents with the word <code>serendipity</code>. <code>/mpg(1 2 3)?/</code> searches for documents containing any of the following: <code>mpg</code>, <code>mpg1</code>, <code>mpg2</code>, or <code>mpg3</code>.

Geographic Search

Operator	Prefix handler <code>geo:</code>
Purpose	See Configuring Geographic Search .
Example	To search within a radius or polygon using UQL, see Search with a Radius or Polygon (UQL) .

Search with INNERJOIN

Operator	INNERJOIN
Purpose	Combine records from two documents whenever there are matching values in a common field. See More About INNERJOIN .

Search by Document Sections

Operator	SPLIT
Purpose	Searches for words in specific sections of a document.

Reserved Characters in UQL

This section gives a list of reserved characters in UQL and describes how to escape their interpretation.

List of Reserved Characters in UQL

If you need to use them as words in your query, you must enclose them in quotes.

Name	Character
Slash	/
	Use it for:

Name	Character
	<ul style="list-style-type: none"> • passing options, for example, <code>NEAR/12</code> • pattern search as a regexp operator, for example, <code>text:/bug.*/</code>
Tab	<code>\t</code>
Line feed	<code>\n</code>
Carriage return	<code>\r</code>
Round brackets	<code>(or)</code>
Square brackets	<code>[or]</code>
Curly brackets	<code>{ or }</code>
Colon	<code>:</code>
Equal sign	<code>=</code>
Greater than or less than	<code>< or ></code>
Comma	<code>,</code>

Note: `%` is not a reserved character.

Escaping UQL Operators Interpretation

You can add a backslash (`\`) to disable the interpretation of UQL operators (parentheses, `=`, `{`, `}`, etc.).

Note: If any tokenization takes place afterward, the tokenizer still decides how to enter the character. For example, `\=` is interpreted as a simple `=` but the default tokenization considers this character as punctuation. It is therefore removed from the query as any other punctuation character.

Operands

This section describes the operands that you can use in UQL queries.

Standard Operands

Operand	Description	Predicate value
(e1)	Parenthesized sub expression, used to modify priority. For example: ((fast OR speed) AND NOT light)	e1
"e1"	Quoted expression, used to escape all special characters. Inside a double quoted group, words are handled in a tight (NEXT) sequence. All operators are ignored.	e1
"word1 word2"	Quoted expression.	word1 NEXT word2
"word1 OR word2"	Quoted expression	word1 NEXT "OR" NEXT word2

Regexp and Wildcard Operands

You can use wildcards and regular expressions in UQL queries. The following table illustrates some examples.

Kind	Syntax	Example
Regular expressions	[field:]/pattern/{options}	title:/desi.*/{w=10000,#=100}
Wildcard	[field:]word*{options} [field:]*word{options}	desi* title:*faces

Score Modifier Operands

Operand	Description	Predicate value
s=N	Replace the predicate's score by an explicit value.	price<500{s=1000}
s+=N	Increase the predicate's score by a given value.	GUI{s+=100000}
s-=N	Decrease the predicate's score by a given value.	corporate/tree:"Top/Attributes/XXX" {s-=100000}

Operand	Description	Predicate value
	Note: The score of a given predicate can be negative, but the final score of the document can never be lower than 0.	
$w=N$	Replace the predicate's weight by an explicit value.	<code>design{w=10000}</code>
$w*=N$	Multiply the predicate's weight by a given value.	<code>design{w*=2}</code>

Note:

- The score modifiers are applied in the same order as they appear. In the case of two conflicting modifiers, for example, `{s=1000,s=2000}`, the last one is applied.
- In the case when an explicit score (`s=`) is specified, the predicate's weight is ignored so the 'w' options have no effect.
- The explicit score (`s=`) can only be used for nontextual predicates (numeric values and categories) since this modifier completely ignores the ranking score class set when the document was indexed.

Word Matching Options

Word matching options are specified inside `{ }` and directly appended to search words. They must be comma-separated.

Option	Description	Example
<code>k=number</code>	Set explicit matching level.	<code>k=1</code> is the lowercase matching mode <code>k=2</code> is the normalized matching mode.
<code>hl=0</code>	Deactivates search result highlighting and summary for a specific node of the query only.	<code>word1 word2{hl=0} word3</code>

Operators by Priority

The query expansion modules rewrite the query based on the operators. To correctly expand the query, each operator has a priority.

Operators by Processing Priority, where $e1$ and $e2$ are Expressions

Priority	Operator	Explanation	Example
1	prefix handlers	Prefix handlers are always processed first.	Obama before:2009/01/01, searches for all documents relating to Obama before January 2009.
2	FUZZYAND/option (expression)	<p>Search for documents that match at least N queries, where N is determined by the fuzzyand option.</p> <p>This option can be either:</p> <ul style="list-style-type: none"> • minimum success: at least N queries must match. N is a positive integer. • or maximum failures: up to N queries can fail. N is a negative integer. 	<p>For a document that contains: The quick brown fox.</p> <p>If min. success=2: FUZZYAND/2 (the quick brown foxes) matches, but FUZZYAND/2 (a brown foxx) does not.</p> <p>If max. failure=-1: FUZZYAND/-1 (a quick brown fox) matches, but FUZZYAND/-1 (a quick foxx) does not.</p>
3	OPT e1	Optional operator	OPT graphical
4	NOT e1	Negation operator	NOT myword
5	e1 NEXT e2	Explicit sequence operator for adjacency match.	user NEXT interface
6	e1 AFTER e2	Proximity (mono-directional) match	interface AFTER user
6	e1 AFTER/distance e2	AFTER with explicit word distance	interface AFTER/4 user
6	e1 BEFORE e2	Proximity with mono-directional match	user BEFORE interface
6	e1 BEFORE/distance e2	BEFORE with explicit word distance	interface BEFORE/4 user
7	e1 NEAR e2	Proximity operator with bidirectional match	user NEAR interface
7	e1 NEAR/distance e2	NEAR with explicit word distance	user NEAR/4 interface
8	e1 SPLIT e2	A document is returned if e1 appears in at least one of the	user interface SPLIT Chapter

Priority	Operator	Explanation	Example
		document sections delimited by the <code>e2</code> delimiter. If the <code>e2</code> delimiter is not present in a document, then the document is returned if <code>e1</code> is valid at the document level.	Matches if <code>user interface</code> appears between two "Chapters"
9	<code>e1 e2</code>	Implicit match operator on a sequence of words. It uses the implicit operator, which is <code>AND</code> by default.	<code>search engine</code>
10	<code>(e1) INNERJOIN/key (e2)</code>	Search for documents matching <code>e1</code> where <code>e2</code> appears in child documents. The relation between documents is contained in a key index field, which must be an unsigned integer. For performance reasons, it is best to enable the Stored in Memory option.	<code>subject:exalead INNERJOIN/msgId fulltext:france</code> We first select documents whose subject is <code>exalead</code> , then the join is made with documents containing the word <code>france</code> .
11	<code>e1 BUTNOT e2</code>	The search matches if there is at least an instance of <code>e1</code> is not also an instance of <code>e2</code> at the same position.	<code>York BUTNOT "New York"</code>
12	<code>e1 AND e2</code>	Explicit conjunction match.	<code>user AND interface</code>
13	<code>e1 XOR e2</code>	Exclusive OR operation. It can be either <code>e1 OR e2</code> , but not <code>e1 AND e2</code>	<code>design XOR conception</code>
14	<code>e1 OR e2</code>	Disjunction match	<code>design OR conception</code>
15	<code>e1 BOR e2</code>	Disjunction match To use only for a fast <code>OR</code> on many documents	<code>design BOR conception</code>

More About INNERJOIN

You can use the INNERJOIN operator to join "left" and "right" documents based on a common numerical field. INNERJOIN only returns matching documents from the "left side" of the query.

For example, if you have `order` and `customer` documents that both contain a `customer_id` join key field, you can perform queries such as:

```
order_price>42 INNERJOIN/customer_id (customer_name:john AND customer_firstname:doe)
```

This returns all orders (the left-side documents) for John Doe that had an order price greater than 42.

Important: INNERJOIN queries have a significant impact on search performance.

When and How to Use INNERJOIN

Only use INNERJOIN for complex multicriteria search on the right-side document. In the above example, this is the `customer`.

Ideally, for better search performance, aim instead to inline the right-side document information in the left-side document. However, INNERJOIN is mandatory for certain multicriteria queries.

Requirements

- The field must be an integer, a date or a value, retrievable and RAM-based.
- For a value field, the `innerjoin` does not support two different join key fields for left and right side. For details about value fields, see [Create Value Facets for Nonhierarchical Metas](#).

Perform an INNERJOIN Query

When the join field name is the same for both left and right documents

- UQL: (left-side query) INNERJOIN/field (right-side query)
- ELLQL: #innerjoin(field, left-side query, right-side query)

When the join field name is different for the left and right documents (not applicable to value fields)

- UQL: (left-side query) INNERJOIN/field1=field2 right-side query)
- ELLQL: #innerjoin(field1,field2, left-side query, right-side query)

Important: When using a join field that was generated by a Data Model property, always specify the full field name, prefixed by the declaring class name, even in UQL.

Hide INNERJOIN from Users

To avoid exposing the `INNERJOIN` operator to the user, you can use query templates to combine ELLQL and UQL to perform the join from the user query. For details, see [Defining Query Templates](#).

Handling Documents in Multiple Slices

The `INNERJOIN` operator is intra-slice only: you cannot join a left-side document (LD) from one slice to a right-side document (RD) in another slice.

There are two main ways of handling this, depending on the use case.

Use Case 1: LD and RD are shardable in the Same Way

For example, when indexing customers and orders, we can shard the index by customer ID, and make sure that all orders go to the same slice as the customer.

To do that, we must override the automatic slice balancing algorithm, using a PAPI directive called `forcedSlice`.

Note: A PAPI directive is a small value associated to a document, which is not part of the textual content, that gives processing instructions or hints to the processing chain.

Set a PAPI Directive

```
document.setCustomDirective(name, value);document.setCustomDirective("forcedSlice", "value")
```

You then must correctly balance all slices.

Use Case 2: LD and RD are not shardable in the Same Way

In this case, the only solution is to fully duplicate the right-side documents (RD) in each slice. As the RD is never retrieved by the `INNERJOIN` query, there is no risk of duplication.

This works best when the set of RDs is relatively small, so duplicating them does not dramatically impact the slices.

You need to push each RD N times (where N =number of slices), each time with a different URI, for example `/myoriginaluri/slice3`.

For each document, you need to set the `forcedSlice` directive as described in the [Use Case 1: LD and RD are shardable in the Same Way](#).

You do not need to do anything special for your LD, since the automatic slice allocator keeps dispatching them.

Exalead Low-Level Query Language (ELLQL)

You can use EXALEAD Low-Level Query Language (ELLQL) for programmatic generation of queries, similar to SQL. ELLQL is typically used by custom programs to enrich user queries and to add additional features. Internally, all user-entered UQL queries are transformed into ELLQL.

Why Use ELLQL?

You can use ELLQL to handle complex queries that would either be too difficult or not supported in UQL, which is the case for:

- Geographic search using Point fields. For more details on Point search, see [Search a Geographic Point](#).
- Advanced Inner Joins, where you can specify which join type to use, are not possible in UQL. To do so, query the search API directly using ELLQL.
- Search for words that have a certain position in an index field. For example, if you want to search for a specific value only, such as all documents that have a title that starts with a particular word. If this word is "Rambo"; you would enter `(#at(1, #alphanum(title, "rambo")))`.

ELLQL vs UQL

You can pass ELLQL queries via the Search API using the `eq` parameter instead of `q` for UQL queries.

ELLQL operators work the same as UQL. Basic leaf nodes, however, work differently in ELLQL: they are index-level operators, not user-level operators. For example:

- In UQL, the query "a b" is tokenized and Exalead CloudView searches for two words. Using query expansion and prefix handling, Exalead CloudView can interpret this query to mean that we want to search in the `text` field.
- In ELLQL, you must write `#and(#alphanum(text, "a") #alphanum(text, "b"))`.

To build a complex query containing some user parts written in UQL, and some advanced parts in ELLQL, you can insert UQL within an ELLQL query with the `#uql` ELLQL operator. For example,

```
#or(#alphanum(text, "c") #uql("a OR b"))
```

ELLQL Syntax

For a complete list of ELLQL syntax, see the [Appendix - ELLQL Language](#).

Filtering Search Results in ELLQL

Using ELLQL, you can pass a parameter that filters which hits to include in the search results for a query. The filtering criteria is defined as a virtual expression. If the virtual expression returns a 0 (false) value for the hit, the hit is excluded from the search results. The query pattern is: ?

```
eq=#filter("virtual expression", ELLQL query).
```

For example,

```
?eq=#filter("document_file_size>100000",#alphanum(title,"CloudView"))
```

returns all documents with Exalead CloudView in the title, where the document file size is greater than 100,000 bytes.

Defining Query Templates

Exalead CloudView supports combining multiple chunks of queries, that we call "named queries", to build the final query. The combination of all named queries is specified by the query template defined in **Search > Search Logics > Your search logic > Query Template**.

Query Template Syntax

The default query template string is:

```
#and(#query(relevance_tuning) #and(#query(_default_) #query(refine) #query(security)
#query(date_restriction) #query(geo_restriction)))
```

The parameters following `#query` are replaced by named queries, whose names are specified between parentheses. For example, `relevance_tuning` is the name of the first named query. You can reference the default query (`q` in UQL or `eq` in ELLQL) in the query template with `#query(_default_)`.

The query template is itself an ELLQL query that you can set dynamically using the `qt` parameter. For example, with a query template of `qt=#and(#query(a) #query(b))`

- passing in `q.a=word1 OR word2` and `q.b=word3 OR word4`
- results in a final query of `(word1 OR word2) AND (word3 OR word4)`

Note: Through the API, you can pass additional queries using the `q.NAME` parameter family. You can pass additional queries as ELLQL, using the `eq.NAME` parameter family.

Options can be passed to the named queries specified after `#query` in the query template and forwarded to top nodes. The options forwarded from the query template always override the

options that might be already present on the top nodes. For example, `qt=#and(#query{w=0.5}(a) #query(b))`

- passing in `q.a=word1{w=2}` and `q.b=word2`
- results in a final query of `word1{w=0.5} AND word2`

Note: The query template and the named queries are parsed (and expanded) separately. Then, the query template and the named queries are merged into a single query, with the options of the query template forwarded to the top nodes of the named queries. Remind that no query expansion is performed on the query template itself, which is only parsed. Query expansion is performed in detached expanded queries.

For example:

- If the query template is: `#and(#query{foo=bar}(_example_) #false)`
- and the `_example_` named query in the search field is: `#fuzzyand(#or(#true))`
- the expanded query is: `#and(#fuzzyand{foo=bar} (#or(#true)) #false)`

Reserved Named Queries

- `refine`: all refinement parameters (`r`, `cr`, `zr`) are interpreted by the query parser, generating a named query called `refine`.
- `security`: all security tokens are interpreted and combined as a large OR query, generating a named query called `security`.
- `restriction`: generated by the content restriction specified in the search logic.

Use Case

It could be useful to add a restriction to user queries coming from a specific Mashup page of your front-end search application. Rather than modifying the user query, you could pass a named query and specify how you want queries to be combined to handle this restriction.

For example, we could add the named query `confidential_doc` to the query template to exclude confidential documents from the search results. This is already managed in the default query template by `#query(security)` checks user security tokens if any.

Using Prefix Handlers

Prefix Handlers allow you to refine queries by targeting specific index fields or change the behavior of the query expansion.

For example, the `title`: prefix handler allows you to refine the search on document titles.

You use prefix handlers in UQL by prefixing your query with a given prefix, followed by colon (:) or a numerical comparison operator like >. For example, the 'title' prefix handler allows you to narrow the search on document titles, by looking only for matches within the 'title' index field.

Important: There must be a default prefix handler in your configuration. In the default configuration, this prefix handler is `text`. It is used as fallback when no prefix handler is specified in your UQL queries.

The Different Types of Prefix Handlers

Specify a Tokenization Configuration for Prefix Handlers

The Different Types of Prefix Handlers

This section describes the default types of prefix handlers available in Exalead CloudView. When adding prefix handlers, you have to select one of these types.

You can add prefix handlers of several types:

Type	Description
Text	<p>Searches in alphanumeric fields (both static and dynamic ones).</p> <p>There are several prefix handlers of Text type in the default configuration: <code>text</code>, <code>trustedqueries</code>, <code>title</code>, <code>rawurl</code>, <code>document_pageurl_inurl</code></p>
Numeric	<p>Searches for numerical values in a numerical field (integer or double).</p> <p>In the default configuration, the <code>document_file_size</code> prefix handler is of Numeric type.</p>
Date	<p>Searches for dates and times in a date field.</p> <p>There are several prefix handlers of Date type in the default configuration: <code>date</code>, <code>document_after</code>, <code>document_before</code>, <code>document_lastmodifieddate</code></p> <p>By default, the input format is detected automatically. If you need to define a custom format, update the Input format field for your prefix handler in Search Logics > Query Language.</p>
Category	<p>Searches for paths in a category field.</p> <p>There are several prefix handlers of Category type in the default configuration: <code>datamodel_class</code>, <code>datamodel_class_hierarchy</code>, <code>corporate/tree</code>, <code>corporate/leaf</code>, <code>categories</code>, <code>source</code>, <code>language</code>, etc.</p>

Type	Description
Numeric (dynamic fields)	Searches for numerical values in a dynamic field.
Date (dynamic fields)	Searches for dates and times in a dynamic field.
Geographic	Defines the geographic field for WITHIN and DISTANCE searches.
Units of measurement	<p>Searches for documents by resolving and converting values in the required unit of measurement.</p> <p>For example, if <code>volume</code> is a Measure index field with a unit symbol set to <code>ml</code>, queries would look like:</p> <ul style="list-style-type: none"> <code>volume>25cl AND volume<16oz</code> <code>volume>250</code> (unit symbol is the default one, <code>ml</code> in our example) <p>The prefix handler detects the unit symbol if specified in the query, operates a conversion when required and then looks for properties according to the normalized numerical expression.</p>
Linguistic options	<p>Specifies a query expansion config on the prefix handler content, but does not perform search.</p> <p>In the default configuration, the <code>spellslike</code> prefix handler is of Linguistic options type. You can also create a <code>soundslike</code> prefix handler if your want to find documents using the phonetic spelling of search terms.</p>
Position	Searches in an alphanumeric field using the anchoring position (that is, restricting the match to be at a specific position).
Split	Searches for expressions, within the bounds of separators. For example, with a Split prefix handler named <code>mypage</code> with separator <code>INPAGE</code> , <code>mypage:(a AND b)</code> only matches if <code>a</code> and <code>b</code> appear on the same page in the document.
Site	<p>Searches for parts of URLs.</p> <p>There are several prefix handlers of Site type in the default configuration: <code>document_pageurl_site</code>, <code>document_pageurl_url</code>,</p>
Similarity	Manually looks up similar values in documents.
Template	<p>Template prefix handlers are used to rewrite the user original query (represented by the <code>__QUERY__</code> variable) so that it targets dedicated index fields with a specific logic if required (typically OR/ AND operators, etc.).</p> <p>For example, with a template set to:</p>

Type	Description
	<p><code><meta1>:__QUERY__ <OPERATOR> <meta2>:__QUERY__</code></p> <p>we could have something like:</p> <p><code>airport_name:__QUERY__ OR airport_city:__QUERY__</code></p> <p>Important: You must consider <code>__QUERY__</code> as a full UQL query string. Therefore, you cannot protect or encapsulate it with double quotes (") or single quotes (') characters, otherwise the variable will not be substituted. These UQL reserved words block the query expansion and make the query fail.</p> <p>Important: If the original UQL query contains a prefix handler, it might lead to invalid UQL query interpretation. For example, if <code>myTph</code> template prefix handler is <code>meta1: __QUERY__ OR meta2: __QUERY__</code> and the query input is <code>myTph:date>30</code>. Then the query template prefix handler interprets the query as <code>meta1:date>30 OR meta2:date>30</code>, which is not a valid UQL query and is rejected. A valid query input would be <code>myTph:(foo or bar)</code>.</p> <p>Important: There is a "default" prefix handler, which is used for query chunks, which have no explicit prefix handler. If this default prefix handler is a template prefix handler, then it is its duty to ensure that every single chunk of the query has an actual (that is, nontemplate) prefix handler after template expansion. Otherwise, you end up with meaningless query chunks. You cannot have <code>DEF_TPH: (file:__QUERY__ OR __QUERY__)</code></p>
Custom	<p>Prefix handler implemented using custom Java code.</p> <p>For more information, see "Add custom query processors or prefix handlers" in the Exalead CloudView Programmer's Guide.</p>

Specify a Tokenization Configuration for Prefix Handlers

Each search logic needs a default tokenization that determines how prefix handlers break down user queries into tokens.

This section covers how to specify a specific tokenization configuration for search-time processing. To learn how to create new tokenization configs and specify them at index-time, see [Customizing the Tokenization Config](#).

Specify Another Tokenization Config for a Search Logic

1. In the Administration Console, create a new default tokenization config as described in [Customizing the Tokenization Config](#).
2. Specify this default tokenization config for:
 - one or more semantic types, if using the Data Model to set up indexing. See [Specify Another Tokenization Config in the Data Model](#)
 - and the analysis pipeline, if manually configuring processors. See [Specify Another Tokenization Config for an Analysis Pipeline](#).
3. Go to **Search > Search Logics > Your search logic > Query Language** tab.
4. From the **Default tokenization config** list, select the new tokenization config.
5. Click **Apply**.

All prefix handlers use this tokenization config to tokenize queries.

Specify a Tokenization Config for a Prefix Handler

For metas that need to be tokenized differently than the default tokenization config, you can specify a different tokenization config for the prefix handlers targeting those metas.

1. In the Administration Console, create a new "exception" tokenization config as described in [Customizing the Tokenization Config](#).
2. Specify this default tokenization config for the index mappings for metas that need to be tokenized differently. See [Specify Another Tokenization Config for an Index Mapping](#).
3. Go to **Search > Search Logics > Your search logic > Query Language** tab.
4. Select the prefix handler for the appropriate meta.
5. From the **Default tokenization config** list, select the "exception" tokenization config you created in step 1.
6. Repeat for all the prefix handlers that need a different tokenization.
7. Click **Apply**.

Configuring Query Expansion

You can set up Exalead CloudView to broaden the scope of a user query, which is known as query expansion.

For example, you can expand queries to include synonyms, so when a user searches for `dba`, Exalead CloudView searches for `dba` OR `"database administrator"` OR `"db admin"`.

If phonetic query expansion is enabled, the query "exaleed" would expand to "exaleed" OR "exalead".

Query expansion does the following:

- enriches the query, using synonyms and semantics
- interprets and normalizes the query, such as by recognizing city names or acronyms.

Query expansion is configured on prefix handlers as targeting specific index fields is useful to make consistent query expansions. For example, while it makes sense to use synonyms when searching the `title` index field with the `title:` prefix handler, it does not when searching on the `author` index field with the `author:` prefix handler.

Query Tree and Query Expansion

Query Expansion Features

Enable query expansion

Stemming

Lemmatization

Phonetization

Approximation

Normalization Exceptions

Synonyms

Japanese Synonyms

Query Tree and Query Expansion

When Exalead CloudView parses a query written in UQL, it is represented as a structured query tree, where the inner nodes are query operators and the leaves are Boolean predicates.

Query expansion generates a new, larger query tree by processing query expansion modules. This is known as the query rewriting step.

Query expansion modules process the query tree to:

- enrich it, for example, by using synonyms and semantics,
- interpret and normalize it, for example, by recognizing city names or acronyms.

In the end, the query expansion generates a new query tree. This is the query rewriting step.

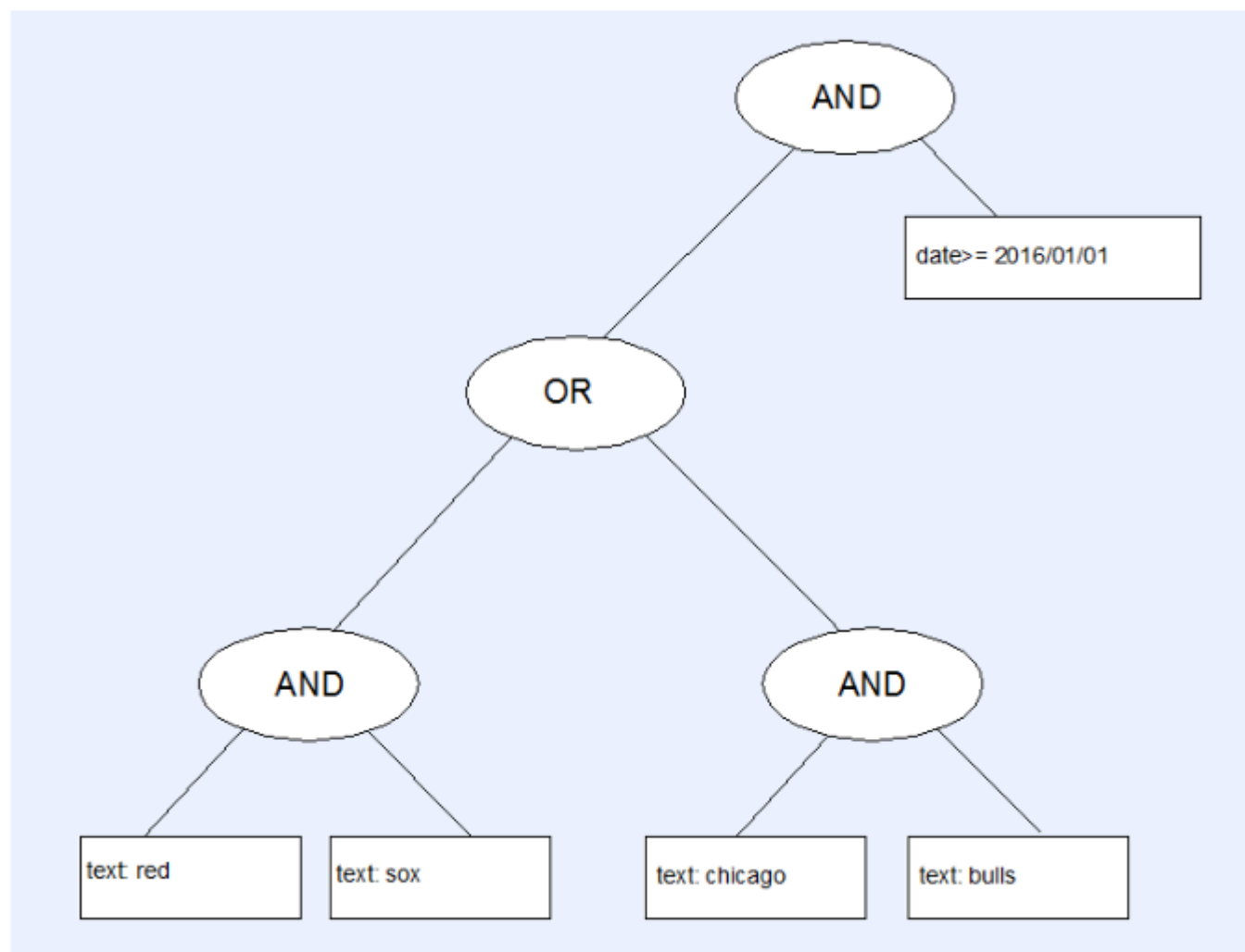
Note: After this step, all `Leaf` and `Rex` nodes in the tree have been converted to `FinalLeaf` nodes.

Query Expansion Example

The figure below shows the query tree for a user's search for documents containing the words "Red Sox" or "Chicago bulls", and modified after January 1, 2016:

```
((Red Sox) OR (Chicago Bulls)) AND lastmodified>=2016/01/01
```

The Default Operator between Two Text Predicates (like "Red" and "Sox") is a Boolean AND.



Character Interpretation

Leaf nodes can have different prefixes such as:

Character	UQL interpretation
–	<p>Excludes terms if – is at the beginning of a leaf.</p> <p>The leaf is interpreted as a NOT context.</p> <p>For example, Sarbanes–Oxley is considered as a single leaf, while Sarbanes –Oxley is considered as two leaves, one of which is negated.</p>

Character	UQL interpretation
+	The leaf is treated as exact, which disables some expansion operations.
., &, -, and other word separators	<p>interpreted as a <code>NEXT</code> operator instead of an <code>AND</code>, when used to separate words without additional white spaces.</p> <p>For example, <code>ASP.NET</code> is parsed as <code>ASP NEXT NET</code> instead of <code>ASP AND NET</code>.</p>

The query processor allows nonalphanumeric characters in words in a few special cases that you can configure.

Leaves can also have options. You must specify the options in brackets and separated by quotes.

The following leaf options are available:

- query rewrite options, which modify the way the query processor works and expands the leaf.
- raw options, which are passed to the index.

Matching Modes

Each word can exist in the index at several matching modes (or index kinds). The matching between word predicates and document words is defined by the matching mode of predicates. The matching mode can be one of the following:

- **Exact match:** Matches only if the words match exactly. For example, `The` only matches with `The`. This is known as `kind 0` (or `k=0`).
- **Case-insensitive or Lowercase match:** Ignores case for matching. For example, `the` and `The` match. This is known as `kind 1` (or `k=1`). This level can be specified by the "i" option in UQL.
- **Normalized match:** Ignores case and accents, for example `the`, `thé` and `Thé` match. This is known as `kind 2` (or `k=2`). This is the default matching mode but it can be changed using predicate options.

Note: You can specify other matching modes in **Linguistics > Advanced > Form indexing**.

Query Expansion Features

Query expansion features are described below.

Query expansion modules allow you to define semantic query expansions to perform on prefix handlers. For example, if lemmatization was configured on the `title` prefix handler, and the user enters the query `title: (mouse and man)`, the query expands to `title: ((mouse OR mice) and (man OR men))`.

Wildcard search is a pattern-matching feature enabled by default. It allows you to find documents that include "test", "tests" and "tesselation" when searching on "tes*". Additional configuration is available to fine-tune your results.

Wildcard search is expanded using a dictionary generated from the corpus. See [Configuring Dictionaries](#).

Spellcheck can be enabled to suggest alternate spellings for words in the query. Spell-check is much more effective if you first extract spell-check ngrams at index-time.

For details, see [Adding 'Did You Mean?' Spell-Check](#).

Enable query expansion

When you install Exalead CloudView, most query expansion modules are already activated, with the exception of Synonyms, or the Custom query expansion module.

By default, these modules are set up for all supported languages. You can, however, choose to add multiple instances of a query expansion module (for example, several lemmatization modules) that are set up for different languages.

However, if you want a certain prefix handler to use a query expansion module, you still need to configure that prefix handler's query expansion accordingly.

To enable query expansion you need to do the following:

- Activate query expansion modules: these are global processing units that must be defined for each search logic. They define the static parameters for the type of expansion. For example with synonym expansion, you must create a synonym dictionary, known as a resource file, for the synonym module.
- In the prefix handler, define which modules to use for query expansion: this is defined in the **Query expansion config** for the prefix handler.

Activate a query expansion module

1. In the Administration Console, go to **Search > Search Logics > Query Expansion**.
2. Under **Query expansion modules**, click **Add module** if the module you want to activate does not already appear on the list.
3. (Optional, except for **Synonyms**) Specify a resource file for the module. To define and compile a synonym resource file, see [Synonyms](#).
4. Click **Save**.

You must now associate this module with one or more prefix handlers as described in the following procedure.

Set up a prefix handler to use a query expansion module

1. Go to the **Query language** tab and click the prefix handler that will use this module.
2. In the expanded view of this prefix handler, click **Edit** beside **Query expansion config**.

The screenshot shows a configuration window for a prefix handler. The title is 'title' and the text is 'Text'. The configuration includes several fields: 'Target index fields' (value: title), 'Index fields weights' (empty), 'Expand end of expression' (checkbox), 'Use indexed pattern search' (checkbox), 'Use nested prefix as dyn. meta' (checkbox), 'Dyn. meta name' (empty), 'Target dictionary' (value: dict0), 'Query expansion config' (empty), 'Set as default handler' (radio button), and 'Aliases' (empty). Each field has an information icon (i) to its right. The 'Edit' button next to the 'Query expansion config' field is highlighted with a yellow box.

3. In the **Query expansion config** dialog, specify the semantic expansion for this prefix handler. You can also dynamic parameters for the expansion. In the following example, the expansion expression uses the approximate query expansion module, and includes specific options that define the maximum number of matches and their relative importance.

The screenshot shows the 'Query expansion config' dialog. The 'Expression' field contains the code: `approximate{max_matches=5,weight_distance1=0.2,weight_distance2=0.02}`. Below the expression field are two lists: 'Modules' and 'Module options'. The 'Modules' list includes: approximate, japanesesynonyms, lemmatization, normalizationexceptions, phonetic, and stemming. The 'Module options' list includes: flm, max_matches, min_chars_distance1, min_chars_distance2, weight_distance1, and weight_distance2. The 'weight_distance2' option is selected. Below the lists is a description for 'weight_distance2 (positive float)': 'Relative ranking weight of the fuzzy forms at distance 2. This controls the "w" parameter of the standard term score formula. Default value: 0.01. Aliases: w2, wd2'. The 'Insert' button is visible. At the bottom are 'Accept' and 'Cancel' buttons.

For descriptions of the options available for each query expansion module, see:

- [Stemming](#)
- [Lemmatization](#)
- [Phonetization](#)
- [Approximation](#)
- [Normalization Exceptions](#)
- [Synonyms](#)
- [Japanese Synonyms](#)

4. Click **Apply**.

Stemming

Stemming expansion allows you to search for words with a common root, or stem.

For example, searching on the word "Britannia" would expand the query to include words with the stem "Britann", such as "Britannic" and "Britanny".

There is often some confusion in understanding the difference between stemming and lemmatization. Both have an objective of finding a common base for query words with several related derivations or inflected forms.

Dependencies

For stemming to work at search-time, you must first create stemmed forms at indexing time. For details, see [Snowball Stemmer](#).

Stemming vs Lemmatization

Stemming is the simpler method, as it seeks to find the root (the stem) of a word by cutting off endings. For example:

- Searching on `alsaciennes` (women from the Alsace region of France) would also search for words with the stem `alsac`.
- Searching on `alsace` would also search for words with the stem `alsac`.

By contrast, lemmatization uses a more complex morphological analysis to find the singular or masculine form of nouns and adjectives.

- Searching on `alsaciennes` also searches for words with the lemma `alsacien`.
- Searching on `alsace` also searches for words with the lemma `alsace`.
- Moreover, searching on `geese` would also search for `goose`.

Stemming Rules

Depending on the language, two kinds of rule dictionaries are used:

- Rules based on the "Snowball" library.
- Internal CloudView rules

For stemming to work, the words must have been extracted from text at indexing time (this is the default configuration).

Stemming Options

When configuring a prefix handler's query expansion, the following stemming options are available.

Option	Description
<code>max_matches</code> or <code>m</code> , or <code>matches</code>	Searches no more than N stems for a word. If the number of available stems exceeds this value, Exalead CloudView searches the N most frequent stems in the corpus. The default value is 10.
<code>weight</code> or <code>w</code>	Relative ranking weight of the stemmed forms. This controls the <code>w</code> parameter of the standard term score formulas. The default value is 0.1. For details on term score and weight, see Ranking and Sorting Search Results .

Lemmatization

Lemmatization expansion allows you to search for the masculine or singular forms of feminine or plural nouns and adjectives.

For example, searching for `geese` would also search for the lemma `goose`.

There is often some confusion in understanding the difference between lemmatization and stemming. Both have an objective of finding a common base for query words with several related derivations or inflected forms. See [Stemming vs Lemmatization](#).

Dependencies

For lemmatization to work at search-time, you must first create lemmatized forms at indexing time. For more information, see [Lemmatizer](#)

Languages Lemmatized Natively

Lemmatization is available in the following languages:

- English (en)
- French (fr)
- German (de)
- Italian (it)
- Portuguese (pt)
- Russian (ru)
- Spanish (es)

Languages Lemmatized with Basis Tech Add-On

If you want to perform lemma query expansion for languages tokenized by the Basis Tech (Extended Languages) tokenizer, you must configure the Lemmatizer semantic processor, as well as follow some additional steps for tokenization.

For details, see [Enable Lemmatization with Basis Tech](#).

Lemmatization Options

When configuring a prefix handler's query expansion, the following lemmatization options are available.

Option	Description
<code>weight</code> or <code>w</code>	Relative ranking weight of the lemmatized forms. This controls the "w" parameter of the standard term score formulas. The default value is 0.1. For details on term score and weight, see Ranking and Sorting Search Results
<code>masculinization</code>	Looks up for the masculine form of a word when a feminine form is entered. For example, entering <code>canadienne</code> searches on <code>canadien</code> .
<code>masculine_weight</code> or <code>mw</code>	When transcribing a word from the feminine to masculine form, the resulting masculine form has this weight value instead of "w". The default value is 0.01.

Phonetization

Phonetic expansion allows you to search for alternative forms that sound like the original query. For example, searching for "exaleed" would also search for "exalead". This query expansion module works by default with the `soundslike` linguistic prefix handler.

Dependencies

For phonetization to work at search-time, you must first extract phonetic forms at indexing time. For more information, see [Phonetizer](#).

Supported Languages

Phonetization is available natively for the following languages:

- Canadian (ca)
- Czech (cs)
- Danish (da)
- Dutch (nl)
- English (en)
- Estonian (et)
- Finnish (fi)
- French (fr)
- German (de)
- Italian (it)
- Norwegian (no)
- Portuguese (pt)
- Slovak (sk)
- Slovenian (sl)
- Spanish (es)

Phonetization Options

When configuring a prefix handler's query expansion, the following phonetization options are available.

Option	Description
<code>max_matches</code> or <code>m</code> , or <code>matches</code>	Searches no more than N phonetic forms for a word. If the number of available phonetic forms exceeds this value, Exalead CloudView searches the N most frequent forms in the corpus. The default value is 10.
<code>weight</code> or <code>w</code>	Relative ranking weight of the phonetic forms. This controls the <code>w</code> parameter of the standard term score formulas. The default value is 0.1. For details on term score and weight, see Ranking and Sorting Search Results .

Approximation

Approximation expansion finds words that are lexicographically similar to other words. This query expansion module works by default with the `spellslike` linguistic prefix handler.

Approximation is useful for full-text search to correct user queries with typos. For example, if the user enters `cropped`, the search results displays hits with `cropped`, the correct spelling, automatically.

Approximation is the search for a query word with a fuzzy match in the corpus. It is performed by calculating the Damereau-Levenshtein distance between the query word and the corpus word.

Approximation Vs Spell Check

Approximation is similar to spell check. The difference is that approximation only applies to the word that follows the prefix handler with which it is associated (`myprefix: word`). Meanwhile, spell check applies to the entire user query and has more configuration options.

The approximation module considers both the word length and the number of transformations allowed to expand the original query with additional words. Transformations mean that you replace (substitute) a letter, add a letter, transpose a letter, or delete a letter.

The approximation module searches for words that are at transformation distance 1 or 2 of the original word from the user's query. Distances are hard-coded, so you can only control the word length that triggers distance 1 or distance 2. In other words, depending on the word length, you set either `max_distance=0, 1` or `2`.

See also [Adding 'Did You Mean?' Spell-Check](#).

Approximation Options

When configuring a prefix handler's query expansion, the following approximation options are available.

Note: The approximation module has default values for these options. To override them, you must define them explicitly in the query expansion config expression.

Approximation Options

Option	Description
<code>flm</code>	<p>Specifies an additional transformation distance (think of it as a penalty) on any transformation on the first letter. This shows that in most cases, people do not make typos on the first letter.</p> <p>For example, "abc" and "zbc" are at:</p> <ul style="list-style-type: none"> distance 1 if <code>flm=false</code>, distance 2 if <code>flm=true</code>. <p>The default value is <code>true</code>.</p>
<code>max_matches</code> or <code>m</code> or <code>matches</code>	<p>Searches no more than N fuzzy matches for a word. If the number of available fuzzy forms exceeds this value, Exalead CloudView searches the N most frequent forms in the corpus.</p> <p>The default value is 10.</p>
<code>min_chars_distance1</code> or <code>mcd1</code>	<p>Only searches for distance 1 fuzzy matches if the original word in the query is at least N characters long.</p> <p>This avoids too much approximation on very short words.</p> <p>The default value is 5.</p>
<code>min_chars_distance2</code> or <code>mcd2</code>	<p>Only searches for distance 2 fuzzy matches if the original word in the query is at least N characters long.</p> <p>This avoids too much approximation on short words.</p> <p>The default value is 10.</p>
<code>weight_distance1</code> or <code>w1</code> or <code>wd1</code>	<p>Relative ranking weight of the fuzzy matches at distance 1.</p> <p>This controls the "w" parameter of the standard term score formula.</p> <p>For details on term score and weight, see Ranking and Sorting Search Results.</p> <p>The default value is 0.1.</p>

Option	Description
<code>weight_distance2</code> or <code>w2</code> or <code>wd2</code>	<p>Relative ranking weight of the fuzzy matches at distance 2.</p> <p>This controls the <code>w</code> parameter of the standard term score formula.</p> <p>For details on term score and weight, see Ranking and Sorting Search Results.</p> <p>The default value is <code>0.01</code>.</p>

Example

Take the word "screwdriver".

If the query expansion config expression is set to `approximate{min_chars_distance1=5}` only

- `srewdriver` => approximation works, there is 1 transformation (deletion of the "c" character).
- `screwdrive` => approximation works, there is 1 transformation (deletion of the last character "r").
- `screwqdriver` => approximation works, there is 1 transformation (insertion of the "q" character).
- `scrweddriver` => approximation works, there is 1 transformation (transposition of the "w" and "e" characters).
- `screqdriver` => approximation works, there is 1 transformation (substitution of the "w" character by the "q" character).
- `sredriver` => approximation does not work as there are 2 transformations (deletion of "c" and "w").

If we set the query expansion config expression to

`approximate{min_chars_distance1=,min_chars_distance2=10}`

- `sredriver` => approximation still does not work because:
 - there are 2 transformations (deletion of "c" and "w")
 - and there are only 9 characters in the query word whereas the minimum number to get 2 transformations is set to 10.

If we set the query expansion config expression to

`approximate{min_chars_distance1=,min_chars_distance2=9}`

- `sredriver` => approximation works because:
 - there are 2 transformations (deletion of "c" and "w")
 - and there is the minimum number of 9 characters to make these 2 transformations.

Example: Approximation for an Error on the First Letter

Let us say that we have created a prefix handler called `approxprefix` using the approximation module with its default option configuration.

We enter the query: `approxprefix:correct` and get matches for documents containing the word `correct`. However, if we search for `approxprefix:vorrect`, we do not get any matches.

This behavior is normal, since in most cases, people do not make typos on the first letter.

Therefore, there is an additional transformation distance by default for any transformation on the first letter (`flm=true`).

Since `vorrect` is a short word (fewer than 10 characters) and substituting "c" for the first letter "v" equals to a transformation distance of 2, the approximation module does not expand the search.

If we search for `approxprefix:vorrection`, the approximation module expands the search to include `correction`, as the search term is 10 characters long.

To disable the additional transformation distance for first letters, edit the query expansion config expression to include `flm=false`. In our example, we would have the following expression:

```
approximate{flm=false}
```

We could also tackle this kind of issue by reducing the minimum length to trigger distance 1 and distance 2 (using the `min_chars_distance1` and `min_chars_distance2` options).

Normalization Exceptions

When the query includes a word that is subject to a normalization exception, it is not usually normalized. If the normalization exceptions module is present, the query is performed on both the normalized and the non-normalized form.

For example in French, "maïs" (corn) is subject to a normalization exception because it conflicts with "mais" (but).

- Without this semantic processor, a search for `mais` only searches for `mais`, and does not find `maïs`.
- With this semantic processor, a search for `mais` searches for `(mais OR maïs)`.

Synonyms

The Synonym expansion module adds alternative forms to user queries. For example, if the text prefix handler uses the synonyms module, the query: `"db architect"` expands to `"db architect" OR "data base architect" OR "database architect"`.

Unlike the other query expansion modules, you must first compile your own synonym dictionary, also known as a resource file, that defines the possible synonyms for a particular expression.

1. Create a synonym XML file containing the following code:

```
<Synonyms xmlns="exa:com.exalead.mot.qrewrite.v10" equivalenceClass="false" matchOnSeparators="true" stopwordsResource="resource:///stopwords/ontology.bin" permutations="false" addStops="true">
  <SynonymSet originalExpr="db architect" lang="en">
    <Synonym alternativeExpr="database architect" />
    <Synonym alternativeExpr="data base architect" />
  </SynonymSet>
</Synonyms>
```

Where:

Attribute	Description
matchOnSeparators	<p>Possible values:</p> <ul style="list-style-type: none">• <code>true</code> (default): synonym matching is punctuation sensitive.• <code>false</code>: punctuation is ignored during matching. For example, the synonym "twenty-seven" matches "twenty seven".
stopwordsResource	<p>Path to the compiled ontology containing stop words used at build time when generating permutations and stop word-free forms.</p> <p>Default value is <code>resource:///stopwords/ontology.bin</code>.</p> <p>Note: Exalead CloudView only provides French and English stop words.</p> <p>You can use your own stop word resource by building an ontology containing a package <code>exalead.stop</code> and the list of forms for each language you want to support:</p> <pre><Ontology xmlns="exa:com.exalead.mot.components.ontology" matchOnSeparators="true"> <!-- this stopword list is used by the synonym compiler to generate stopword-free forms and permutations for english and french synonyms --> <Pkg path="exalead.stop"> <Entry lang="en"> <Form value="of" level="lowercase"/> <Form value="the" level="lowercase"/> <Form value="a" level="lowercase"/> ... </Entry> <Entry lang="fr"> <Form value="de" level="lowercase"/> <Form value="du" level="lowercase"/> <Form value="la" level="lowercase"/> ... </Entry></pre>

Attribute	Description
	<pre></Pkg> </Ontology></pre>
permutations	<p>Possible values:</p> <ul style="list-style-type: none"> <code>true</code>: For each synonym, extra forms made of word permutations are added. Before computing permutations, stop words are removed. For example, the synonym "lyrics of the song" matches "song lyrics". <code>false</code> (default): Word permutations are not added.
addStopwordFreeForms	<p>Possible values:</p> <ul style="list-style-type: none"> <code>true</code>: For each synonym, an extra form (from which stop words have been removed) is added. <code>false</code> (default): Extra forms are not added.
originalExpr	expression specified by the user
alternativeExpr	Expressions that are matched to the <code>originalExpr</code> .
equivalenceClass	<p>Possible values:</p> <ul style="list-style-type: none"> <code>true</code>, synonym searching works in both directions: queries using <code>originalExpr</code> return documents including <code>alternativeExpr</code>, and vice versa. <code>SynonymToSynonymSet</code>, when you search for one of the <code>alternativeExpr</code> expressions, the query also expands with the <code>originalExpr</code>. <code>SynonymSetToSynonym</code> (or <code>false</code>, kept for backward compatibility), when you search for the <code>originalExpr</code>, the query is expanded respecting the <code>alternativeExpr</code> order.
level	<p>(optional) This attribute specifies which form must be matched. For more information, see Available Matching Normalization Levels.</p> <p>For example, if you use a lemmatizer and want your synonyms to match lemmatized forms, set this attribute in your <code>SynonymSet</code> objects to <code>lemmaSingular</code>.</p>

Example with `level` attribute

```
<Synonyms xmlns="exa:com.exalead.mot.grewrite.v10" equivalenceClass="false" matchO
stopwordsResource="resource:///stopwords/ontology.bin" permutations="false" addSto
  <SynonymSet originalExpr="Dog" lang="en" level="lemmasingular">
```



```

    <Synonym alternativeExpr="cat" />
    <Synonym alternativeExpr="bird" />
  </SynonymSet>
</Synonyms>

```

Results when "dogs" is entered in the search box:

- The first result is the lemmatization of "dogs", that is to say "dog".
- As the lemma "dog" matches the `SynonymSet` original expression, the results are then expanded to: "cat" and "bird".

Note: The display of synonyms follows the sort order specified in the `SynonymSet` node.

2. Go to `<DATADIR>/bin` and compile the XML file using the following `cvadmin` command:

```
cvconsole> cvadmin linguistic compile-synonyms input=<PATH TO SYNONYM.XML> output=
```

3. Check that the `.BIN` file is created in the specified directory.
4. Complete the steps in [Activate a query expansion module](#).

Japanese Synonyms

You can activate the Japanese synonyms module to get a good support of synonyms in Japanese.

When configuring a prefix handler's query expansion, the following Japanese synonym options are available.

Option	Description
<code>max_distance</code> or <code>md</code>	Maximum distance allowed for synonyms expansion. This limits the query expansion to synonyms within the specified distance.
<code>max_expansions</code> or <code>me</code>	Maximum number of synonyms to expand for a query word. For example, if you set this option to 5, only the first five synonyms serve for the query expansion.

Configuring Dictionaries

Dictionary is a separate structure from the index that stores all the words from an indexed document, plus their number of occurrences in the corpus. It serves for linguistic expansion mechanisms such as spell-checking or regular expression matching.

About Dictionaries

Setting Up a Dictionary

Compacting and Building Dictionaries

Clearing Dictionaries

About Dictionaries

During installation, features requiring a dictionary are set up with the default dictionary, **dict0**. You can change the configuration of dictionary resources in the default dictionary or create additional dictionaries to suit your needs.

Dictionary Resources

All these resources are already configured for the default dictionary, **dict0**. Use this list to change default settings or to build new dictionaries.

Resource	Description
Words	Stores words & their frequency to calculate relevance and term expansion. If word occurrences are under the specified Min Frequency, they do not appear in the dictionary.
Ngrams	Used to improve spell check accuracy. PREREQUISITE: Select the Extract spell check ngrams for the semantic types associated with this dictionary.
Phonetic Forms	Used to improve spell check accuracy, to calculate relevance and term expansion. It is required for phonetic term expansion. PREREQUISITE: a phonetic semantic processor must be defined in the pipeline, or you must select the Extract phonetic forms option for the semantic types associated with this dictionary.
Related Terms	Required to provide related terms in this language. PREREQUISITE: Define a related terms semantic processor in the pipeline.

Multiple Dictionaries

Exalead CloudView supports multiple dictionaries. Each dictionary is configured separately with its own name, maximum size, and so forth.

- On the indexing side, you can configure a semantic type to use a specific dictionary. So when you associate a data model property with a semantic type, it ensures that the generated index field is associated to a specific dictionary. This dictionary can only contain words likely to appear in that field.

- Symmetrically, each prefix-handler at search time can target a specific dictionary (for regexp search, etc.).

Moreover, the dictionary allows you to define filtering rules for controlling which words are stored in the dictionary. This allows you to store only words with a minimum number of characters, or words matching a regular expression.

Setting Up a Dictionary

Create a New Dictionary

1. In the Administration Console, go to **Index > Linguistics > Dictionaries**.
2. Click **Add Dictionary**.

TIP: For **Creation mode**, select **copy**.

To determine which elements you need in this dictionary, see [About Dictionaries](#).

3. Click **Apply**.

Associate a Dictionary to Metas via Semantic Types

1. In the Administration Console, go to **Index > Data Model > Semantic Type**.
2. Expand a semantic type, and in the **Dictionary** field, select the dictionary.

Note: If you do not want to store words in a dictionary, select **None**.

3. Select the prerequisite options, depending on which elements are in your dictionary. See [About Dictionaries](#).
4. Click **Apply**.

Associate a Dictionary to Metas via Mappings

1. In the Administration Console, go to **Index > Data processing > pipeline name > Mappings**.
2. Under the **Mapping sources** column, expand the meta you want to associate with a dictionary.
3. Under the **Mapping targets** column, select the dictionary name, and then under the **Details** column, select the elements where you want this meta to be stored in the dictionary. See [About Dictionaries](#).
4. Repeat for all mappings you want to associate to dictionaries.
5. Click **Apply**.

Change the Default Dictionary

The first dictionary in your list of dictionaries is the default dictionary. Since a new Exalead CloudView installation only includes one dictionary, **dict0**, it automatically becomes the default dictionary.

1. To set another default dictionary, use the **Default dictionary** list under **Dictionary**.

Set Up a Dictionary Resource

This procedure shows how to set up the Words resource. You can configure other resources similarly.

1. In the Administration Console, go to **Index > Linguistics > Dictionaries**.
2. Select (or add) your dictionary.
3. Expand **Words**.
4. Under **Actions**, click the edit tool next to the language you want to configure.
5. From the **Edit language config** dialog box, configure:
 - **Max No. terms**: Set the maximum number of terms allowed for the selected language.
 - **Min frequency**: How often the word needs to occur for it to be stored for that language in the dictionary.
 - **Regex filter**: Define a pattern of words to exclude from the dictionary for this language.
6. Click **Accept**.

Compacting and Building Dictionaries

The dictionary capabilities include compact and building policies.

- Compact policies: Dictionary data is regularly compacted after N import operations and/ or N seconds, to keep a single file per resource.
- Build policies: Dictionaries are regularly rebuilt after N compact operations and/ or N seconds to be up-to-date.

The following procedures explain how to configure compact and build operations.

Compact Individual Dictionaries

1. In the Administration Console, go to **Index > Linguistics > Dictionaries > Dictionary > dictn > Configuration**.
2. Select **Enable compact** and specify the compact policy.

- Choose to compact when N import streams have been done since the last compact operation.
- Choose to compact every N second.

3. Click **Save** and **Apply**.

Fine-Tune the Compact Size

1. Edit the `<DATADIR>/config/Dictionary.xml` file
2. Add a `FrequencyCompactFilter` to the `CompactPolicies` node, as shown in the following example.

```
<dict:CompactPolicies disjunctives="true">
  <dict:ImportCountCompactPolicy countThreshold="1"/>
  <dict:FrequencyCompactFilter lang="fr" minFrequency="10"/>
</dict:CompactPolicies>
```

In this example, the compact file is lightened of all French terms that do not have at least 10 occurrences.

Build Individual Dictionaries

1. In the Administration Console, go to **Index > Linguistics > Dictionaries > Dictionary > dictn > Configuration**.
2. Select **Enable build** and specify the build policy.
 - Choose to build when N compact operations have been done since the last build operation.
 - Choose to build every N second.
3. Click **Save** and **Apply**.

Force a Compact and a Build Operation

Sometimes, you do not want to wait for the end of a compact or a build operation, and start them at once.

1. In the Administration Console, go to **Index > Linguistics > Dictionaries**.
2. In the **Dictionary status** panel, click **Compact & Build** for the dictionary you want to compact and build immediately.
3. Click **Save** and **Apply**.

Clearing Dictionaries

You sometimes need to clear your dictionaries after you have edited:

- the configuration of dictionary resources in **Linguistics > Dictionaries > dictN > Resources**, for example, related terms or ngrams parameters.
- the tokenization config associated to dictionary features in **Linguistics > Dictionaries > dictN > Features**.

Recommendation: Clear your dictionaries when documents have been deleted from your corpus to ensure their reliability.

Clear Individual Dictionaries

1. In the Administration Console, go to **Index > Linguistics > Dictionaries**.
2. From the **Dictionary status** section, click **Clear** for the dictionary you want to clear out.

Clear All Dictionaries

1. In the Administration Console, go to the **Home** page.
2. From the **Indexing** section, click **Clear**.
3. Select **Dictionary data for ALL build groups** and click **Clear**.

Clear All Dictionaries (Alternative Procedure)

1. Go to **Index > Linguistics > Dictionaries**.
2. Click **Clear all dictionaries**.

Adding 'Did You Mean?' Spell-Check

You can enable your search logic to include spell-check, which suggests alternate words or expressions to replace the original user query.

About Spell-Check

Setting Up Spell-Check

About Spell-Check

A score determines spell-check suggestions. The score is calculated based on:

- First, the Damereau-Levenshtein transformation distance from the original word.
- Second, the frequency of terms in your corpus.

You can also force or prevent them using allow list and block list resources. In that case, the transformation distance and the term frequency are bypassed.

How is the Transformation Distance Calculated

The transformation distance (Damereau-Levenshtein distance) is the number of changes (replace a letter, add a letter, transpose a letter, delete a letter) required to transform word A to word B. Each change represents a distance. The greater the transformation distance, the larger the difference between two words.

Exalead CloudView performs distance 1 and distance 2 spell-checking. This means spell-check suggests alternate words that have a distance of 1 (or 2, for longer words) from the original word.

For example, let us assume each transformation change (replace, add, transpose, delete) has the same distance value of 1. If you index the Exalead CloudView documentation and enable spell check, a search on "exaleed" would suggest "exalead". "exalead" qualifies as a valid suggestion of spell check because it is at distance 1 from the original search term (an "a" replaced the third "e"), and occurs quite frequently in the documentation.

By contrast, "exaloda" (if this word really exists in the corpus) would not qualify because it is at distance 2 from the original search team: it requires substituting the "o" with an "e", plus transposing the "d" and "a".

Improve Spell-Check with Ngram and Phonetic Extraction

Spell-check is far more effective when your corpus is indexed with phonetic and Ngram extraction. You can configure this in the Data Model, as described in [Set Up Ngram and Phonetic Extraction in the Data Model \(Optional\)](#).

An ngram is a sequence of N words. Its sole purpose is to improve spell checking for multiword expressions. At indexing time, 2-, 3-, and 4-word ngrams are extracted from your corpus and stored in the dictionary. Exalead CloudView is then able to compare a user's query with these ngrams, calculate the probability that the user meant to use a different spelling, and then suggest alternate expressions.

Important: These options improve quality but also impact performance. Extracting spell check ngrams is CPU-intensive. Storing ngrams and phonetic forms significantly increases dictionary size, since it stores the various phonetic forms for tokens in addition to the tokens themselves.

Force or Prevent Spell-Check Suggestions with Allow Lists and Block Lists

You can force or prevent certain spell-check suggestions through the use of allow lists and block lists.

- Allow list: Always suggests X when a user queries for Y.

For example, people often enter "excede" instead of "exceed". To automatically suggest "exceed" (regardless of which spelling occurs most frequently in your dictionary) when a user types "excede", add this word pair to your allow list.

- **Block list:** This does not block specific spell check suggestions. Instead, it blocks specific query words from correction.

Say that you want to search for documents containing brand names, and you do want the spellcheck to suggest other names. For example, for "Informatica", by default, spellcheck suggests "information". Solution: add "Informatica" to the spellcheck block list to ensure that spellcheck never corrects this term.

Preferred Settings

Default settings have been chosen so that search performance is not impacted by the spell-checker activation.

To get a more effective spell-check:

- Check that **Extract spell check ngrams** is clicked in **Data Model > Semantic Types > text**.

Note: The ngram dictionary is built from a sample of document texts. Only one document out of 5 takes part extraction. Therefore, you have to index a minimum number of documents to reach a reasonable level of quality for spell-check. Being based on statistical algorithms, the more documents you have in the index, the better the spell-checker works.

- Select **Keep only most important suggestions** in **Search > Search Logics > your search logic > Query Expansion** to increase suggestions relevancy. It is assessed by querying the index and checking the count of documents returned.
- If spell-check suggestions are displayed too often, set a value for **Only check spelling if query returns less than** in **Search > Search Logics > your search logic > Query Expansion**.

Setting Up Spell-Check

This section explains how to enable and configure the spell-check feature.

Enable Spell Check

1. In the Administration Console, go to **Search > Search Logics > your search logic > Query Expansion**.
2. Under **Spell check**, select **Enable**.
3. (Optional) Select another dictionary to use for spell check.

4. (Optional) Modify the remaining spell check options. These are described in the tooltips on the interface.
5. Select an allow list or a block list. For more information, see [Force or Prevent Spell-Check Suggestions with Allow Lists and Block Lists](#).
6. Click **Apply**.

Set Up Ngram and Phonetic Extraction in the Data Model (Optional)

Ngrams and phonetic forms improve spell-check quality. See [Improve Spell-Check with Ngram and Phonetic Extraction](#).

1. In the Administration Console, go to **Index > Data Model > Semantic Types**.
2. Click the **text** semantic type to display its configuration.
3. Select **Extract spell check ngrams** and **Extract phonetic forms**.
4. Click **Apply**.

This adds the **Phonetizer** and **NgramsExtractor** semantic processors to your analysis pipeline.

5. Reindex your documents.

Enable Dictionaries for Spell-Check (Allow or Block List)

Before you can specify an allow list or block list for spellcheck, you must first enable these on the target dictionary.

1. In the Administration Console, go to **Index > Linguistics > Dictionaries**.
2. Select an existing dictionary or create a new one. See [Create a New Dictionary](#).
3. Go to **Features** and click **Spellcheck** to display its setup pane.
4. In the setup pane, specify a new tokenization config. To understand tokenization, see [Tokenizing Text](#).
5. Under **Allow list** or **Block list**.
 - If you have already created a resource file: specify a name for this allow list or block list, and in **List path**, select the resource file (list of resources created in the Resource Manager) or alternatively, if you created a resource file using `cvadmin`, type the path to the resource file using the format `resourcemanager://group_name/resource_name`.
 - OR, create a new resource: click **Create new**, specify a name for the allow list or block list, and click **Accept**. This adds a resource file to the Resource Manager, which ensures correct deployment of interdependent resource files in multihost environments.
 - Click **Add allow list** or **Add block list** to add more lists.
6. Click **Apply**.

7. To define the contents of the resource file, click **Edit**. This takes you to the Business Console. For more information, see "Add a spellcheck block list" and in the Exalead CloudView Business Console User's Guide.

Set Up Spell-Check Allow Lists or Block Lists on a Search Logic

1. In the Administration Console, go to **Search > Search logics > Your search logic > Query Expansion** tab.
2. Under **Spell check**, select **Enable** if not already enabled.
3. Expand **Block and allow lists**.
4. Next to **Allow list** or **Block list**, select a list. It includes all allow list or block lists created on the dictionary used for spell check.

If you do not see any lists in the list, it means that none exists in the selected dictionary.

Either select another dictionary, or add at least one list to the current dictionary. See [Enable Dictionaries for Spell-Check \(Allow or Block List\)](#).

5. Click **Apply**.
6. (Optional) To define the contents of the resource file, click **Edit**. This takes you to the Business Console. For more information, see "Add a spellcheck block list" and in the Exalead CloudView Business Console User's Guide.

Set Up Spell-Check for CJK (Chinese-Japanese-Korean)

Check and apply the [Preferred Settings](#) first.

1. In the Administration Console, go to **Search > Search logics > Your search logic > Query Expansion** tab.
2. Under **Spell check**, expand **Edit distance** and set:
 - a. 0 for **First letter**.
 - b. 1 for **Insertion**.
 - c. 1 for **Deletion**.
3. Expand **Triggers** and set 4 for **Min word length anywhere for distance 2**.

Adding Search Suggestions

The goal of search suggestion is to auto-complete the user's query by providing relevant suggestions for what the user wants to search. It shows some of the terms associated with the beginning of the user search query.

Note: Exalead CloudView also offers another query suggestion feature, called "Trusted Queries". It guides end users by suggesting categories from indexed facets. For more information, see "Adding Trusted Queries" in the Exalead CloudView Mashup Builder User's Guide.

About Search Suggestions

Create a Suggest Dictionary

Enable the Suggest in the Mashup UI

Use the Suggest Via the Search API

Export Suggest Dictionary Content to an XML File

Dispatch a Query to Several Suggest Dictionaries

Performance Considerations and Options for Search Suggest

About Search Suggestions

Search suggest relies on precomputed dictionaries to offer efficient matching (millisecond-range, thousands of queries per second). It can be based on:

- Exalead CloudView index content – fetching the values of an index field or a category facet
- Previously performed queries
- Custom XML dictionaries provided by the Administrator.

Suggest dictionaries are recomputed periodically.

A suggest dictionary contains suggest entries. These entries are the suggestions to be made to the user. Each entry has a given score. The number of possible matches for a given input string is a fixed parameter when building a suggest. For each input string, only the N best matches can be returned.

Available Suggest Types

Suggest Type	Description
Suggest from the index	<p>These take the result of an index query and build a suggest dictionary from a part of each hit.</p> <p>You can use this to either build a suggest dictionary based on the whole index, by using "#all" as the query, or to restrict to a subset of the index.</p> <ul style="list-style-type: none"> • Index field suggest – Takes the value of an index field. You can enable security for this type of suggest (expand the Build options node). It uses both documents and user security tokens to restrict

Suggest Type	Description
	<p>suggestions. To build suggests on alphanumeric index fields, see also Performance Considerations and Options for Search Suggest.</p> <ul style="list-style-type: none"> • Category title suggest – Takes all category titles from a subpath of one category index field. • Category path suggest – Takes all category paths from a subpath of one category index field. • CSV index field suggest – Takes one value from a multiencoded CSV field, also known as the "metas" field in the default configuration. • Related terms suggest – Takes the value of the keyword field.
Suggest from query reporting logs	<p>The option Query reporting suggest automatically builds a suggest dictionary from all stored query logs. These query logs also serve for search reporting (see Analyzing User Queries with Reporters).</p>
Suggest from a custom XML dictionary	<p>If you want to suggest dictionary from an external data source, use the Static XML suggest option:</p> <ul style="list-style-type: none"> • Create your own XML dictionary (see XML dictionary structure below). • Place the XML file on the server hosting the main gateway. <p>In the Administration Console, add a Static XML suggest and give the path to the XML file, prepending it with "file://" (for example <code>file:///data/mydir/mydictionary.xml</code>).</p> <p>The root node of the XML file is <code>SuggestDictionary</code>, it contains:</p> <ul style="list-style-type: none"> • a set of <code>SuggestDictEntry</code> • <code>maxEntries</code> • <code>subExpr</code> • <code>subString</code> • <code>permutation</code> (optional, see the Compute permutations option in Configure Build Options) <p>Each entry of the dictionary is defined by a <code>SuggestDictEntry</code> node, which contains the attributes:</p> <ul style="list-style-type: none"> • <code>entry</code> – the expression to match • <code>score</code> – the score • <code>display</code> – what must be displayed

Suggest Type	Description
	<p>A suggest dictionary entry can contain alternative expressions. For example, if you want to add synonyms ("resto" for "restaurants"), you can add a <code>SuggestDictEntryAlternativeForm</code> node inside <code>SuggestDictEntry</code>. This specific node contains two attributes:</p> <ul style="list-style-type: none"> <code>form</code> - the alternative expression to add <code>score</code> - the score of the alternative form, if not set we use the score of <code>SuggestDictEntry</code> <p>Example: Here the Suggest query "a" returns 3 results: "aircraft", "airlines", "air".</p> <pre><sugg:SuggestDictionary xmlns="exa:com.exalead.mot.suggest.v10" maxEntries="3" subExpr="false" subString="false" > <sugg:SuggestDictEntry entry="airport" score="1" display="Airport" /> <sugg:SuggestDictEntry entry="air" score="2" display="Category:Air" /> <SuggestDictEntry entry="airlines" score="3" display="Airlines"/> <sugg:SuggestDictEntry entry="aircraft" score="4"/> <sugg:SuggestDictEntry entry="airelles" score="1" /> <sugg:SuggestDictEntry entry="airpower" score="1" /> <sugg:SuggestDictEntry entry="trucpower" score="1" / > <sugg:SuggestDictEntry entry="restaurant" score="10"> <sugg:SuggestDictEntryAlternativeForm form="resto" score="2" /> </sugg:SuggestDictEntry> </ sugg:SuggestDictionary></pre> <p>Example with extra information: A Suggest dictionary entry can also contain a set of extra information (a set of URL for example), stored in <code>SuggestDictEntryExtraInfo</code>. Extra information are strings associated to each entry. They are returned by the suggest API when doing a query. You can also store the information in <code>SuggestDictEntryKeyValue</code> if you want to store a set of keys/values. In this example, the entries "airport", "air", "aircraft" and "airelles" have extra information.</p> <pre><sugg:SuggestDictionary xmlns="exa:com.exalead.mot.suggest.v10" maxEntries="3" subExpr="false" subString="false" > <sugg:SuggestDictEntry entry="airport" score="1"> <sugg:SuggestDictEntryExtraInfo info="http://</pre>

Suggest Type	Description
	<pre> www.c.com"/> <sugg:SuggestDictEntryExtraInfo info="http://www.d.com"/> </sugg:SuggestDictEntry> <sugg:SuggestDictEntry entry="air" score="2"/> <sugg:SuggestDictEntry entry="airlines" score="3" / > <sugg:SuggestDictEntry entry="aircraft" score="4"> <sugg:SuggestDictEntryKeyValue key="url.first" value="http://www.a.com"/> <sugg:SuggestDictEntryKeyValue key="url.second" value="http://www.b.com"/> </SuggestDictEntry> <sugg:SuggestDictEntry entry="airelles" score="1" > <sugg:SuggestDictEntryExtraInfo info="http://www.e.com"/> </sugg:SuggestDictEntry> <sugg:SuggestDictEntry entry="airpower" score="1" /> <sugg:SuggestDictEntry entry="trucpower" score="1" /> </sugg:SuggestDictionary> </pre>
Suggest from a custom precompiled resource	<p>A static resource suggest takes an already-compiled suggest dictionary as a parameter. This dictionary is loaded by the search server <code>suggest</code> command to answer queries. All the suggest types available in the Administration Console help retrieving the entries that must be compiled to produce this suggest resource.</p> <p>This suggest dictionary cannot be scheduled or built.</p>

Force or Prevent Suggestions with Allow Lists and Block Lists

You can add block list and allow list resources to your suggest dictionaries.

- Allow list: Always suggests the listed entry when a query matches one of its alternative forms. No need to rebuild the suggest dictionary. Note this gives the same behavior as if you manually add entries into the suggest dictionary with a maximum score.
- Block list: Deletes the specified suggest expression at search time (suggest time). No need to rebuild the suggest dictionary.

Create a Suggest Dictionary

This section gathers all the procedures you need to add and configure a Search Suggest dictionary.

Add a New Suggest Dictionary

1. In the Administration Console, go to **Search > Suggest**.

- Click **Add suggest** and select one of the suggest types. For more information, see [Available Suggest Types](#).

Add Allow Lists or Block Lists to a Suggest Dictionary

- Expand **Block and allow list**.
- Next to **Allow list** or **Block list**, specify your resource file.
 - If you have already created a resource file, click **Browse**. Select the resource file, which contains all allow list and block list resources created in the Suggest group of the Resource Manager. Then click **Accept**. If you have created a resource file using `cvadmin`, type the path to the resource file using the format `resourcemanager://group_name/resource_name`.
 - OR, create a new resource: click **Create new**, specify a name for the allow list or block list, and click **Accept**. This adds the resource to the Suggest group in the Resource Manager, which ensures correct deployment of interdependent resource files in multihost environments.
- Click **Apply**.
- (Optional) To define the contents of the resource file, click **Edit**. This takes you to the Business Console. For more information, see "Add a Suggest Block List" and in the Business Console.

Configure Query-Time Options

- Expand **Query-time options** to specify how the suggest handles queries.

Option	Description
Distance	<p>Allows approximate matching. The higher the distance the more approximate the match.</p> <ul style="list-style-type: none"> 0: exact match. 1: distance tolerance of 1 between the result and the query 2: distance tolerance of 2 between the result and the query <p>For more information about approximate matching, see Approximation.</p>
Autocomplete	<p>Appends suggest results to the last query word entered in the search field to autocomplete it.</p> <p>It only applies to suggests built with the Subexpr matching or Substring matching build options.</p>
Recursive	Discards the leftmost word of the query progressively.

Option	Description
	<p>It sends each new subquery to the suggests until you reach the max number of suggestions, or until there is no more word to use.</p> <p>For example, for a query "A B C", the suggest is called 3 times, with "A B C", "B C", and "C".</p>

Configure Build Options

Important: These options can have a tremendous performance impact, read carefully [Performance Considerations and Options for Search Suggest](#).

1. For all suggests (except those based on custom dictionaries), you can configure build options.

Build option	Description
Subexpr and Substring matching	<p>Normally, suggest matching is prefix-based: "first" returns entries "first test" and "first image".</p> <p>Sometimes, you want to be able to do a wider matching, not always prefix-based.</p> <ul style="list-style-type: none"> • Subexpr matching allows you to find matches on every start of word. For example, "first test" returns both for "fir" and for "tes". • Substring matching allows you to find matches on every letter. For example, "first test" returns for "fir", for "rs", for "es", ...
Sentence split and Ngram split	<p>For performance reasons, use these options to avoid long entries. By "long", we mean entries longer than 100 characters (100 bytes).</p> <p>Sentence and ngram split options allow you to break up a suggest entry into several entries, and to perform matches independently on the chunks.</p> <ul style="list-style-type: none"> • For sentence split, if the entry is multisentence, an entry is created for each sentence. • For ngram split, a sliding window of ngrams of a given size is created and an entry created for each step of the window. For example, "a b c d e f" with a split on 4-grams gives entries "a b c d", "b c d e", "c d e f". <p>Note: 0 means no splitting.</p>
Compute permutations	<p>Computes all permutations for an entry and adds them as separate entries. For example, if you start entering "Angeles", Exalead CloudView automatically suggests "Los Angeles".</p>

Build option	Description
	<p>Note: Entries longer than 8 words are not permuted for performance reasons.</p> <p>This action is performed after the sentence split if the Sentence split option is selected.</p> <p>To apply permutation to Static XML suggest and Static resource suggest types, you need to add <code>permutation="true"</code> to the <code>SuggestDictionary</code> tag in your XML file or suggest resource in the Business Console.</p>
Max. entry length	<p>The maximum number of characters in a suggest entry.</p> <p>This is a security measure to prevent overly long entries. They are automatically truncated after the specified length.</p> <p>0 means no limit.</p>
Max. suggestions	The maximum number of suggestions that can be shown to the user for a given input string. You cannot change this dynamically.
Tokenization config	Specifies the Tokenization configuration to use.
Sanitize entry	<p>This option strips the entry of punctuation, and encloses any UQL operators in quotes.</p> <p>It is useful when you want to suggest among a list of product references containing "-" (hyphens) or other delimiters, and you do not want any tokenization on these characters.</p>
Build after import	Triggers a build automatically after the index refreshes.
Enable security	Makes use of documents and users' security tokens to restrict suggestions.

Compile the Suggest Dictionary

Once created, suggests must be compiled in the Administration Console.

Important: Building suggest fails if there is not enough disk space to calculate it. It is best to allocate substantial disk space for the suggest build to copy/compute raw files from temporary files (in `build/resources/tmp`). If **Build options** are enabled, for example **subexpr matching** and **substring matching**, the required disk space is even bigger. Read carefully [Performance Considerations and Options for Search Suggest](#).

1. Go to **Search > Suggest** and click **Build now**.

For each suggest, you can also schedule suggest builds using the **Build scheduling** options.

Enable the Suggest in the Mashup UI

To display suggests in the Mashup UI, you need to enable this option in the Mashup Builder.

1. In Mashup Builder, go to a page using a search form widget. For example, the `/index` page, which uses the **Standard Search Form** widget.
2. Click the widget header to display its properties panel.
3. On the **Suggest** tab, complete the following:
 - a. Select **Enable suggest**.
 - b. For **Suggest Name**, click inside the field.
 - c. From the dynamic list that displays on the left, select the suggest service.

Note: If you do not see the suggest you created, refresh the list.

4. Repeat these steps for the `/search` page.
5. Click **Apply**.

Use the Suggest Via the Search API

If you are using a custom UI, you probably want to access the suggest backend API, which directly provides the suggestion.

It is available as a Search API command, by default on `/suggest`.

For example, if the name of your suggest is "mysuggest", then the API is available on:

```
http://<searchserver_host>:<searchAPI_port>/suggest/service/mysuggest
```

It supports HTTP GET queries, with the following input parameters.

Parameter	Value	Description
q	string	The input query
distance	integer (0, 1, 2)	The suggest dictionaries supports fuzzy matching at runtime. This specifies the maximum Levenshtein distance between the input string and the suggestion. 0 means exact match
minLenForDis	integer	Only searches for distance 1 fuzzy matches if the original word in the query is at least N characters long. This avoids too much approximation on very short words. The suggested value is 3.

Parameter	Value	Description
minLenForDist	integer	Only searches for distance 2 fuzzy matches if the original word in the query is at least N characters long. This avoids too much > approximation on very short words. The suggested value is 6.
logic	string	Specify a Search Logic name.
exhaustive	true/false Boolean	Displays exhaustive results.
recurse	true/false Boolean	Suggests new matches on query words recursively.
autocomplete	true/false Boolean	Suggests matches for the last word only.
output	string (xml or json)	<p>Output format:</p> <ul style="list-style-type: none"> xml – returns a complete output, with text suggestions, score, distance. json – returns text suggestions only. <p>Other search output format such as csv, flea, and atom, are not supported.</p> <p>Note: The Accept HTTP header is also taken into account if output is not specified.</p>
callback	string	When using JSON output, the name of a Javascript function to call. The returned Javascript fragment is "callback && callback(json_object)".

Export Suggest Dictionary Content to an XML File

It can be useful for debugging purpose or generating other resources, to see the entire content of a suggest dictionary.

1. Make sure that the Exalead CloudView instance is running.
2. Go to <DATADIR>/bin/ and run cvadmin.
3. Start the following command:

```
cvconsole cvadmin> suggest dump-suggest-to-xml [args]
```

Where the args are:

- [name=]: The Suggest name (type: STRING)

- `[output=]`: Path to the output XML file (type: FILE)
- `[dictionary=]`: Dictionary name for related-terms based suggest (type: DICTIONARY)

Dispatch a Query to Several Suggest Dictionaries

Suggest dispatchers allow you to use several suggests in a single query and therefore quickly refine your data at search time.

You can:

- Map prefix handlers to suggest dictionaries and then start queries made of several prefixes and associated suggests.
- Define a default suggest to get suggestions without entering prefix handlers in the search field. By combining this default suggest with prefix handler/ suggest pairs, you can further extend search suggestion possibilities.

Add and Configure a Suggest Dispatcher

1. Click **Add suggest dispatcher**, enter a name, and click **Accept**.
2. Specify the options to apply:

Option	Description
Match whole query	<p>Sends the whole query to the default suggest if the cursor is outside a prefix handler.</p> <p>For example, if the query is:</p> <pre>author: "George Lucas" Star Wars</pre> <p>If the cursor is after the last quote, you are outside the <code>author:</code> prefix handler scope, and:</p> <ul style="list-style-type: none"> • If the option is selected, the suggest is made on the whole query, <code>author: "George Lucas" Star Wars</code> • If cleared, the suggest is applied to <code>Star Wars</code> only. <code>author: "George Lucas"</code> is not considered.
Use default suggest for non configured prefix	<p>Sends the query to the default suggest if the cursor is within an undefined prefix handler.</p> <p>If cleared, undefined prefix handlers are ignored and there is no suggestions.</p>
Add quotes to suggestions	<p>Adds quotes where required so that the whole suggestion is included in the prefix handler.</p>

Option	Description
Add prefix handler to suggestions	Adds prefix handlers automatically when you enter a query
Check with search logics	Selecting specific search logics allows prefix handler suggestion and configuration check while configuring prefix handler/suggest pairs below.
Max. suggestions	<p>Allows you to define a maximum number of suggestions to be displayed (default is 0, meaning no limit).</p> <p>Note: You can also define a maximum number of suggestions to be displayed for each prefix handler. See below.</p>
Boost variety	<p>Allows to retrieve the best matches for each suggest according to the maximum number of suggestions defined previously.</p> <p>Note: This mode does not return the best global results but the best results for each suggest.</p> <p>Example: a suggest dispatcher is configured to display 10 suggestions maximum, for 3 suggest dictionaries.</p> <p>Without Boost variety, you get:</p> <ul style="list-style-type: none"> • Suggest 1: 3 results • Suggest 2: 10 results • Suggest 3: 8 results <p>With Boost variety, you get:</p> <ul style="list-style-type: none"> • Suggest 1: 3 results • Suggest 2: 4 results • Suggest 3: 3 results
Prefix handler Suggest	<p>Maps a prefix handler to a suggest dictionary. You can map as many pairs as required.</p> <p>Select Default to specify the suggest dictionary to use by default for a specific prefix handler.</p> <p>Note: You must specify at least one default suggest, using the following options:</p> <ul style="list-style-type: none"> • Match whole query

Option	Description
	<ul style="list-style-type: none"> • Use default suggest for non configured prefix • Add prefix handler to suggestion <p>If required, define a maximum number of suggestions to be displayed for each prefix handler in the Max. suggestions field.</p>

3. Click save and apply your configuration.

Note: You do not need to rebuild the suggest dictionaries for suggest dispatchers.

Enable a Suggest Dispatcher in Mashup Builder

1. In Mashup Builder, go to a page using a search form widget, for example the **Standard Search Form** widget.
2. Click the widget header to display its properties panel.
3. On the **Suggest** tab, complete the following:
 - a. Select **Enable suggest**.
 - b. For **Action**, select **dispatcher**.
 - c. For **Suggest Name**, click inside the field. From the dynamic list that displays on the left, select the suggest dispatcher previously created in the Administration Console.
4. Click **Save** and then apply your changes.
5. Open the Mashup UI and enter a query using the prefix handlers defined previously.

You get suggests for each of these prefix handlers.

Example: Two Prefix Handlers Mapped to Two Suggests

In the following example, we have set a suggest dispatcher to map two prefix handlers (`categories` and `text`) to two different suggests. We are then able to enter queries with the following format: `categories: suggestX text: suggestY`

1. Open the Mashup UI and enter a first prefix handler, for example `categories:` and a few characters to get a first list of suggestions.



categories: con	Search	Advanced Search
categories: "common"		
categories: "connectors"		
categories: "biz_console"		
categories: "connector_prog"		
categories: "configuration_reference"		

2. Enter a second prefix handler, for example `text:` and a few characters to get a second list of suggestions.

categories: "connectors" text: mana	Search	Advanced Search
categories: "connectors" text: "About Log Management "		
categories: "connectors" text: "Configuration Management "		
categories: "connectors" text: "License Management "		
categories: "connectors" text: " Managing Applications"		
categories: "connectors" text: " Managing Components"		
categories: "connectors" text: " Managing Configurations"		
categories: "connectors" text: " Managing Semantic Resources"		
categories: "connectors" text: " Managing Semantic Resources in cvconsole"		
categories: "connectors" text: " Managing the Crawler Connector"		
categories: "connectors" text: "The Management API MAMIs"		

3. You can also make this query with the Search API `suggest/dispatcher` command.

The URL format is the following:

```
http://<HOSTNAME>:<BASEPORT+10>/suggest/dispatcher/<DISPATCHER_NAME>
```

For example, to get the suggest obtained in step 2, we could enter the following URL:

```
http://myhost:10010//suggest/dispatcher/mydispatcher?q=categories:"connectors" text:
```

```

- <SuggestXMLAnswer>
  <SuggestXMLEntry entry="categories:"connectors" text:"About Log Management" score="1"
    source="indexfield" matchOffset="40" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"Configuration Management" score="1"
    source="indexfield" matchOffset="44" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"License Management" score="1"
    source="indexfield" matchOffset="38" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"Managing Applications" score="1"
    source="indexfield" matchOffset="30" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"Managing Components" score="1"
    source="indexfield" matchOffset="30" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"Managing Configurations" score="1"
    source="indexfield" matchOffset="30" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"Managing Semantic Resources" score="1"
    source="indexfield" matchOffset="30" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"Managing Semantic Resources in cvconsole"
    score="1" source="indexfield" matchOffset="30" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"Managing the Crawler Connector" score="1"
    source="indexfield" matchOffset="30" matchLength="4"/>
  <SuggestXMLEntry entry="categories:"connectors" text:"The Management API MAMIs" score="1"
    source="indexfield" matchOffset="34" matchLength="4"/>
</SuggestXMLAnswer>

```

Performance Considerations and Options for Search Suggest

To perform extremely efficient matching, we have to compute the exact matches for each input substring.

For example, if the suggest entries are:

- "first test" score=10
- "first of a kind" score=20
- "second test" score=10
- "first test of the world" score=25

And the number of matches is set to 2

- "first" returns "first of a kind" and "first test of the world"
- "first t" returns "first test" and "first test of the world"

The build time and temporary space required can roughly be computed as:

$$(\text{number of entries}) \times (\text{length of entries})^2$$

When you enable substring matching, we have to recreate this prefixing for each letter of the entry. Therefore, the build time and temporary space can be computed as:

$$(\text{number of entries}) \times (\text{length of entries})^2 \times (\text{length of entries})$$

When you enable subexpr matching, we have to recreate this prefixing for each word of the entry. Therefore, the build time and temporary space can be computed as:

$$(\text{number of entries}) \times (\text{length of entries})^2 \times (\text{words per entry})$$

The build time is therefore highly dependent on the entries size. It is therefore an extremely bad idea to compute a suggest on the "text" field without any options. Such a suggest can take hours to build, even with a few thousand documents. If you want to build suggest based on the textual content of the index, you must use:

- Sentence splitting or ngram splitting
- Maximum entry size limitation (about 50 chars is a sane default value)

Adding Related Terms

You can configure Related Terms to offer users-related query terms that might be relevant depending on their original queries.

About Related Terms

Configure Related Terms and Similar Documents Detection

About Related Terms

Related terms are a list of nouns or adjectives separated by link words, and shared by at least N documents of your corpus. The link words are identified by an internal, language-specific resource file that you cannot edit.

Related terms are flagged at index time as semantic annotations, based on the configuration of the **Related Terms Extractor** semantic processor.

Note: You can also add text directly to the dictionary using the dedicated annotation `relatedTermCustom` when defining annotations (**Kind** or **Name** field).

For related terms to display on the Refinements panel at search time, they must meet the following criteria:

- Not be shared by more than X% of your hits (X=25 by default).
- Be in at least Y hits (Y=3 by default).
- Have a corpus frequency of at least Z (Z=0 by default).

Force or Prevent Related Terms with Allow Lists and Block Lists

- **Allow list:** An indexing-time instruction that ensures whenever the specified expression is detected in the text. It is sent to the dictionary as a possible related term.

Note: This does NOT bypass the selection criteria for displaying related terms in the Refinements panel. The related terms that appear are determined according to their relevance to the search results. You cannot force a certain related term at search time.

- **Block list:** A search-time filter that blocks the specified expression from displaying as a related term in the Refinements panel. There is little to no performance impact, and no need to reindex.

Configure the Detection of Similar Documents Based on Related Terms

Similar documents search means that for each hit, Exalead CloudView generates a new query that retrieves all indexed documents deemed close enough to it.

Depending on the frequency of each related term in your corpus, the number of related terms in your documents, and the conditions specified in the **Similarity** configuration parameters, the prefix handler generates:

- A new dynamic virtual field that specifies the similarity formula.
- A new dynamic sort on this virtual field, to display more similar documents first.

UQL Query

For example, your similar query probably looks like:

```
similar:(264 328 579 628 730 782 806 847 853 871 955 1064 1071 1073 1074 1134 1137 1139 1390 1391 1474 1537 1539 1560 1579 1585)
```

ELLQL Query

This generates the following ELLQL query, which defines the similarity virtual field and assigns a weight to each keyword lookup according to its corpus frequency.

```
#query{nbdocs=934, score.expr="@term.score * @proximity + @b",
  similarity.expr="(#length(keyword) >= 5) * ((score >= 332925) * score / #sqrt(30* #1
/ 0.083231266666667",
  proximity.maxDistance=1000,
  term.score=RANK_TFIDF} (#or (#num{b=53646} (keyword,==,264)
#num{b=70688} (keyword,==,328) #num{b=75574} (keyword,==,579)
#num{b=74506} (keyword,==,628) #num{b=34317} (keyword,==,730)
#num{b=40264} (keyword,==,782) #num{b=107885} (keyword,==,806)
#num{b=143583} (keyword,==,847) #num{b=76695} (keyword,==,853)
#num{b=80417} (keyword,==,871) #num{b=60146} (keyword,==,955)
```

```
#num{b=88194} (keyword,==,1064) #num{b=61715} (keyword,==,1071)
#num{b=30021} (keyword,==,1073) #num{b=46950} (keyword,==,1074)
#num{b=61715} (keyword,==,1134) #num{b=143583} (keyword,==,1137)
#num{b=96514} (keyword,==,1177) #num{b=90061} (keyword,==,1194)
#num{b=51783} (keyword,==,1270) #num{b=130086} (keyword,==,1285)
#num{b=90061} (keyword,==,1362) #num{b=83255} (keyword,==,1390)
#num{b=161950} (keyword,==,1391) #num{b=161950} (keyword,==,1474)
#num{b=43783} (keyword,==,1537) #num{b=92059} (keyword,==,1539)
#num{b=76695} (keyword,==,1560) #num{b=99010} (keyword,==,1579)
#num{b=69832} (keyword,==,1585) )
```

Configure Related Terms and Similar Documents Detection

If you selected the option **Enable related terms** during setup, the related terms feature is already set up. You can yet customize default values or add block lists and allow lists.

If you did not select the option **Enable related terms** during setup, see the procedure below to enable related terms first.

Find Out Which Languages Support Related Terms

1. In the Administration Console, go to **Index > Linguistics > Dictionaries > dictionary_name > Related Terms**.

or, check your `<DATADIR>/config/dictionary.xml` file.

Enable Related Terms

1. In the Administration Console, select **Data Model > Semantic Types > text**.
2. Select the **Extract related terms** check box. A default semantic processor (`RelatedTerms.default`) is added to your analysis pipeline and a facet (`rt_keyword`) is added to your search logic.
3. To allow the display of related terms in the **Refinements** panel of your search application, select **Search > Search Logics > Your_Search_Logic > Facets**.
4. Under **Related terms** (at the bottom), select **Enable**.

Related terms are enabled. See the procedure below to customize default values or add block lists and allow lists.

Configure Related Terms

1. In the Administration Console, select **Search > Search Logics > Your_Search_Logic > Facets**.
2. In the **Related terms** section, configure the following options:

Parameter	Description
Dictionary	Specify the dictionary to use.
Value field indexing RT	Index field in which related terms have been indexed (by default, named <code>keyword</code>).
Block list	Blocks the specified expression from displaying as a related term in the Refinements panel. See Set Up Related Terms Block Lists . You can also set up related terms allow lists. See Set Up Related Terms Allow Lists .
Maximum number of RT	Maximum number of related terms to be computed for a query.
Minimum frequency for a RT	Minimum number of occurrences in the whole index for a term to be possibly selected for synthesis.
Result-set Low-pass filter	Filters out terms occurring more than this threshold in the result set (value comprised between 0 and 1).
Corpus Low-pass filter	Filters out terms occurring more than this threshold in the whole index (value comprised between 0 and 1).

3. Optionally, for big corpuses, you can enhance the quality and performance of Related Terms calculation by tuning two parameters in the Search Logic XML configuration.
 - a. Open the API Console and click **Manage**
 - b. Search for **SetSearchLogicList** and go the `<ns:#RelatedTermsSynthesisConfig>` node and configure its parameters.

Note: For information about these parameters, see "RelatedTermsSynthesisConfig" in the Exalead CloudView XML Configuration Reference Guide.
4. Click **Apply**.

Set Up Related Terms Block Lists

1. In the Administration Console, go to **Index > Data processing > Pipeline name > Semantic Processors**.
2. Under **Block list**, specify your resource file.
 - If you have already created a resource file, click **Browse** to select the resource file. If you have created a resource file using `cvadmin`, type the path to the resource file using the format `resourcemanager://group_name/resource_name`.
 - OR, create a new resource: click **Create new**, specify a name for the list, and click **Accept**. This adds a resource file using the same name as the Administration Console's Resource

Manager, to ensure correct deployment of interdependent resource files in multihost environments.

- To define the contents of the resource file, click **Edit**. This takes you to the Business Console. For more information, see "Add a related terms block list or allow list" in the Exalead CloudView Business Console User's Guide.

3. Click **Apply**.

Set Up Related Terms Allow Lists

1. In the Administration Console, go to **Index > Data processing > Pipeline name > Semantic Processors**.
2. Under **Allow list**, specify your resource file.
 - If you have already created a resource file, click **Browse** to select the resource file. If you have created a resource file using `cvadmin`, type the path to the resource file using the format `resourcemanager://group_name/resource_name`.
 - OR, create a new resource: click **Create new**, specify a name for the list, and click **Accept**. This adds a resource file using the same name as the Administration Console's Resource Manager, to ensure correct deployment of interdependent resource files in multihost environments.
 - To define the contents of the resource file, click **Edit**. This takes you to the Business Console. For more information, see "Add a related terms block list or allow list" in the Exalead CloudView Business Console User's Guide.
3. Click **Apply**.

Enable the Detection of Similar Documents

The similarity of two documents is based on related terms. You must set up related terms before you can use similar documents. See [Enable Related Terms](#).

1. In the Administration Console, go to the **Search Logics > Hit Content** tab.
2. In the **Similarity** section, select **Enable**.
3. Change the configuration parameters if required.

Parameter	Description
Prefix handler name	Specifies the prefix handler that must be entered in the search box before the query.
Min. shared keywords	Does not return documents that do not share at least the specified number of shared keywords with the reference document.

Parameter	Description
Min. keywords per document	Does not return similar documents that do not have at least the specified number of keywords.
Min. similarity threshold	Specifies the minimum similarity score for two documents to be considered similar. Value must be between 0 and 1, 1 meaning exact match.
Language constraint	Forces to detect similar documents in the same language.


- Click **Apply**.
- Start a query in the Mashup UI.

You can view similar documents by clicking the Similar results link (on the lower right).

Microsoft Word - releve_decisions_20070108.doc

Télécharger

Aperçu



Chemin du fichier

/udir2/ /corpus/chess/releve_d

ecisions_CD20070108.pdf

Nom du fichier

releve_decisions_CD20070108.pdf

similar: (264 328 579 628 730 782 8 06 847 853 871 955 1064 1071 107 3 1074 1134 1137 1177 1194 1270 1285 1362 1390 1391 1474 1537 15 39 1560 1579 1585)

Generated query

keyword

264 , 328 , 579 , 628 , 730 , 782 , 8 06 , 847 , 853 , 871 , 955 , 1064 , 1 071 , 1073 , 1074 , 1134 , 1137 , 11 77 , 1194 , 1270 , 1285 , 1362 , 139 0 , 1391 , 1474 , 1537 , 1539 , 1560 , 1579 , 1585

Retrieved RelatedTerm ids

Taille du fichier

41685

Source

fs

Classe du modèle de données

document


Auteur

MD

Extension

pdf

Langue

 Français

Type de document

Document PDF

Dernière modification

2009 > 12 > 23

id: /%2Fudir2%2F %2Fcorpus%2Fchess/releve_decisions_CD20070108.pdf

Similar link

Résultats similaires

Configuring and Using Similarity Measures

The `#attrsimilar` function calculates similarity between a given vector and vectors in the index. For example, you can use it to detect 3D parts with similar shape or size.

`#attrsimilar` is a query node in the index, which returns all the documents matching the similarity query and calculates the similarity measure. As it does not filter search results at all, you must combine it with a `#filter` to return only the documents having a similarity higher than a given threshold value.

Note: Similarity is the inverse of distance and calculated as follows: $\text{similarity} = 1 - \text{distance}$

Important: The standard way to use `#attrsimilar` is inside a query template. See [Defining Query Templates](#).

Configure the Index for Similarity Queries

Use the `#attrsimilar` Function in the Search API

Code Samples to Create Similarity Query Prefix Handlers

Configure the Index for Similarity Queries

This section describes how to index and process signature values to be able to enter similarity queries in the Search API and calculate similarity measures.

Configure the Data Model and the Data Processing

The following procedure explains how to store a signature in an index field represented by the `SIGNATURE_INDEX_FIELD` variable.

Note: If you need to store multiple signatures, use a dynamic field. To do so, follow step 2 and in the field **Advanced options**, select the **Multivalued** and **Store meta names** properties.

1. In the Administration Console, go to **Index > Data Model > Advanced Schema**.
2. Add a `SIGNATURE_INDEX_FIELD` to store signature values.
 - a. Click **Add field**.
 - b. Enter a name, for example, `my_signature_bin` and set the type to **Binary**.
 - c. Set the new field as **RAM-based** for performance reasons.
3. Go to **Index > Data Processing > Analysis pipeline > Document Processors**.

4. Add a **SimilarStringToPart** document processor for part conversion to the pipeline, and in **Input from**, enter the name of the `SIGNATURE_INPUT_META` containing all values of the signature vector, for example, `my_signature_meta`.
This document processor can:
 - Parse signature values and convert them into binary blob ready to use by the index.
 - Delete the meta to create a part with the same name.
5. In the **Mappings** tab, create the mapping between the `SIGNATURE_INPUT_META` and the `SIGNATURE_INDEX_FIELD`.
 - a. Add a mapping source. Give it a name, for example, `my_signature_meta` and set its type to **Part**.
 - b. Add the `SIGNATURE_INDEX_FIELD` as mapping target. For example, target the `my_signature_bin` index field.
6. Click **Apply**.

Test the Configuration

1. Go to the API Console to push a test document.
 - a. In **URI**, enter a document name, for example, `doctest`.
 - b. In **Metas**, add your `SIGNATURE_INPUT_META` in the **Name** column and a list of float separated by spaces in the **Value** column.

For example, Name = `my_signature_meta`, Value = `0.458 -1.68 2`

- c. Click **Push document**.

The result must be **"The document was successfully pushed."**

2. Open the Search API and test the `#attrsimilar` function with the following syntax:

```
http://HOSTNAME:BASEPORT+10/search-api/search?eq=
%23attrsimilar{name=SIGNATURE_SCORE_OUTPUT}(SIGNATURE_INDEX_FIELD,
SIGNATURE_FLOAT_VALUES)&hit_meta.SIGNATURE_OUTPUT_META_NAME.expr=@SIGNATURE
```

In this example:

- The `SIGNATURE_FLOAT_VALUES` variable is set with the float values `0 2 3` (you do not need to surround these values by double quotes `"`).
- The `SIGNATURE_SCORE_OUTPUT.value` returns a numerical value, which is the similarity score calculated by the similarity function.

```
<metas>
  <Meta name="url">
    <MetaString name="value">doctest</MetaString>
  </Meta>
```



```
<Meta name="SIGNATURE_OUTPUT_META_NAME">
  <MetaString name="value">0.4833253026008606</MetaString>
</Meta>
</metas>
```

For more details about the use of #attrsimilar in the Search API, see the following section.

Use the #attrsimilar Function in the Search API

This section describes the use of the #attrsimilar function in the Search API, after the `http://HOSTNAME:BASEPORT+10/search-api/search?eq=%23` part of the URL. Do not forget to remove the # before attrsimilar in the URL.

#attrsimilar Syntax

You can call the #attrsimilar function in a query using the following Search API syntax:

```
#attrsimilar{name=SIGNATURE_SCORE_OUTPUT}(SIGNATURE_INDEX_FIELD,SIGNATURE_FLOAT_VALUES)
```

Where:

- The SIGNATURE_FLOAT_VALUES (for example, 0 2 3) is compared with all the signatures stored in the SIGNATURE_INDEX_FIELD.
- SIGNATURE_SCORE_OUTPUT is the name of the ranking key that stores the similarity measure.

This value can be:

- Displayed in all hit metas with this meta value:
`&hit_meta.SIGNATURE_OUTPUT_META_NAME.expr=@SIGNATURE_SCORE_OUTPUT.value.`
- Used as a sorting key to display best values first:
`&s.SIGNATURE_SORT_KEY_NAME.expr=@SIGNATURE_SCORE_OUTPUT.value&s=desc(SIGNATURE_SCORE_OUTPUT)`

To use #attrsimilar with a dynamic field containing multiple signature values, use the following syntax:

```
#attrsimilar{...}(MULTICONTEXT_INDEX_FIELD, "context_signature_1", SIGNATURE_FLOAT_VALUES)
```

Where:

- the MULTICONTEXT_INDEX_FIELD variable corresponds to the dynamic field name containing the signatures.
- context_signature_1 is the name of a context in this dynamic field.

Similarity Functions

The similarity measure varies depending on the function used to compare vectors two by two.

Important: With most similarity functions, it is not possible to compare two vectors that do not have the same size. In that case, indexed documents for which the signature vector does not have the same size than the query vector, are not returned to the `#attrsimilar` node.

To choose a function, use the following syntax:

```
#attrsimilar{name=SIGNATURE_SCORE_OUTPUT,
  function=euclidian_normed}(SIGNATURE_INDEX_FIELD,SIGNATURE_FLOAT_VALUES)
```

Similarity is calculated as follows: $\text{similarity} = 1 - \text{distance}$. For all `_normed` functions, we can summarize the calculation as:

```
similarity = 1 <--> close; similarity = 0 <--> far
dist = 1 <--> far; dist = 0 <--> close
```

For non-normed similarity functions (for example Manhattan, Euclidian, etc.), the calculation is identical but the distance milestones change from `[0;1]` to `[0,Infinity]` and similarity is delimited by `[-Infinity;1]`.

The `cosine` similarity function is the exception, with milestones `-1` (unsimilar) and `1` (similar). The `angular` similarity function allows you to bring `cosine` similarity between 0 and 1, and be consistent with other similarity functions.

Function	Use
<code>manhattan</code> (default function)	For L1-normalized vectors. Formula: $\text{sim} = 1 - (\text{Sum}\{\text{abs}(x1[i] - x2[i])\}/2)$ The similarity is between 0 and 1.
<code>manhattan_normed</code>	Same as <code>manhattan</code> with L1-normalized vectors first. Formula: $\text{sim} = 1 - (\text{Sum}\{\text{abs}(x1[i]/\text{NormL1}(x1) - x2[i]/\text{NormL1}(x2))\}/2)$, $\text{NormL1}(x) = \text{sum}_i\{\text{abs}(x[i])\}$ The similarity is between 0 and 1.
<code>manhattan_dist</code>	For any vectors. Formula: $\text{dist} = \text{Sum}\{\text{abs}(x1[i] - x2[i])\}$ The distance is between 0 and infinity.
<code>multi_manhattan_normed</code>	Compares 2 sets of vectors having the same dimension. For example, 2 vectors of 8 floats and 3 vectors of 8 floats, using the exclusive min between all <code>MANHATTAN_NORMED</code> distances. The similarity is between 0 and 1.
<code>euclidian</code>	For L2-Normalized vectors.

Function	Use
	<p>Formula: $\text{sim} = 1 - \sqrt{(\text{Sum_i}\{(x1[i] - x2[i])^2\})/2}$</p> <p>The similarity is between 0 and 1.</p>
euclidian_normed	<p>Same as euclidian with L2-normalized vectors first.</p> <p>Formula: $\text{sim} = \text{NormL2}(x) = \sqrt{\text{sum_i}\{x[i]^2\}}$</p> <p>The similarity is between 0 and 1.</p>
euclidian_dist	<p>For any vectors.</p> <p>Formula: $\text{dist} = \sqrt{\text{Sum_i}\{(x1[i] - x2[i])^2\}}$</p> <p>The distance is between 0 and infinity.</p>
cosine	<p>Angle between 2 vectors.</p> <p>Formula: $\text{COSINE} = (\text{Sum}\{x1[i] * x2[i] / (\text{NormL2}(x1) * \text{NormL2}(x2))\})$</p> <p>Similarity is between -1 and 1, where -1 is unsimilar and 1 is similar.</p>
angular	<p>Formula: $\arccos(\text{COSINE}) / \text{PI}$</p> <p>The similarity is between 0 and 1.</p>
dice	<p>For binary bits strings. It computes the intersection between bits to 1 of 2 sequences.</p> <p>Formula: $D = (2 * X \text{ inter } Y / (X + Y))$</p> <p>The similarity is between 0 and 1.</p>
jaccard	<p>For binary bits strings. It computes the intersection between bits to 1 of 2 sequences.</p> <p>Formula: $J = (2 * X \text{ inter } Y / (X + Y - X \text{ inter } Y))$</p> <p>The similarity is between 0 and 1.</p> <p>Note: jaccard is sometimes called TANIMOTO</p>
hamming	<p>For binary bits strings. It computes the number of ones in an XOR of bits sequence.</p> <p>Formula: $H = 1 - (XOR(X, Y) / \text{lenBit}(X))$</p> <p>The distance is between 0 and length(vectors).</p>

Combine #attrsimilar with a Filter

To combine `#attrsimilar` with a filter, use the following syntax:

```
#filter("@SIGNATURE_SCORE_OUTPUT.value>SIGNATURE_SCORE_THRESHOLD",
#attrsimilar{name=SIGNATURE_SCORE_OUTPUT,function=euclidian_normed}
(SIGNATURE_INDEX_FIELD,SIGNATURE_FLOAT_VALUES))
```

This syntax allows you to keep only the documents with a similarity measure higher than (>) the `SIGNATURE_SCORE_THRESHOLD`. For example, you could use a float value like `0.55`.

You can also combine several signature computations in one `#filter` expression. For example:

```
#filter("@SIGNATURE_1_SCORE_OUTPUT.value>SIGNATURE_1_SCORE_THRESHOLD&&
@SIGNATURE_2_SCORE_OUTPUT.value>SIGNATURE_2_SCORE_THRESHOLD",
#and(#attrsimilar{name=SIGNATURE_1_SCORE_OUTPUT,function=euclidian_normed}
(SIGNATURE_1_INDEX_FIELD,SIGNATURE_1_FLOAT_VALUES),
#attrsimilar{name=SIGNATURE_2_SCORE_OUTPUT,function=euclidian_normed}
(SIGNATURE_2_INDEX_FIELD,SIGNATURE_2_FLOAT_VALUES))
```

Code Samples to Create Similarity Query Prefix Handlers

The standard use of `#attrsimilar` is inside a query template using the ELLQL language. For advanced Exalead CloudView users who want to manage similarity queries in UQL, you can adapt the following code samples.

To create your similarity query prefix handler, adapt the following code samples to your use case and package your custom prefix handler as a `CVPlugin`. For more information, see in the Exalead CloudView Programmer's Guide.

Code for simple `attrsimilar` prefix handler (`SimpleAttrSimilarPrefixHandler.java`)

```
package com.exalead.example.search;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import org.apache.log4j.Logger;
import com.exalead.mercury.component.CVComponent;
import com.exalead.mercury.component.config.CVComponentConfigClass;
import com.exalead.search.query.QueryContext;
import com.exalead.search.query.QueryProcessingException;
import com.exalead.search.query.node.AttrSimilar;
import com.exalead.search.query.node.If;
import com.exalead.search.query.node.IndexOptions;
import com.exalead.search.query.node.Node;
import com.exalead.search.query.node.NodeVisitor;
import com.exalead.search.query.node.PrefixNode;
import com.exalead.search.query.node.UserQueryChunk;
import com.exalead.search.query.prefix.CustomPrefixHandler;
```

```

import com.exalead.search.query.util.LongOrDouble;
@CVComponentConfigClass(configClass=SimpleAttrSimilarPrefixHandlerConfig.class)
public class SimpleAttrSimilarPrefixHandler extends CustomPrefixHandler implements
    CVComponent {
    private static final Logger log = Logger.getLogger(SimpleAttrSimilarPrefixHandler.class);
    private final SimpleAttrSimilarPrefixHandlerConfig config;
    public SimpleAttrSimilarPrefixHandler(SimpleAttrSimilarPrefixHandlerConfig config) {
        super(config);
        this.config = config;
    }
    @Override
    public Node handlePrefix(Phase phase, PrefixNode node,
        NodeVisitor parentVisitor, QueryContext queryContext)
        throws QueryProcessingException {
        if(phase == Phase.POST_PARSE){
            if (node.content instanceof UserQueryChunk) {
                UserQueryChunk uqc = (UserQueryChunk) node.content;
                String[] tokens = uqc.value.split(",");
                IndexOptions options = new IndexOptions();
                LongOrDouble filterValue = null;

                String signatureField = config.getIndexField();
                String signatureContext = null;
                String function = config.getDistance();

                //let's parse node options to override config ones
                if(uqc.indexOptions != null && uqc.indexOptions.getRawOptions() != null){
                    for(Map.Entry<String, String> entry : uqc.indexOptions.getRawOptions().entrySet()){
                        if("filter_value=".equals(entry.getKey())){
                            filterValue = new LongOrDouble(Double.parseDouble(entry.getValue()));
                        } else if("index_field=".equals(entry.getKey())){
                            signatureField = entry.getValue();
                        } else if("function=".equals(entry.getKey())){
                            function = entry.getValue();
                        } else {
                            options.addRawOptions(entry.getKey(), entry.getValue());
                        }
                    }
                }

                if(filterValue == null && config.getFilterValue() != null){
                    filterValue = new LongOrDouble(config.getFilterValue());
                }
                queryContext.query.hitOrder.clone();
                options.addRawOptions("function=", function);

                String vfName = tokens[0];
                List<LongOrDouble> signature = parseSignature(tokens[1]);
            }
        }
    }

```

```

        if(signatureField != null && signatureField.contains("@")){
            String[] signatureFieldTokens = signatureField.split("@");
            signatureField = signatureFieldTokens[1];
            signatureContext = signatureFieldTokens[0];
        }

        return createAttrSimilarNode(vfName, options, signatureField, signatureContext,
            signature, filterValue);
    }
}
return node;
}

private Node createAttrSimilarNode(String vfName, IndexOptions options,
    String signatureField, String signatureContext, List<LongOrDouble> signature,
    LongOrDouble filterValue) throws QueryProcessingException{
    Node res = null;
    options.addRowOptions("name=", vfName);
    if(signatureField == null){
        log.error("Missing signature field config or option");
        throw new QueryProcessingException("Missing signature field config or option");
    }
    //the #attrsimilar node
    res = new AttrSimilar( signatureField, signatureContext, signature, null, null, null);

    if(filterValue != null){
        //here we create the surrounding #filter node
        String filter = "@"+vfName+".value>="+filterValue.toString();
        IndexOptions opts = new IndexOptions();
        res = new If(res, filter, opts);
    }
    return res;
}

/**
 *
 * @param signature a space-separated list of double
 * @return The List<LongOrDouble> containing the signature
 */
private List<LongOrDouble> parseSignature(String signature){
    String[] tokens = signature.trim().split(" ");
    List<LongOrDouble> res = new ArrayList<LongOrDouble>(tokens.length);
    for(int i = 0; i<tokens.length; i++){
        res.add(new LongOrDouble(Double.parseDouble(tokens[i])));
    }
    return res;
}
}

```

Code for simple attrsimilar prefix handler configuration

(SimpleAttrSimilarPrefixHandlerConfig.java)

```

package com.exalead.example.search;
import com.exalead.config.bean.IsHidden;
import com.exalead.config.bean.IsMandatory;
import com.exalead.config.bean.PropertyDescription;
import com.exalead.mercury.component.config.CVComponentConfig;
import com.exalead.search.query.util.LongOrDouble;
public class SimpleAttrSimilarPrefixHandlerConfig implements
    CVComponentConfig {
    public SimpleAttrSimilarPrefixHandlerConfig() {
    }
    private String indexField;
    private String distance = "manhattan_normed";
    private LongOrDouble filterValue;

    @IsMandatory(false)
    @PropertyDescription("The binary index field that contains the signatures."
        + "If it's a dynamic field (multi valued and storing meta names) "
        + "use the following syntax: signatureName@indexFieldName. "
        + "This value can be overridden in query option \"index_field\".")
    public void setIndexField(String indexField) {
        this.indexField = indexField;
    }

    public String getIndexField() {
        return indexField;
    }

    @PropertyDescription("The distance function to use. "
        + "Some possible values: manhattan, manhattan_normed, euclidian, "
        + "euclidian_normed, cosine. "
        + "Normed versions of the distance must be used when the signatures "
        + "in the index have not been normed before indexing. "
        + "This value can be overridden in query option \"function\".")
    public void setDistance(String distance) {
        this.distance = distance;
    }

    public String getDistance() {
        return distance;
    }

    @IsMandatory(false)
    @PropertyDescription("The minimum similarity score that a hit must have "
        + "to match the query. ")

```

```

+ "This value will generally between 0 and 1. "
+ "It can be overridden in query option \"filter_value\". "
+ "If empty, there will be no filtering based on the score.")
public void setFilterValue(Double filterValue) {
    this.filterValue = new LongOrDouble(filterValue);
}

public Double getFilterValue() {
    if(filterValue == null){
        return null;
    }
    return filterValue.getDouble();
}

@IsHidden
public LongOrDouble getLongOrDoubleFilterValue() {
    return filterValue;
}
}

```

Configuring Geographic Search

This section explains how to create geographic points in Exalead CloudView

Note: To keep high performance, Exalead CloudView distance calculation uses an efficient but approximative algorithm, which may introduce precision errors up to 0.2%.

Note: For information about geographic facets, see [Geographic Facets](#).

About Geographic Points

Create a Geographic Point

Search a Geographic Point

Calculate Distances in Virtual Fields

Use Geolocation Based on Place Detection

About Geographic Points

Exalead CloudView supports the following geographic coordinates:

- GPS fields, also called WGS84, that are a latitude and a longitude expressed in decimal.
- XY fields, also called Meter, that are two integers used on a map.

GPS Points

GPS points are indexed in decimal format, to an accuracy of 6 decimal places (which represents on average an accuracy of 10cm on Earth). A meta consists of two double values separated by commas.

Example: 37.818667,-122.478383 is a valid meta.

XY Points

XY points consist of two integer values, separated by commas.

Example: 125, 8215 is a valid meta.

No unit is defined, so you can consider the unit as meters, miles, or whatever unit you need.

Create a Geographic Point

Geographic search is available through the **Point** field type.

For an example of creating a geographic point and using it to perform a geographic search in Mashup UI, see the "Restrict the search results to a Geographical Area" in the Exalead CloudView Mashup Builder User's Guide.

Create a Point Using the Data Model

1. In the Administration Console, go to **Index > Data model > Classes**.
2. Click **Add property**.
 - Give a name to the property.
 - For **Data type**, select either **GPS point** or **XY point**.
 - Click **Accept**.
3. Optionally, select **Use separate metas for each coordinate** if latitude/ X and Longitude/ Y come from two different metas in your data source. Exalead CloudView concatenates the two coordinates to store them in a single index field.
4. Click **Apply**.

The Data model configures the Index Schema and the Mapping automatically.

Create a Point Field Using the Index Schema

To enable the feature, you need to set the parameter `indexExact` to `false` in `<DATADIR>/config/IndexSchemaList.xml`

1. In the Administration Console, go to **Index > Data model > Advanced Schema**.

2. Click **Add field**.
 - Give a name to the field.
 - Select the **Point** type.
3. For the **Geographical coordinates type**, select **WGS84** (GPS) or **Meter** (XY).
4. In **Index > Data processing > Pipeline name > Mappings**, add a new mapping and a new target.
5. Under **Indexing options**, select **Index exact form**. You can clear all the other indexing options.
6. Click **Apply**.

Search a Geographic Point

You can search a geographical point:

- in ELLQL, in the search-api.
- in UQL, using the `geo:` prefix handler.

You can search for a point by radius or within a polygon.

Search within a Radius (ELLQL)

1. To get all the points around a certain point:

```
#distance(point_field, lat/x, long/y, distanceInMeters)
```

.

2. To also retrieve the distance of each point from the center:

```
#distance{name=pointQuery}(point_field, lat/x, long/y, distanceInMeters)
```

3. Then use `@pointQuery.distance` in a virtual field.

Search within a Polygon (ELLQL)

1. To get all the points within a polygon:

```
#within(point_field, (X1, Y1; X2, Y2; X3, Y3; ...))
```

where X_n Y_n are corners of your polygon in the same coordinate format as your points.

For example:

```
#within(gps, (0.0, 0.0; 1.0,0.0; 1.0,1.0; 0.0, 1.0))
```

Returns all the points within the square (0.0, 0.0) - (1.0, 1.0).

```
#within(gps, [(0.0, 0.0; 2.0, 0.0; 2.0, 2.0; 0.0, 2.0) (1.0, 1.0; 3.0, 1.0; 3.0, 3.0; 1.0, 3.0)])
```

Returns all the points within the square (0.0, 0.0) - (2.0, 2.0) OR (1.0, 1.0) - (3.0, 3.0) but not in both.

Search with a Radius or Polygon (UQL)

1. In UQL, you can use the following operands with a Geographic prefix handler (`geo:`).

Operand	Example
<code>within(lat1,lng1; lat2, lng2; lat3, lng3;)</code> searches all the locations within the specified polygon	<code>geo:WITHIN(48.33, 2.51; 45.65, 1.34; 24.54, -4.54; 12.34, -6.65)</code>
<code>distance(lat, lng, distance_in_meters)</code> retrieves all the documents where the geographic field is located <code>distance_in_meters</code> from the specified <code>lat, lng</code>	<code>geo:DISTANCE(48.33, 2.51, 250000)</code>

Use UQL with ELLQL for Geographical Search

1. You can mix UQL queries with ELLQL, for example:

```
eq=#and(#distance(gps, 3.4, 4.3, 100) #uql("query in uql OR file_size<100"))
```

Calculate Distances in Virtual Fields

You can calculate distance using virtual fields that are based on retrievable index fields. These distances can then display as hit meta.

- When `#distance` is used with a name, `@thename.distance` returns the distance to the center. It is also possible to use `#dist(pointField, lat/x, lng/y)` to return the distance of the point to (lat, lng) or (x, y) depending on the coordinate format.
- When GPS points are used the distance is in meters.

You can also use `#lat(point)` to get the latitude or the X component of a point, and `#lng(point)` to get the longitude or the Y component of a point.

Use Geolocation Based on Place Detection

Exalead CloudView is delivered with two resource files to enable geolocation based on place detection in your search results. They include a predefined list of cities with gps coordinates.

This section describes the procedures to follow to use geolocation based on place detection.

Step 1 - Add an Ontology Matcher and Annotation Manager to the Analysis Pipeline

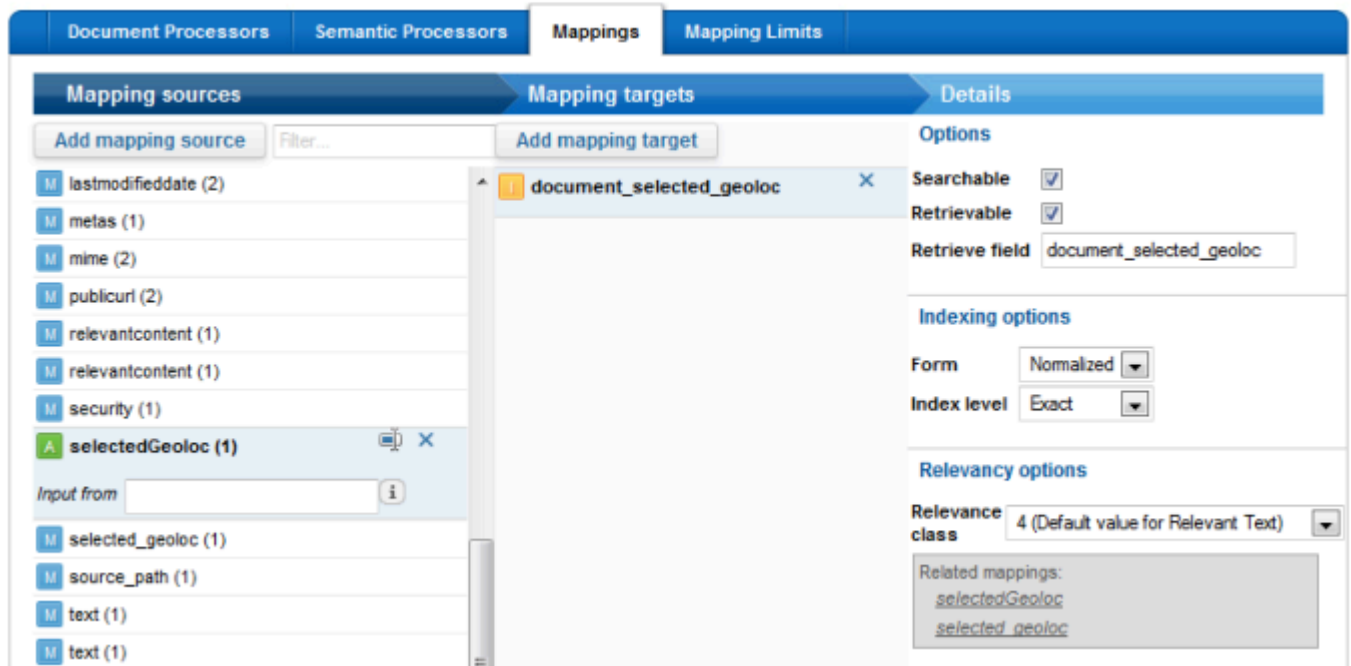
1. In the Administration Console, go to **Index > Data processing > Pipeline name > Semantic Processors**.
2. Drag an **Ontology Matcher** processor to the list of current processors.
3. Click the **Ontology Matcher** link to expand the view, and in **Resource directory**, enter:
`resource://namedentities/geoloc/geoloc.bin`
4. Add an **Annotation Manager** processor after the Ontology Matcher.
5. Click the **Annotation Manager** link to expand the view, and in **Resource file**, enter:
`resource://namedentities/geoloc/selectmostfrequentgeoloc.xml` (this resource uses the `geoloc.bin` resource defined above).

Step 2 - Create a New GPS Point Index Field

1. Go to **Index > Data model > Classes**
2. Click **Add property** and for:
 - a. **Name**, enter `selected_geoloc`.
 - b. **Data type**, select **GPS point**.
 - c. **Field type**, select **Dedicated field only**.
3. Expand **Other advanced options** and select **Multivalued**.

Step 3 - Add a Mapping Source and a Mapping Target

1. Go to **Index > Data processing > Pipeline name > Mappings**
2. Click **Add mapping source** and for:
 - a. **Name**, enter the annotation name `selectedGeoloc` (this annotation is delivered by the annotation manager with the `documentAnnotation` attribute)
 - b. **Type**, select **Annotation**.
3. Click **Add mapping target** and for:
 - a. **Type**, select **index field**
 - b. **Name**, enter `document_selected_geoloc`



4. Click **Apply**
5. Import documents with your connectors.

Step 4 - Add the Google Maps Widget to Your Mashup UI Application

1. In Mashup Builder, go to the **/search** page, and select the **Design** view
2. In the **Widgets** panel, select **Visualizations > Maps**, and drag the **Google Maps** widget on the page.
3. Click the **Google Maps** widget header. The widget properties panel opens at the bottom of the screen.
4. In the **Based on entries** tab, edit **location (2)** with the following MEL expression:


```
${entry.metas["selected_geoloc"]}
```
5. Click **Apply**.

You can now enter queries in your Mashup UI application and see all the locations detected in the search results on the Google Maps widget.

Adding a Query Cache

You can improve search performance by enabling the query cache, which resides on the Search API command of the Exalead CloudView search server.

The query cache contains a list of hits and facets associated with a specific query, available to users who enter the same query.

To count as an exact match, the query must:

- Have the same $q=$ (UQL) or $e_{q=}$ (ELLQL) value as the cached query.
- Have the same values for all other parameters passed to the Search API.

About Query Cache

Create and Manage a Query Cache

About Query Cache

A query cache can be either:

- Consistent: clears the existing cache every time the index is updated, which ensures up-to-date results but impacts performance for frequently changing indexes.
- Time-based: keeps a query in cache for N seconds, this generates a new cache to create a continuous update of the cache at the risk of the occasional stale result.

Warm-Up Queries

Each query cache also includes the option to define warm-up queries for popular searches, which are automatically loaded into cache:

- When the search server starts.
- After each index update (consistent query caches only).
- Every N seconds (time-based query caches only).

Sample Use Case

Imagine you have a search application that displays a sales dashboard for a chain of retail stores. Executives frequently want an overview of financial trends that includes all the stores in their search results. However, individual store managers want to see the latest sales figures for their store only.

In such a case, you could create two caches:

- One to store user queries in a consistent cache.
- Another to store warm-up queries in a time-based cache.

Create and Manage a Query Cache

Create a Query Cache

You can add multiple query caches to the same command.

1. In the Administration Console, go to **Search > Search API**.

2. Under **Commands**, select **/search-api > Search cache**.
3. Under **Search cache**, click **Add search cache** and create one of the following:
 - **Consistent**: creates a new cache every time the index is updated.
 - **Time-based**: keeps a query in cache for N seconds.
4. (Optional) Under **warm-up queries**, define one or more queries to cache automatically.
 - Enter the query and any parameters as they appear in the search URL.
 - For example, to search for "new york city" while restricting the search to the s10 search logic, enter: `q=new%20york%20city&logic=s10`
 - For details on available parameters, see [Appendix - Search API Parameters](#).
 - (Optional) Modify the other configuration options from their default values.
5. Click **Apply**.

Store Warm-Up Queries Only in a Cache

1. In the search cache settings for your search-api command, clear the **cache queries** options.

Know if Your Query Comes from Cache

This procedure assumes that you have indexed documents.

1. In the browser, open the search-api service at BASEPORT +10 with your query or #all.
For example, `http://localhost:10010/search-api/search?q=%23all`
2. The search results `<Stats>` node returns:
 - When not cached: `<Stats status="ok" queueTime="0" ... />`
 - When cached: `<Stats status="from_cache" cache="CACHE_NAME" />`

Configuring Search Results

The chapter describes how to configure what is displayed in the search results.

This goes from the content of the search results themselves that we call **hit content**, the facets that can be added to refine search results, and the ranking and sorting of search results.

[Defining Search Results Content](#)

[Creating Facets to Refine Search Results](#)

[Calculating Results On-The-Fly with Virtual Fields](#)

[Specifying a Timezone for Date Time Metas](#)

[Ranking and Sorting Search Results](#)

[Collapsing/ Grouping Search Results](#)

[Setting the Limits of Search Results](#)

Defining Search Results Content

A search result often called hit corresponds to a document in the index that contains a match to the user query.

[Configure the Search Result Summary](#)

[Configure Value Selection for Metas](#)

[Configuring the Highlighting of Search Terms](#)

Configure the Search Result Summary

A search result can contain a summary, which is a small number of selected sentences, centered around the query terms. You can typically configure how long it is and whether to highlight search terms in its content.

Note: You can apply a summary on any meta, but it mostly applies to the `text` meta.

1. In the Administration Console, go to **Search > Search Logics > SearchLogic > Hit content**.
2. Click a meta to display its settings.
3. In the meta settings, click **Customize**.
4. Select the **Summary** check box.
5. Expand **Operations**, and expand **Summary**.

Configure the summary as required. For details on summary options, see the tooltips.

6. Click **Apply**.

Configure Value Selection for Metas

You can configure the number of values that are displayed for metas in hit content. This is typically useful to restrict the number of values retrieved from multivalued fields. Especially when you do not want to clutter hit content with too many values for a given meta.

Set the Value Selection

In the Administration Console, when you go to **Search > Search Logics > SearchLogic > Hit content** and select a meta, you can add the **Value display selection** operation. This operation has two required settings:

- **Min. values** – minimum number of values to display for the meta in the hit content. If set to 0, no value is displayed if there are no matching values. If set to more than the number of matching values, some nonmatching values are displayed in addition to the matching values.
- **Max. values** – maximum number of values to display for the meta in the hit content. It selects the n first values. For examples, if you set it to 10 and the query matches more values, only the first 10 values are displayed.

Note: It is a stand-alone feature but you can associate it with the highlight or the summary operations.

Value Selection Use Case

We consider a corpus of cooking recipes where you have an `ingredients` meta, which comes from a multivalued index field.

The problem is that this meta displays by default all the ingredients found in the documents matching your queries, and not only the relevant ingredients. You may thus get extra-long hits with metas containing hundreds of worthless values.

If we search for `"curry AND paprika AND chicken"`, what we expect is to see these 3 ingredients, as values of the `ingredients` meta.

If we want to display at least these 3 ingredients, and then possibly a few other ingredients retrieved randomly, we can for example, define a Value selection on the `ingredients` meta, with **Min. values** to 5 and **Max. values** to 5. The ingredients meta displays: curry, paprika and chicken and 2 other ingredient values in the order they are retrieved from the document. For example, your meta may display as follows: `ingredients: curry, cardamon, paprika, coriander, chicken`

If we want to display matching values only, we must set **Min. values** to 0. We get: ingredients: curry, paprika, chicken in the order they are retrieved from the document.

If we want to display 3 matching values, and at least 2 values (whether matching or not), we set **Min. values** to 2 and **Max. values** to 3. We get:

- ingredients: cardamom, salt – if the document contains ingredient values cardamom, salt, pepper; since we set it to keep a minimum of 2 values whether matching or not.
- ingredients: cardamom, chicken – if the document contains ingredient values cardamom, salt, pepper, chicken; since we set it to keep a minimum of 2 values whether matching or not.
- ingredients: chicken, paprika – if the document contains ingredient values cardamom, chicken, paprika, salt, pepper, since we keep in priority matching values.
- ingredients: chicken, paprika, curry – if the document contains ingredient values cardamom, chicken, paprika, curry, salt, pepper, since we can keep a maximum of 3 matching values.
- ingredients: chicken, green curry, red curry – if the document contains ingredient values cardamom, chicken, green curry, red curry, paprika, salt, pepper, since we can keep a maximum of 3 matching values and the order in which values are retrieved is taken into account.

Configuring the Highlighting of Search Terms

A common search application requirement is to display the results with the matching terms highlighted, in bold or in a different color.

About Highlighting

Setting Up Hit Highlighting

About Highlighting

You can configure highlighting on any meta, but it mostly applies to the `text` and `title` metas, as they allow users to quickly identify whether the hit is interesting or not.

Highlighting Behavior with Prefix Handlers

By default when searching with a prefix handler, the matching terms are only highlighted in the hit meta of the same name.

Using the `title: Prefix Handler`, Matches are only highlighted in the Title Text.

The screenshot shows the Exalead search engine interface. At the top, the Exalead logo is on the left, and a search bar contains the query 'title:connectors'. Below the search bar, a blue bar indicates 'Results › 1-10 of 10' and a 'Sort by' dropdown. The first search result is titled 'Connectors' and includes a summary of updates for V6R2014, mentioning HTTP crawler improvements, Digest and NTLM proxy authentication, and a new CSV connector. The second result is titled 'Controlling Connectors' and provides a guide on managing connectors, including actions like 'Clear documents' and 'Full scan'. Both results show a table with columns for 'File path' and 'File size'.

You can, however, associate extra prefix handlers and facets for highlighting matching terms.

This is the case of the `title` hit meta, which is pre-configured with additional prefix handlers. This means that matching terms are highlighted in a document's title when:

- Searching with the `title:` prefix handler. This is the default behavior described above.
- Searching with one of these prefix handlers: `text:`, `soundlike:`, or `spellslike:`. These are specified in the `title` hit meta setup as extra prefix handlers.

Using the `text:` Prefix Handler, which is an extra Prefix Handler for the `title` Hit Meta, Matches are highlighted in the Title, as well as in the Summary Text.

EXALEAD

Results › 1-10 of 312 Sort by

About Connectors

CloudView **Connectors** Guide : **About Connectors** **About Connectors** This section introduces CloudView **connectors** and describes common procedures. © 2010-2013 Dassault Systèmes exalead.documentation@3ds.com

File path: /data/.../cloudview-dev_59136-linux-x64/docs/html/connectors.03.01.html File size: 3486

XML Connectors

CloudView **Connectors** Guide : **XML Connectors** **XML Connectors** This section describes the functionality and configuration of the **XML Connectors**. © 2010-2013 Dassault Systèmes exalead.documentation@3ds.com

File path: /data/.../cloudview-d File size: 3486

Deactivating the Highlighting for a Subquery Node

In specific use cases, you may want to disable the highlighting and the summary for a specific node of your query only. In other words, you want the highlighting/summary to be activated for the overall query and deactivated for a subpart of it.

This can be specified in both UQL and ELLQL languages by appending the `{hl=0}` parameter to the node that must not be highlighted. For example, to highlight `New York` but not `city` in the UQL query `New York city`, we can enter the following query: `New York city{hl=0}`

Note: You could also change the query template in **Search > Search Logics > Your search logic > Query Template**, to exclude a subquery node globally by appending the `{hl=0}` parameter to `#query`

Setting Up Hit Highlighting

This section describes how to highlight matching search terms in result hits.

Display and Configure a Hit Highlight

1. In the Administration Console, go to **Search > Search Logics > yourSearchLogic**.
2. On the **Hit Content** tab, click a meta to display its settings.
3. In the meta settings, click **Customize**.
4. Select the **Highlight** check box.
5. Expand **Operations**, and expand **Highlight**.

Configure the highlight as required. For details on highlight options, see the tooltips.

6. Click **Apply**.

Specify Extra Prefix Handlers or Facets for Highlighting

1. In the Administration Console, go to **Search > Search Logic > yourSearchLogic**.
2. On the **Hit Content** tab, click a meta to display its settings.
3. In the meta settings, click **Customize**.
4. Expand **Operations**, click **Add operation**, and then select **Highlight**.
 - a. For **Extra prefix handlers for highlight**, enter the names of the additional prefix handlers, separated by commas (the list of prefix handlers is on the **Query Language** tab).
 - b. For **Facet ids for highlight**, enter the facet names, separated by commas.
5. Click **Apply**.

When you search using these prefix handlers or facets, matching search terms in this hit meta are highlighted.

Creating Facets to Refine Search Results

Facets allow your users to filter their search results.

About facets

Create Facets

Numerical Range facets

Date Facets

Configure Date Facets

Multidimension Facets

Geographic Facets

Create Value Facets for Nonhierarchical Metas

Create Aggregations for Facets

Exclusive vs. Disjunctive Refinements

About facets

In your Mashup UI applications you typically use facets through the Refinements panel. By refining your query on a specific facet, you narrow the number of hits displayed in the search results.

Facets are also displayed in charts and table widgets, where you can refine the view by selecting values. For example, in a chart displaying a document count by month, you can focus on the values of a specific month.

Index-Time and Search-Time Facets

Exalead CloudView has 2 notions of facets:

- Index-time facets, stored as hierarchical categories in the `category` field of the index. These index-time facets are often referred to as category facets. They are typically used for alphanumeric metas. After creating index-time facets, you must reindex to store the resulting categories in the `category` field. In turn, these hierarchical categories display as facets in your search application.
- Search-time facets, also known as virtual facets. They are based on index fields stored in RAM, and are typically used for dates and numbers. A big advantage of search-time facets is they are available as soon as you create them; you do not need to reindex.

Which Facet Type Should I Use?

The following table explains which type of facet to create, depending on what you want to display in your application. In each section is a cross-reference for learning more about that particular facet type.

Facet Types

To display the following	Use this facet type
Dates	<p>Choose from the following Date facets:</p> <ul style="list-style-type: none"> • Date: provides fine-grain control on date sorting. • Dynamic dates: Automatically adjusts date granularity, depending on the maximum number of categories specified, and the date range of the search results.

To display the following	Use this facet type
Multiple facets, typically to display in tables, charts, or pivot tables	<p>Choose from the following Multidimension facets:</p> <ul style="list-style-type: none"> • Multi-dimension: Based on N single-dimension facets, and generates a grid, with one facet per dimension. Each cell contains the number of documents and aggregation values for each of the N single-dimension facets. Displays only one level at time, which means better facet performance compared to Hierarchical2D. • Hierarchical2D: Based on two single-dimension facets for each axis. Displays all levels in the hierarchy at once. Use for multiple timelines.
Ranged values, such as prices (\$0-100, \$101-200...)	<p>Numerical ranges are virtual numerical facets used to organize search results in ranges. Specify how to create these ranges:</p> <ul style="list-style-type: none"> • automatically: to specify the number of ranges only, and leave Exalead CloudView define ranges based on search results automatically. • that are the same size: to define a fixed size for all ranges. For example, a range of 100 creates the ranges 0-100, 101-200. • manually: to define everything, that is, the number of ranges, their max and min values, and the range title to display in the Refinements panel.
Geographic data, either rasters or vectors	<p>Choose from the following Geographic facets:</p> <ul style="list-style-type: none"> • Auto-tile geographic: Use for rasters. It creates geographic facets based on a bounding box. Must be based on a XY or GPS point Data Model property. • Geographic: Use for vectors. It creates geographic facets based on disks or polygons. Must be based on a XY or GPS point Data Model property.
Alphanumeric metas with hierarchical values	<p>Category: Use this type of facet if you want to display hierarchical values, or want to define meta mappings to a category manually, instead of using Data Model properties.</p> <p>This type of facet resides in the index, under the <code>category</code> field in the format. It is the only facet type that requires reindexing to be available in your application.</p>

To display the following	Use this facet type
Alphanumeric metas with nonhierarchical values, such as a list of countries, or file formats	<p>Value: Use this instead of a Category facet. It is saved under the <code>value</code> field, which is optimized for faster faceting.</p> <p>Only use for nonhierarchical metas. Create this facet on a <code>value</code> type index field.</p>

Concept of Category Facet

All facet types output as a tree made up of facet value objects, called "Category".

In addition, in the data model, when a property is defined as a category facet, such a facet is always created from the values of the properties. For more information about properties, see [Using Properties to Configure Document Metas](#).

Even facets created for numerical data model properties, are always category facets. This means that each value of the property creates a category.

Create Facets

Create Facets Using the Data Model

This procedure applies to **category** facets only. Facets created by the data model are available as `Top/ClassProperties/Property_name`.

1. In the Administration Console, go to **Index > Data model**.
2. (Optional) Under **Classes**, click **Add class** and specify a new class.
3. Under **Properties for the <class name>**, you can either:
 - modify an existing property by selecting the **Category facet** option,
 - create a new property and select the **Category facet** option.
4. (Optional) Change the default configuration.

For more information, see the "Datamodel" section of the CloudView XML Configuration Reference Guide.

5. Click **Apply**.
6. On the Administration Console **Home** page, scan your data source.

The next time you search your index, this facet is included in the results.

Create Facets in the Search Logic

This procedure applies to both **category** (index-time) and **virtual** (search-time) facets.

1. In the Administration Console, go to **Search > Search Logics > Facets**.

2. Click **Add facets**.

- a. Specify a **Name**
- b. Select a **Type**.
- c. Click **Accept**.

3. Configure according to the facet type.

For more information, select the section corresponding to the facet type from the related topics panel above. See also the "Search" section of the CloudView XML Configuration Reference Guide.

4. Click **Apply**.

The next time you search your index, this facet is included in the results.

Create Facets Dynamically Using the Search API

You can also declare facets dynamically at query-time, on a per-query basis.

1. Use the "f" parameter as described in [Appendix - Search API Parameters](#).

Create Facets Dynamically in Mashup Builder

1. In Mashup Builder, select an application page, for example, the **search** page.
2. Select the **Feeds** view.
3. In the **CloudView Search** feed, expand the **Facets** section.
4. In **Add facet**, define one or more new facets. The syntax is **[facetId][key][value]**.

For a description of possible facet keys, see [Appendix - Search API Parameters](#).

5. Click **Apply**.

Numerical Range facets

Virtual numerical facets are used to organize search results in ranges.

Unlike category facets, the values created by a virtual numerical facet do not have a hierarchy. They are, however, attached to a "virtual root", which is a category path under which the virtual categories are created.

The computation is based on a virtual field expression, which allows you to categorize based on any calculation supported by the virtual expression syntax.

You then need to specify how you want to create these ranges.

Create Ranges Automatically

When facet ranges are created automatically, they generate ranges across the results of the query. However, this feature can be costly in terms of memory consumption.

The following range-generation policies are available:

- Linear
- Geometric

Linear Range Policy

A linear range policy defines ranges so each range includes a similar number of documents. The **Max number of ranges** option defines the number of ranges to generate and the parameter policy defines which policy must be applied.

For example, for these search results:

- 10 documents that each contain 1
- 9 documents that each contain 2,3,4,...,10

With the following configuration:

- **Range generation policy:** linear
- **Max. number of ranges:** 2

Displays the dynamic range facets as follows:

```
Top/
  dynamic2/      19
    [1;1]        10
    [2;10]        9
```

Geometric Range Policy

A geometric range policy accepts the same parameters as linear, except that each range contains twice as many documents as the range that follows.

Create Ranges That Are the Same Size

In this variant, a fixed range size is given, and a facet value is created for multiples of this range. For example, create a facet on price with a range size of 100 creates the following facet values:

- 0-100
- 101-200
- 201-300

Create Ranges Manually

In this variant, you can manually specify the ranges. Each range receives a title, which is the value of the category.

Range boundaries are inclusive, therefore, if you have 2 ranges `[0;1]` and `[1;2]`, 1 is in both ranges.

For example, we can define an explicit range facet on the expression:

`#now()` - `last_modified_date`, with the following ranges:

`min=0, max=86400*7 --> title = "Modified last week"`

`min=86400*7, max=86400*31 --> title = "Modified last 30 days"`

Date Facets

Date faceting is used to:

- Create a `Top/myDate/YYYY/MM/DD` category hierarchy at indexing time. This is a category date facet.
- Apply a `CategorySynthesis` to `Top/myDate` at search time. This is a virtual date facet, of which there are the following types:
 - **Date:** allows you to control date sorting for individual date units (year, month, and so on). See [Control Sorting with Static Date Facets](#).
 - **Dynamic date:** adapts dynamically to the chosen date range, so dates can display as years for multiyear date ranges, but as month/year or day/month/year for shorter date ranges. This is useful for displaying large date ranges in widgets. See [Control Date Display with Dynamic Date Facets](#).

Advantage of Virtual Date Facets

Using virtual date faceting saves on index, while providing greater flexibility.

For example, let us say that each document has a date/time of:

```
2013/05/12 15:05:12
```

Before virtual date faceting, date synthesis required that the analysis preprocess all dates and create a category per year, month, and day for them, as well as one per hour, minute, second if time synthesis was required.

Now, with virtual faceting, no specific analysis is required. In your search logic, you can define which categories to create. For example, for year and month faceting, create a `DateFacet` with `withYear` and `withMonth` set to `true`, and all others set to `false`.

Synthesis can be enabled independently for year, month, week, day, hour, minute, and second.

Note: `DateFacets` refinements behave differently for full dates (such as Year/month or Year/month/day/hour) than for other date-time combinations (such as Year/hour).

- For full dates, refinements are done efficiently with a numerical search.
- For other date-time combinations, refinements are done with a less efficient `attrnum` query.

Control Sorting with Static Date Facets

When sorting date facets by date, you can control the sort on a category by category basis.

For example, by default a date facet sorts categories by:

- years in descending order
- months in ascending order
- days in ascending order

MyDateFacet Date

Virtual root: Top/MyDateFacet

Expression: Edit

Categories: Year ▼ Month ▲ Week □ Day ▲ Hour □ Minute □ Second □

Sort by: Date ▼ i

In the Mashup UI's Refinement panel, the date facets are displayed like this:

▼ MyDate	
2013	2638
01	6
14	6
11	2341
22	2
27	6
28	2333
12	291
03	1
12	1
13	8
18	289
19	12
2012	11
05	3
25	1
29	2

However, you can change this by changing the arrows for each category in the facet. In the following example, years, months, and days are displayed in ascending order.

MyDate Date

Virtual root Top/MyDate

Expression document_lastmodifieddate Edit

Categories Year ▲ Month ▲ Week ▼ Day ▲ Hour ▼ Minute ▼ Second ▼

Sort by Date ▼ i

Control Date Display with Dynamic Date Facets

Use Dynamic Date facets to:

- Drill down in the Refinements panel. Depending on the date granularity you selected for the facet, you can drill from year to month, or from month to days. By contrast, static Date facets display all levels in a hierarchy.

Dynamic Date Facets Allow You to Drill down in the Refinements Panel

dynamic_date

Oct 2011	2
Nov 2011	2
Sep 2012	10

dynamic_date

2012/09/06	2
2012/09/07	2
2012/09/10	1
2012/09/11	5

- Control how many date values display in widgets.

Widgets have a limited width, which makes it hard to display large date ranges in a legible way. Dynamic date facets solve this by choosing a granularity that displays the most date values possible for the date range, without exceeding the maximum date values allowed. By default, this maximum is 40 but you can change it.

For example, if the search results included 1-year worth of documents, the facet initially displays 12 months. It uses a month granularity because displaying four quarters does not take full advantage of the number of date values allowed (40 by default), while displaying 52 weeks exceeds it.

However, if search results only included 2 weeks' worth of documents, the facet adapts to the shorter period by initially displaying 14 days.

- Fill in missing dates with null values, which is useful for time-series widgets.

If some documents are missing dates, there is an option to generate missing dates automatically, with a count of 0 and an aggregation value of `NO_VALUE`.

Note: You can only generate missing dates when sorting the dynamic date facets by date. The auto-generated dates do not display in the Refinements panel, but they do display in Mashup Builder chart widgets.

Configure Date Facets

Define Dynamic Date Facets

Dynamic date facets automatically adjust the granularity of the date display, according to the maximum number of facet values specified, and the date range for your search results.

1. Follow the steps to create a new facet, selecting **Dynamic date** for the **Type** field. For details, see [Create Facets in the Search Logic](#).
2. For **Expression**, define an expression based on a RAM-based index field for this facet.
3. For **Units**, select the granularity you want to be able to display. For example, select **Year**, **Month** and **Day**.
4. (Optional) In **Max. categories**, change how many date values this facet can display. By default this is 40.

The lower this number, the more likely dates display at a lower granularity (years instead of months, or weeks instead of days). Displaying lower-granularity dates is useful when representing large date ranges in widgets.

5. Select **Enable ISO 8601 compliance**, to get weeks starting on Monday and a few rules determining the first and last week of each year. Use compatible Output formats. Typically, use `%V` instead of `%U` otherwise week numbering is not consistent with grouping.
6. Click **Apply**.
7. In the Mashup UI, perform a search that returns a date range that spans several months or weeks.

As you refine on these search results, the dynamic date facet display on the Refinement panel. For an example of how dates adjust when displayed in a chart, see [Generate Missing Dates](#), Step 8 and Step 9.

Generate Missing Dates

1. In the dynamic date facet setup options, for **Sort by** check that **Date** is selected.
2. Select **Generate missing intervals**, if not already selected.
3. Click **Apply**.

4. In Mashup Builder, add a line chart widget to your /search page:
 - a. Go to the search page and select the Design view
 - b. In the Widgets panel, expand **Visualizations > Charts**, and drag the Line Chart widget on to the page.
5. Click the header of the Line Chart widget to fill in the required properties:
 - a. For **Facet type**, select **normal**.
 - b. For **X**, select the dynamic date facet from the list at left. In our example, this is `lastmodifieddate`.
 - c. Under **Aggregation**, select **count**.
 - d. For **C**
 - e. For **A**

Widget title: Documents by date

Facet type: normal

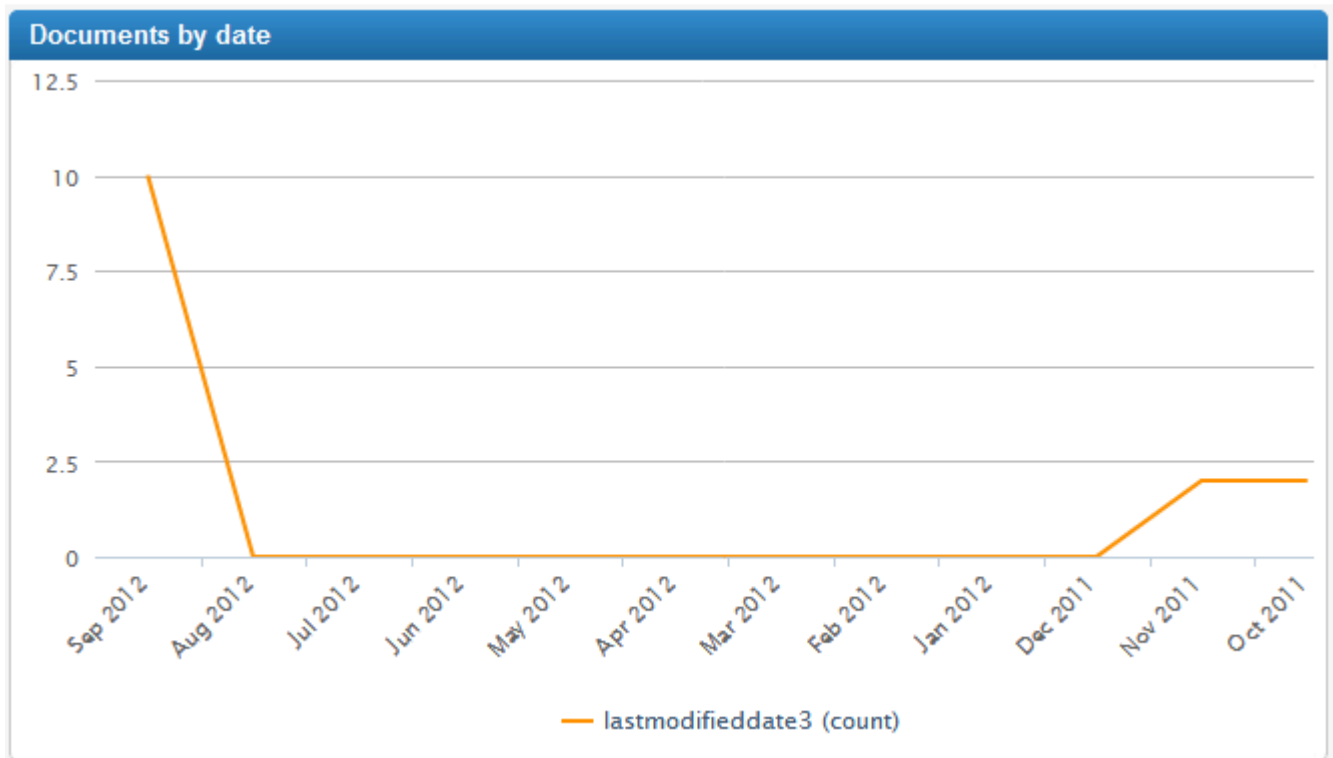
X: lastmodifieddate

Y: Aggregation Serie type Axis Legend

count line 1 \$facet.description

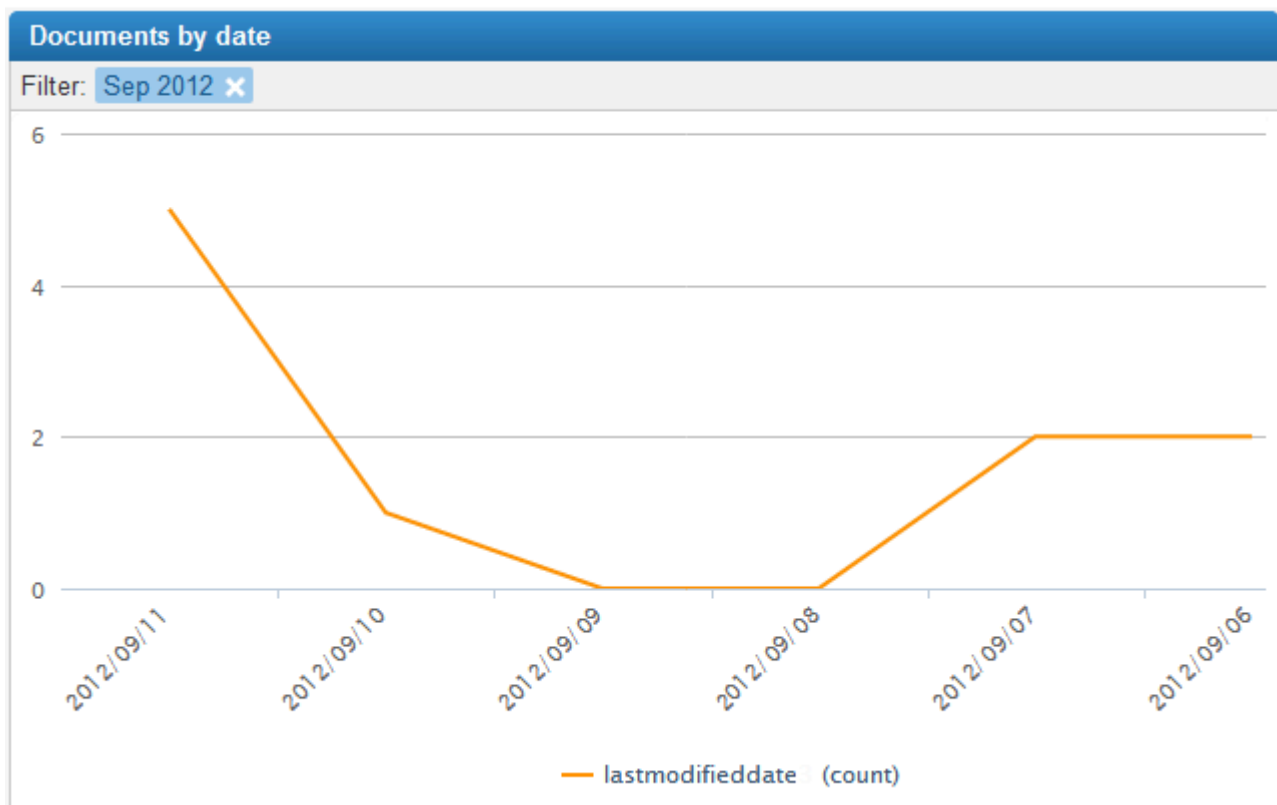
Merge or compare feeds: compare

6. Click **Preview** to make sure that the chart looks as expected.
7. Click **Apply**.
8. In the Mashup UI, perform a search that returns a date range that you know includes some missing dates.



Note: While the chart widget displays auto-generated missing dates, the Refinement panels does not. It only displays facet dates that exist in the underlying index field.

- Refine your search results to shorten the matching documents' date range.



Change the Display Format for the Dates

The output format is defined using UNIX date syntax.

1. In the dynamic date facet setup options, expand **Output formats**.
2. Modify the various output formats. See [Indexing Options for Date Properties](#).
3. Click **Apply**.

Multidimension Facets

Multi-Dimension facets and Hierarchical2D facets are different from the other facet types. They do not apply directly to expressions, but use the categories generated by two or more other facets (any type, virtual or not, but one-dimensional only) to create its own categories.

Multi-Dimension Facets

A `MultiDimensionFacet` takes the categories generated by N single-dimension facets, and generates a grid, with one facet per dimension as shown in the figure below. Each cell of the grid contains the number of documents and aggregation values that have a given category for each of the N one-dimension facets.

Here is an example with N=2 (two dimensions):

```
<CategoryFacet id="location" root="Top/loc"/>
<CategoryFacet id="date" root="Top/date"/>
<MultiDimensionFacet id="multi" virtualRoot="Top/multi">
  <MultiFacetDimension id="location"/>
  <MultiFacetDimension id="date"/>
</MultiDimensionFacet>
```

Which generates the following:

Example of Multidimension Facet

Refined on: Top/date/2010					Refined on: Top/date/2010 and Top/Centre				
	Top/date/2009	Top/date/2010	Top/date/2011	Top/date/2012		Top/date/2010/01	Top/date/2010/02	Top/date/2010/03	Top/date/2010/04
Top/loc/Auvergne	101	96	91	103	Top/loc/Auvergne	9	9	7	8
Top/loc/Centre	205	156	185	198	Top/loc/Centre	16	15	13	12
Top/loc/Ile-De-France	301	253	312	350	Top/loc/Ile-De-France	30	25	15	16

	Top/date/2010/01	Top/date/2010/02	Top/date/2010/03	Top/date/2010/04
Top/loc/Centre/Chartres	6	5	3	6
Top/loc/Centre/Dreux	4	5	3	5
Top/loc/Centre/Orleans	6	5	9	1

Important: When one of the two single-dimension facets used to build a Hierarchical 2D facet has no values (for example, when value facet F1 has /a/b/c categories, and value facet F2 has no

categories), the Hierarchical 2D facet does not display any categories. To bypass this behavior, you can create default values for the empty facet. To do so, go to **Data Model > Classes** and for the empty category facet, enter the required value in the **Default value** field.

Hierarchical2D Facets

A `Hierarchical2DFacets` creates a two-dimension (or matrix) hierarchy of categories. They enable the calculating of pivot tables. They are based on two single-dimension facets for the two axes.

For example, if you have a `country` category and a `sales` index field, you can define:

- a category facet on `country`
- a ranged facet on the `sales`
- a Hierarchical2D facet on the previous two, which calculates a matrix of sales per country

For example, let us consider a Hierarchical2DFacet, `hier2d`, that references facets `id1` and `id2`.

`id1` generates:

```
Top/root1/A/x
Top/root1/A/y
Top/root1/B/z
```

`id2` generates:

```
Top/root2/C/r
Top/root2/D/x
Top/root2/E/t
```

The documents have the following categories:

```
doc1: Top/root1/A/x, Top/root1/B/z, Top/root2/D/x
doc2: Top/root1/A/x, Top/root2/D/t
doc3: Top/root1/A/x, Top/root2/D/t
doc4: Top/root1/A/y, Top/root2/C/r
doc5: Top/root1/A/y, Top/root2/D/x, Top/root2/C/r
doc6: Top/root1/B/z, Top/root2/C/r
doc7: Top/root1/B/z, Top/root2/C/r
doc8: Top/root1/B/z, Top/root2/D/t
doc9: Top/root1/B/z, Top/root2/E/u
doc10: Top/root1/B/z, Top/root2/E/u
doc11: Top/root1/B/z, Top/root2/E/u
doc12: Top/root1/B/z, Top/root2/E/u
doc13: Top/root1/B/z, Top/root2/E/u
```

Therefore `hier2d` generates the following 2D hierarchy:

Example of Hierarchical2D Facet

Top/mRoot count=13	A count=5		B count=9
C count=4		y r c=2	z r c=3
D count=5	x x c=1	y x c=1	
	x t c=2		z t c=2
E count=5		y u c=1	z u c=5

Important: When one of the two single-dimension facets used to build a Hierarchical facet has no values (for example, when value facet F1 has /a/b/c categories, and value facet F2 has no categories), the Hierarchical facet does not display any categories. To bypass this behavior, you must create default values for the empty facet. To do so, you can either use the **Data Model > Classes > Default value** field or use a **Data Processing > Document Processors > Value Selector** processor with a **Does not exist** condition to indicate that the given meta is empty .

Displaying Multidimension Facets in Mashup UI

Mashup Builder includes several widgets that support Multi-Dimension and Hierarchical2D facets, such as:

- 2D Table
- Stacked Column Chart
- Mutable Chart
- Timeline

For more information, see the Widget Reference.

Geographic Facets

You can create facets based on geographic points (both GPS and XY types). Like other facets, they automatically display a count of matching documents for each facet value.

You can also aggregate document values that fall within one or several geographic areas.

You can define the following types of geographic facets:

- To return matches within a specified disk or polygon, create a **Geographic** facet. Use this facet when you know where to focus your search. You can define multiple disks or polygons for a facet. Each disk or polygon represents a facet value.
- To highlight areas containing matches within a bounding box, create an **Auto-tile geographic** facet. Use this facet when you want to locate areas containing matches (known as tiles) within a larger area.

By default, this bounding box (or the extent) is the entire globe, but you can modify this. Each tile represents a facet value.

Create Facets Based on Disks or Polygons

1. In the Administration Console, go to **Search > Search Logics > Facets**.
2. Click **Add facet**, specify a name, and select the **Geographic** type.
3. In **Index field**, select a geographic field from the list.
4. In **Domains**, for **Type** select the shape of the area, that is to say **Disk** or **Polygon** and specify the area's extent.

The coordinates depend on whether the facet is based on a GPS or XY point index field:

- For a disk: specify a center point and a radius.
- For a polygon: specify a list of coordinates, separated by semicolumns. Two successive points must be connected by an edge, and the polygon is automatically closed (an edge is added between the last and first point).

Important: The time required for facet synthesis is directly proportional to the number of points in the polygon.

5. (Optional) Add another disk or polygon area. You can mix polygons and disks within the same facet, and the areas can overlap.
6. (Optional) Expand Aggregation to create a summary on an index field for this facet.
7. Click **Apply**.

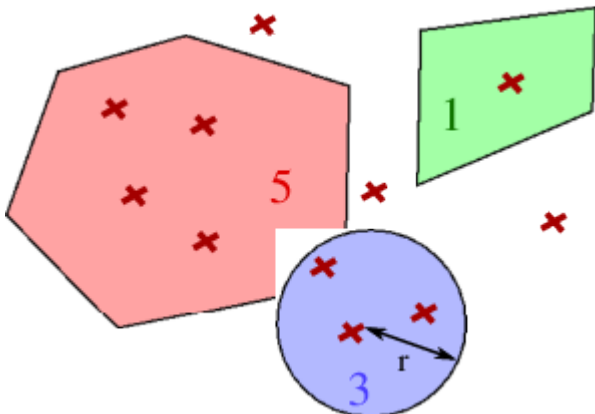
For an example (created in the XML configuration file, rather than in the Administration Console), see below.

Example 1. Example

This example uses an ExplicitGeoFacet with one DiskDomain (blue) and two PolygonDomains (red and green), as shown in the figure below.

```
<PolygonDomain title="redpoly" vertices="346,401;94,230;156,319;293,296;390,160;204,3
<PolygonDomain title="greenpoly" vertices="689,377;542,397;539,526;685,464"/>
<DiskDomain title="bluedisk" x="501" y="641" radius="78"/>
```

Example of an ExplicitGeoFacet with One DiskDomain (Blue) and Two PolygonDomains (Red and Green).



In this example, dark red crosses represent indexed points for each document. The red and blue domains overlap, which is allowed, and the point that falls within the intersection counts once for each domain.

Five points fall into the red polygon domain, three into the disk, and one into the green polygon domain. Three points fall outside all domains, and so are not counted in the results.

Create Facets Based on a Bounding Box

1. In the Administration Console, go to **Search > Search Logics > Facets**,
2. Click **Add facet**, specify a name, and select an **Auto-tile geographic** type.
3. In **Index field**, select a geographic field from the list.
4. By default, the bounding box is the entire globe. If you want to change this, the coordinates depend on whether the facet is based on a GPS or XY point index field:
 - a. **Bounding box min**: define the lower left limit.
 - b. **Bounding box max**: define the upper right limit.
 - c. **Tile size**: specify the size of the area to highlight when there is a matching point on the map.
5. (Optional) Expand Aggregation to create a summary on an index field for this facet.
6. Click **Apply**.

Display Geographic Facets in a Map Widget

1. In Mashup Builder, drag a Google Maps widget onto your page.
2. Select a feed for your widget.
3. In the widget properties, go to the **Based on Geo Facets** tab.
 - a. In **Geo facets list**, click inside the field, then select your geographic facets from the panel on the left.

- b. In **Aggregation**, click inside the field, then expand **Aggregation** from the panel on the left. From the list, select an aggregation. The higher this value, the darker the tile appears on the map.
4. Click **Apply**.

Create Value Facets for Nonhierarchical Metas

You can use value facets instead of category facets for alphanumeric metas that are not hierarchical and do not need to be tokenized. For example a list of countries, or file formats.

Value facets are saved under the `value` field, which is optimized for faster faceting.

Create a Value Facet Using the Data Model

1. In the Administration Console, go to **Index > Data Model**.
2. Select the appropriate class, and then click **Add property**.
 - **Data type**: `Alphanum`
 - **Semantic type**: accept the default, since no semantic processing is applied on a value field.
 - **Field type**: `Dedicated field and facet`
3. In the settings for this property, expand **Other advanced options**, and select **Enumerated**.
4. Click **Apply**.

The next time you scan the corresponding data source for this property, Exalead CloudView creates a `value` field for this property in the index, as well as a `value` facet in the search logic.

Create a Value Facet Using the Index Advanced Schema

1. In the Administration Console, go to **Index > Data Model > Advanced Schema** tab.
2. Click **Add field**. For **Type**, select **Value**.
3. Configure the field's options as required.
4. Click **Apply**.

The next time you scan the corresponding data source for this property, Exalead CloudView creates a `value` field for this property in the index, as well as a `value` facet in the search logic.

Create Aggregations for Facets

All facet types allow you to calculate aggregations for each value of the facet.

You can define an aggregation by:

- A virtual field expression defining the value to use for each hit.
- A function to create the aggregated value.

For example, if you have a facet based on the "country" of a sale, you can add an aggregation, on the `price * quantity` expression, using the "SUM" aggregation function. For each country, you get the total revenue generated by the sales on this country.

Available Aggregation Functions for Facets

Function	Description
MIN	The minimum value of the expression for all documents in the facet value.
MAX	The maximum value the expression for all documents in the facet value.
SUM	The total of all values of the expression over all documents in the facet value.
AVG	The average value of the expression over all documents in the facet value.
STDDEV	The standard deviation of all values of the expression for all documents in the facet value.
CENTILE (x)	Computes Nth percentile of the expression for all documents in the facet value, where N is a double between 0.0 and 100.0.
COUNT	The count of all documents that contain a value for the expression set in the facet value. Note: Do not confuse it with the category document count.
MAXDATE	Computes the max date based on index time. You can specify output format in a custom expression using date formats (see Which Facet Type Should I Use?).
MINDATE	Computes the min date based on index time. You can specify output format in a custom expression using date formats (see Which Facet Type Should I Use?).

Most Mashup UI widgets that display facets can use aggregations, instead of the simple count of the facet value.

The results of aggregation functions on NULL values in the Search API follows these conventions:

- `max(empty)` = empty (that is, no value)
- `avg(empty)` = empty (that is, no value)
- `sum(empty)` = empty (that is, no value)

These values are returned as follows in the Access API (no values for unvalued aggregations):

```
<exa:facet id="Language" path="Top/language" data="" description="Language" count="2"
refinable="true" refinementPolicy="exclusive" type="category" nbClippedCategories="0"
```

```

<exa:facetInfo key="sort" value="count"/>
<exa:facetInfo key="reverse" value="false"/>
<exa:facetInfo key="hasRefine" value="false"/>
<exa:aggregation id="mini" type="LONG"/>
<exa:aggregation id="moyen" type="LONG"/>
<exa:aggregation id="maxi" type="LONG"/>
<exa:category id="f/Language/fr" path="Top/language/fr" data="" description="fr" co
state="DISPLAYED" nbClippedChildren="0">
  <exa:aggregation id="mini" type="LONG"/>
  <exa:aggregation id="moyen" type="LONG"/>
  <exa:aggregation id="maxi" type="LONG"/>
</exa:category>
</exa:facet>

```

Exclusive vs. Disjunctive Refinements

The **Refinement policy** option specifies how search results are filtered when you select a facet value from the Refinements panel in a Mashup Builder application.

Exclusive Refinement

Exclusive is the standard refinement policy, it allows you to drill down on a single facet value, excluding all other values in the facet.

```
CategoryFacet(Top/Language, exclusive)
```

- Refines on the `language/en` facet.
- The result set only contains English documents.
- All other languages have disappeared from the navigation sidebar.

Disjunctive Refinement

Disjunctive refinement allows you to build a checkbox-based navigation. This navigation also allows you to drill down on a single facet value, while still displaying all other facet values on the Refinements panel.

Use this policy to refine on multiple facet values, which are combined using an `OR` operator when querying the index. To display check boxes next to each facet, select the **Disjunctive facets** option for the **Standard Facets** widget in Mashup Builder.

```
CategoryFacet(Top/Language, disjunctive)
```

- Refines on the `language/en` facet.
- The result set only contains English documents.

- All other languages are still included in the navigation sidebar, with counts on them. If you then refine on the `language/fr` facet, the result set contains both French and English documents.

Note that the `HorizontalFacets` widget displays only the first level of disjunctive facets.

No Refine

Use this when you need to group search results, but do not need to refine on them.

Calculating Results On-The-Fly with Virtual Fields

Virtual fields allow you to compute values from many elements of the Exalead CloudView index. The main purpose of a virtual field is to access stored index fields.

For example, a virtual field called `revenue` with expression `price * quantity` accesses the two fields and calculate the total price.

When to Use Virtual Fields

You can use virtual fields for multiple areas within your search application. For example, for a given hit, virtual fields can calculate:

- A meta to display in the search client.
- The value for the hit in a dynamic numerical facet (see below).
- The value for the hit in a facet aggregation (see below).
- Ranking elements (see below).
- A value to use for filtering queries.

For example, you can define a numerical prefix handler, `total_price`, allowing a user to queries directly on the total price, such as. `total_price: > 500`.

Performance Considerations

Virtual fields have a significant overhead at query time, because of the evaluation that must take place for each hit.

In some cases, this evaluation must absolutely be performed at query time.

For example, if your geographic expression calculates the distance between a hit and the current position of the user, then it is fully contextual to the query. However, if you find yourself using a large number of virtual fields with totally static expressions that are always used, it can be a good compromise to precompute them using processors in the analysis pipeline.

Virtual Field Syntax

The virtual fields syntax supports a wide range of built-in functions:

- All numerical operators
- Basic math functions
- Time manipulation functions
- Geographic manipulation functions; for more information, see [Configuring Geographic Search](#).

For a complete list of virtual field syntax, see [Appendix - Virtual Field Expression Syntax](#).

Specifying a Timezone for Date Time Metas

If a timezone is detected inside the date time value, it is interpreted to convert and store this value in UTC format. By default, date values are also restituted at search time in UTC format.

There are however several methods to adjust the display of date time values to a specific timezone.

Specify a Timezone in the Output Format

The best method is to specify a default timezone output format for the date time meta (`%Y-%m-%dT%H:%M:%S+00:00`). This method allows you to get a complete date format that is easy to adjust to the web browser timezone settings.

1. Go to **Search Logics > Hit Content**.
2. Expand the date time meta and click **Customize** if it was generated by the Data Model.
3. Click **Add operation** to add a **Time format** operation.
4. Expand **Time format** and for **Output format**, enter a format including a timezone, for example, `%Y-%m-%dT%H:%M:%S+00:00`
5. Click **Apply**.

Convert Date Time Values to a Specific Timezone

The following procedures describe how to adjust the timezone server-side to match the end user's timezone specified at query time. The goal of these procedures is to display date time values with the end user's setting for both hit metas and facets.

To Adjust Date Time for Hit Metas

To adjust date time for hit metas, you need to use virtual field.

1. Go to **Search Logics > Virtual Fields**.
2. Add a new virtual field and give it a name, for example `date_timezone_adjusted`
3. Click **Edit** and in the **Expression builder**:
 - a. In **Functions**, double-click the **#adjust_timezone** function to insert it in the **Expression** field.
 - b. Place your cursor between the parentheses and in **Index fields**, and set how to adapt the time display.

Your expression must be: `#adjust_timezone(<Name of date time field>)`

4. Go to **Search Logics > Hit Content**.
5. Expand the date time meta and from **Index field**, select the virtual field you have created.

To Adjust Date Time for Facets

To adjust date time values for a date facet, you need to change its formula.

1. Go to **Search Logics > Facets**.
2. Expand your date facet, and click **Edit** next to the **Expression** field:
 - a. In **Functions**, double-click the **#adjust_timezone** function to insert it in the **Expression** field.
 - b. Place your cursor between the parentheses and from **Index fields**, double-click the index field impacted by the timezone adjustment.

Your expression must be: `#adjust_timezone(<Name of date time field>)`

To Adjust the Default Timezone

You finally have to adjust the default timezone.

1. Go to **Search Logics > Locale Settings**.
2. Specify the **Default timezone** that applies to all date values. This is calculated based on the difference with the UTC timezone.
3. Click **Apply**.

End users can now specify the timezone of their choice at search time. See [Specify a Timezone at Search Time](#). This overrides the default timezone specified in step 2.

Specify a Timezone at Search Time

Once the server-side configuration is prepared to convert date time values, end users can specify the timezone of their choice at search time.

To Specify a Timezone at Search Time in the Search API

You can also specify a timezone as a query parameter to search for documents precisely.

1. In the Search API, use the `timezone` or `tz` parameter. For example, to adjust the timezone to UTC -01:30 (UTC being the default timezone), you can enter a query like: `http://<HOSTNAME>:<BASEPORT+10>/search?q=%23all&tz=-01:30`

For more information, see [Appendix - Search API Parameters](#).

To Specify a Timezone at Search Time with a UQL Prefix Handler

It is also possible to specify a timezone at search time when using Date prefix handlers.

1. For example, to get all documents created after 9:30AM on February 12, 2016, with a timezone of UTC -02:00, you can enter: `date>="Fri 12 Feb 2016 09:30:00 -02:00"` or any other supported format between quotes.

How is the Timezone Handled in Mashup UI Applications?

When creating Mashup UI applications with the Mashup Builder, the user's browser timezone is automatically detected and sent to each query.

For more information, see the **Cookie to parameter** prerequisite trigger description in "Add triggers to an application or a page" (in the Exalead CloudView Mashup Builder User's Guide).

Ranking and Sorting Search Results

The ranking and sorting you define for a search logic controls the order in which hits display in the search results.

There are two phases in ranking and sorting: calculating the ranking elements, then defining sorting and grouping using these ranking elements.

About Ranking

Sorting

About Ranking

In the default configuration, Exalead CloudView uses a single ranking element, which is defined by a virtual field called `text_relevance`.

Default Ranking Model

The `text_relevance` virtual field is used in a single `SortBy` clause. The expression of this virtual field is: `@term.score * @proximity + @b`

Where

- `@term.score` – A value assigned to each alphanumeric node in a query. A node's `term.score` value is determined by the textual ranking algorithm for the node. For more information, see [Term Scoring](#).
- `@proximity` – proximity boost, applied to the document as a whole. For more information, see [Proximity Boost](#).
- `@b` – boost. It is a node property that is commonly used to indicate that elements that match a particular term must be boosted. Boost is defined on a query-by-query basis. For more information, see [Boost](#).

Term Scoring

Each alphanumeric node in a query has a special property, called a `term.score`. A node's `term.score` value is the result of the textual ranking algorithm for the node.

The `term.score` uses the default merge policy, which is to sum its values over the whole query. In the Administration Console, it can be set in **Search > Search Logics > Sort & Relevance > Term Scoring**.

Scoring Algorithms

The following table describes the available scoring algorithms.

Note: TF-IDF, IDF, and BM25 are standards, and not described in this section. For more information about them, look for documentation on the internet.

Algorithm	Description
No Ranking	With No ranking , the term score is always 0, for all alphanumeric nodes of the query. When term scoring is not really required, disabling <code>term.score</code> significantly improves hit matching performance (by up to +30%).
Rank	The Rank term score uses only the statically defined rank of each word. In the index, each word can have a rank for each document. The <code>term.score</code> value is <code>rank * w</code> .

Algorithm	Description
	<p>w is a special node option, which can be set on each alphanumeric value, both in ELLQL and in UQL:</p> <ul style="list-style-type: none"> • in UQL: <code>a OR b{w=2}</code> • in ELLQL: <code>#alphanum{w=2}(text, "a")</code> <p>The default value of w is 1.0</p> <p>Use w to increase, decrease, or cancel the importance of the presence of one word with a specific rank.</p> <p>For example, for query <code>a AND b</code> we have two matching documents:</p> <ul style="list-style-type: none"> • doc 1: <code>a[rank=4] b[rank=6]</code> • doc 2: <code>a[rank=6] b[rank=4]</code> <p>With the default configuration, both doc1 and doc2 have <code>term.score=10</code>.</p> <p>With the query <code>a AND b{w=2}</code>:</p> <ul style="list-style-type: none"> • doc1 has <code>term.score=16</code> • doc2 has <code>term.score=14</code> <p>You can also use w to ignore a given word for the textual relevance calculation, by setting $w=0$.</p>
Rank IDF	<p>The Rank IDF term score adds the notion of IDF, or Inverse Document Frequency. The IDF represents the relative rarity of a word in a corpus. The more frequently the word appears in the corpus, the lower its IDF. The idea behind this algorithm, is the rarer the word, the greater its importance.</p> <p>For example, on query <code>the OR economy</code>, we want the documents matching <code>economy</code> first, because they are more specific.</p> <p>For a given word, $IDF(word) = 1 + \log(\text{number of docs in corpus} / \text{number of docs containing this word})$</p> <p>The <code>term.score</code> of one word with this algorithm is: <code>rank * w * idf * 10000</code></p> <p>IDF is a positive double above 1.0 (for a word that is in all documents).</p> <p>For example, for a word present in only one document out of a corpus of one million, $IDF = 20.9$</p>
Rank TF-IDF	<p>The Rank TF-IDF term score adds the notion of Term Frequency. To represent the importance of a term within a document, it takes into account term density instead of term occurrences.</p> <p>For example, we have the query: <code>iphone</code> and the following documents:</p>

Algorithm	Description
	<ul style="list-style-type: none"> Doc1: {iPhone} Doc2: {iPhone accessories} <p>Both have the same number of <code>iphone</code> occurrences, but doc1 is more dense with <code>iphone</code> and intuitively a better match. We consider that the number of occurrences is not as meaningful as the term density.</p> <p>To use this algorithm, go to Data Model > Advanced Schema, click the index field to modify, select Compute TF, and click Apply.</p> <p>For a word <code>w</code> in a document <code>d</code>, a simple version of TF would be:</p> $\text{SimpleTF}(w, d) = (\text{number of occurrences of } w \text{ in } d).$ <p>To avoid overranking documents where a word occurs frequently, Exalead CloudView uses a more advanced version of the formula:</p> $\text{TF}(w, d) = (2.2 * \text{SimpleTF}(w, d) / (1.2 + \text{SimpleTF}(w, d)))$ <p>The <code>term.score</code> of one word with this algorithm is: <code>rank * w * tf * idf * 10000</code></p> <p>TF varies between 1 (for a word present only once) to 2.2. Therefore, the <code>term.score</code> varies between <code>rank * w * 10000</code> and <code>rank * w * 10000 * 2.2</code>.</p>
BM25F	<p>TF-IDF does not use the length of the document to normalize the term frequency. The BM25 term score uses a more complete version the TF formula:</p> $\text{SimpleTF}(w, d) = (\text{number of occurrences of } w \text{ in } d) * (\text{length of the document} / (\text{average length of all the documents}))$ <p>As for TF-IDF, this SimpleTF is normalized to avoid overranking. Moreover, Exalead CloudView combines this term frequency with the TF-IDF value, using the following formula:</p> $\text{TF}(w, d) = \frac{\text{SimpleTF}(w, d) \times 2.2}{0.55 + 1.65 \times \frac{\text{document length}}{\text{average documents length}} + \text{SimpleTF}(w, d)}$ <p>The <code>term.score</code> of one word with this algorithm is: <code>rank * w * tf * idf * 10000</code>.</p> <p>TF varies between almost 0 (for a word that occurs once in a very large document) and 2.2 (for a word that occurs once in a small document and where all the other documents are large).</p>
Custom	<p>You can define your own custom ranking by selecting the Custom scoring algorithm and defining a formula.</p>

Ranks Remapping

During indexing, a static rank or relevance class is set for each meta. This relevance class is a numerical value that is used to rank search results.

You can display current relevance classes for each meta in **Index > Data Processing > Mappings > Details > Relevancy options > Relevance class**. Nine values are available (from 0 to 8), 8 being the highest rank:

- 0: No score
- 1: Hidden text
- 2: Text
- 3: Boosted text
- 4: Relevant text
- 5: Boosted relevant text
- 6: Title
- 7: Boosted title
- 8: URL

After indexing, relevance classes cannot be modified anymore. To change the relevance class set for a meta, you must use the **Ranks remapping** field in **Search > Search Logics > Sort & Relevance > Term scoring**. Use numerical values in increasing order and separated by commas to set the new rank of existing relevance classes. Example: I want to give more weight on titles. I must specify that titles (relevance class=6) have now the highest rank (relevance class=8). I fill the **Ranks remapping** field as follows: 0,1,2,3,4,5,6,9,10.

Proximity Boost

A special ranking element called proximity is the result of the proximity algorithm.

Proximity is a double value, between 0 and 10, where 0 is out of range.

To set proximity boost in the Administration Console, go to **Search > Search Logics > Sort & Relevance > Proximity boost**.

Boost

Boost, or b , is summed over all nodes.

- in UQL: `a OR b{b=100}`
- in ELLQL, like all node properties: `#alphanum{b=1000}(text, "a")`

A common use of `b` is to assign a score for nodes that do not normally have them, like categories:

```
a AND b AND (source:important_source{b=1000} OR
source:less_important_source{b=0})
```

The score of an alphanumeric value can be forced, replacing the `term.score`, by setting `{w=0,b=DESIRED_SCORE}`.

`b` can also be negative, to unboost certain terms.

Custom Ranking Elements

For advanced ranking use cases, you can create custom numerical key-value pairs attached to each node of the query tree to use as ranking elements.

For example, to create a behavior similar to the boost one, we can define the following query:

- in UQL: `fruit{interest=1} tomato{interest=10}`
- in ELLQ: `#and(#alphanum{interest=1}(text, "fruit")
#alphanum{interest=10}(text, "tomato"))`

With this query, a hit that matches:

- `fruit` has a lower score of 1 as custom ranking element.
- both `fruit` and `tomato` have 11 as custom ranking element.

By adding a sort expression on `@interest`, we get the interesting hits first.

The default policy is to sum the values for numerical ranking keys, from all children nodes where it matches. You can also keep the maximum or minimum values among children:

- `#and{interest.policy=MAX} (#alphanum{interest=10}(text, "tomato")
#alphanum{interest=1}(text, "fruit")) (score 10)`
- `#and{interest.policy=MIN} (#alphanum{interest=10}(text, "tomato")
#alphanum{interest=1}(text, "fruit")) (score 1)`

Reusing Ranking Elements in Virtual Fields

You can re-use node properties and predefined ranking elements as expressions in the virtual field syntax for:

- Constructing higher-level ranking elements
- Metas
- Dynamic faceting
- Faceting aggregations

For more information, see [Calculating Results On-The-Fly with Virtual Fields](#).

For example, if you define a complex ranking element to calculate the relevance of a hit, you may want to reuse this calculated value to compute an aggregation, which is the sum of this relevance score for each value of a facet, indicating the "total relevance" of this facet value.

The syntax to access a given ranking element is `@elementname`.

As the ranking elements are computed once a hit has been identified, there is a major restriction, which is that they generally cannot be used in a virtual field for querying. For example, you cannot use `#attrnum(@proximity, ==, 42)` because when we want to evaluate whether the hit matches, the proximity has not yet been computed.

The main consequence is that if you define a numerical facet, which uses a ranking element, you cannot refine on it. For example, if you defined a numerical facet with expression `#floor(@proximity)`, you can use this to obtain a histogram of the documents by the proximity of the query terms within them. However, you cannot refine, because a query "I want all documents where the computed proximity score is between 1.3 and 1.7" is not supported.

One exception is the `#filter` ELLQL node, see [Filtering Search Results in ELLQL](#).

Sorting

You can perform sorting on RAM-based retrievable fields. In the Data model property, or advanced schema index field configuration, select the **RAM based** option for all the fields on which you want to apply sorting.

You can specify one or several sort clauses to be treated one after the other cumulatively. For example, if our index schema has the `lastname` and `age` index fields, we could want to sort:

- First on the `lastname` field by ascending order.
- Then on the `age` field by descending order.

Note: Sorting can also be performed dynamically at query time using specific Search API parameters. For more information, see the [Sorting and Grouping Parameters](#) in the Search API Parameters Reference.

Set Sorting in the Administration Console

1. Go to **Search > Search Logics > Sort & Relevance > Sort**
2. Click to add as many sort clauses as required.
3. For each clause, you must specify:
 - a **Name**
 - a virtual **Expression** - click **Edit** and specify the expression with the **Expression Builder**. It can be the name of an index field only or a more complex virtual field expression.

For example, if you want the sort to be in lowercase, use the `#strlower` function, to make an expression like `#strlower(myAlphanumericField1)`. If you want the sort to be normalized (that is, lowercase and unaccentuated), use the `#strnormalize` function, to make an expression like `#strnormalize(myAlphanumericField2)`.

- A **Sort order** (ascending/descending).
- And if the sort clause is **Active**, so that it is executed.

Reduce the Performance Overhead at Query-Time

Alphanumeric sorting has a high-performance overhead at query-time.

1. To reduce this performance issue, you can set a limit at query time using the **Chars limit** option, which is the maximum number of characters on which to perform the sort.

Only documents that do differ on the first limit characters are ordered.

Collapsing/ Grouping Search Results

Collapsing or grouping search results means keeping similar results together so they display in a concise, readable way.

There are typically two scenarios where you want to collapse search results:

- To display a concise list of results, often based on multiple criteria. You do not want to explore, you want to see specific hits.

To do this, use grouping as explained in this section.

Examples:

- For web search, you want to keep only one result per site.
- You want a list of every Apple phone that had more than 1000 units sold in the past week, worldwide, regardless of store or country.
- To see the overall distribution of search results in a dashboard, and to explore the results by drilling down on certain areas.

To do this, use faceting. See [Creating Facets to Refine Search Results](#).

For example, you want to know how different Apple products are selling in different countries.

About Grouping

Setting Up Grouping

About Grouping

The main purpose of grouping is to remove duplicates so search results are easier for users to read.

With grouping, your results only retains a hit (or N hits) for each group. The particular hit that displays for each group is determined by how you sort the hits within the group. Within these groups, you can nest additional grouping clauses.

Important: As with other search-time manipulations like facets and sorting, you can only define group or group sorting expressions on fields that are stored in RAM.

While grouping in Exalead CloudView works similar to the SQL GroupBy in that you can use multiple clauses and create aggregations, keep in mind that it is not identical to database grouping.

For search results, the ultimate purpose to collapse several hits into one (or several) "representative" hit. The following examples demonstrate how grouping works.

Simple Grouping Example

Say that you are sporting gear retailer. Spring is coming and you want to find out what kind of bike merchandise you have in stock.

Without using any grouping, here are the results for a query on `bike`:

Hit rank in results	Name	Quantity	Warehouse distance (km)
1	Bike	10	100
2	Bike helmet	100	1
3	Bike seat	50	12
4	Bike shoes	20	28
5	Biking jersey	500	5

You want to reduce this list by grouping the results on which the quantity is higher than 50. To do so, define a group where `quantity > 50`.

How Many Search Results Do You Have?

Two, because defining a group creates 2 groups:

- More than 50: `Bike helmet, Biking jersey`

- 50 or fewer (the "other" group): `Bike`, `Bike seat`, `Bike shoes`.

Which hits Do Display for the Group?

This is defined by:

- The number of hits to represent the group. Let us keep the default, 1. This means that 2 hits are displayed in the search results, one for each group.
- The sorting expression for the group. In our example, you want to know what's close by, so let us sort on the distance field in ascending order. This means the hit with the lowest distance value in the group represents the group.

In the grouped search results, this returns:

- One hit from More than 50, `Biking helmet` (because `distance = 1 km`, less than for `Biking jersey`).
- One hit from 50 or fewer, `Bike seat` (because `distance = 12 km`, less than for `Bike` or `Bike shoes`).

Which Group Displays First in the Search Results?

The group sorting determines which hit represents the group. But ultimately it is the representative hit's relevance score before grouping that determines the group's rank (display order) in the search results.

In this case, `Bike seat` ranked higher than `Biking jersey` before grouping. So the grouped results look like this:

Grouped hit rank	Name	Quantity	Warehouse distance (km)
1	Bike seat	50	1 km
2	Biking jersey	500	5 km

Multicriteria Grouping Example

You can add additional clauses to group on multiple criteria.

Without using any grouping, here are the results for your `bike` query, this time with country information.

Hit ranking	Name	Quantity	Warehouse distance (km)	Country
1	Bike	10	100	Germany
2	Bike helmet	100	1	France

Hit ranking	Name	Quantity	Warehouse distance (km)	Country
3	Bike seat	50	12	Germany
4	Bike shoes	20	28	France
5	Biking jersey	500	5	France

In addition to grouping by quantity, you also want to group by country.

This means you now have three groups:

- More than 50 in France: `Bike helmet`, `Biking jersey`.
- 50 or fewer in France: `Bike shoes`
- 50 or fewer in Germany: `Bike`, `Bike seat`.

Still displaying only one hit from each group, and sorting in ascending order on `distance`, this will return three hits:

- `Bike helmet` from More than 50 in France
- `Bike shoes` from 50 or fewer in France
- `Bike seat` from 50 or fewer in Germany

Since CloudView ranks these "representative" hits to their initial rank before being grouped, the hits display in this order:

Grouped hit rank	Name	Quantity	Warehouse distance (km)	Country
1	Bike helmet	100	1	France
2	Bike seat	50	12	Germany
3	Bike shoes	20	28	France

Displaying an Aggregation Example

You can also calculate an aggregation for group and display it in the grouped hit. Note that the aggregation is based on all hits within the group, regardless how many hits display in the grouped hit.

Available aggregations:

- **Sum**
- **Max**
- **Min**
- **Average**

- **Standard deviation**
- **Concatenation** - this lists all the values for a field within the group. Unlike other aggregations, it needs an alphanumeric field.

Say that you want to display the total quantity in stock for all hits within a group. Let us go back to the simple grouping example with only two groups.

Group	Name	Quantity
More than 50	Biking jersey	500
	Bike helmet	100
50 or fewer	Bike seat	50
	Bike	10
	Biking shoes	20

By creating a `sum` aggregation on the `quantity` field, you can display the aggregation in your grouped hit:

Grouped hit rank	Name	Quantity	Total quantity for group
1	Bike seat	50	80
2	Biking jersey	500	600

Setting Up Grouping

Add a Group

1. In the Administration Console, go to **Search > Search Logics > Your search logic > Sort & Relevance**.
2. Under **Group**, click **Enable**.
3. Define your grouping and sorting criteria by clicking **Edit**.

For our example, specify:

- a. **Group by:** `quantity > 50`.
- b. **Sort hits in group by:** `distance`.

When in the Expression builder, if you do not see the field that you want on the list of **Index fields**, it is because the field is not being stored in RAM. You need to enable that field's RAM-based option, apply your changes, then clear your build group & reindex.

4. (Optional) Specify an additional clause to your **Group by** expression by clicking **Add clause**.
For an example of how additional clauses impact your search results, see [Multicriteria Grouping Example](#).
5. Click **Apply**.

Create an Aggregation for a Group

1. In the **Group** section, expand **Aggregations**
2. Select the type of aggregation. For our example, select **Sum**.
 - a. Enter an expression. For our example, type `quantity`.
 - b. Specify the name of the meta to store the aggregation. For our example, type `total_quantity_for_group`.
3. Click **Apply**.

You can now add a hit meta to display this aggregation.

Display the Aggregation in the Grouped Hit

1. In your aggregation definition, copy the meta name that is storing the aggregation to the clipboard.
2. On the **Hit content** tab, click **Add meta**.
3. Paste the meta name into the **Name** box. In our example, this is `total_quantity_for_group`.
4. In the setup panel for this meta, click **Add meta source**.
5. In **Index field**, paste the aggregation's meta name again: `total_quantity_for_group`.
6. Click **Apply**.

The next time you perform a search in your application, this new meta display in your grouped hits.

Setting the Limits of Search Results

Exalead CloudView provides advanced control on query execution, and allows you to define a compromise between search performance and exhaustiveness.

Fully exhaustive search is available, but it is also possible to impose timeouts and limits on the number of matching hits. When a timeout or limit is reached, partial results are returned.

1. Go to **Search > Search Logics > Limits**
2. Set the limits as required and click **Apply**.

Managing Saved Configurations

This chapter describes how to manage saved Configurations of CloudView.

[About Saved Configurations](#)

[Comparing Configuration Versions](#)

[Rolling Back to a Previous Configuration](#)

[Editing the Configuration Manually](#)

[Apply changes when Exalead CloudView has stopped](#)

About Saved Configurations

Exalead CloudView has a versioned configuration. When you save a modification in the Administration Console, the new configuration is not immediately applied to the running instance. Instead, it is stored in the configuration store. When you apply the configuration, it is transferred to all running components.

The configuration store is located in `<DATADIR>/config`. It contains the latest version of all configurations, stored as XML files. For a list of the available configurations, see the CloudView XML Configuration Reference.

How Applying Configuration Works

Exalead CloudView's configuration defines the high-level functional setup of the product. For example, in `/config/deployment.xml` your Exalead CloudView installation's server deployment is defined through roles such as `Indexing`, `SearchAPI`, or `ConnectorsServer`.

In the running Exalead CloudView instance, these roles are mapped to running processes.

When you apply the configuration, Exalead CloudView checks the high-level configuration for errors, then computes a low-level configuration for each process, known as the GCT. This process is described below.

Apply Configuration Process

When Exalead CloudView applies a configuration, it does the following:

1. Checks the entire configuration store for errors.
2. Computes and creates a new version of the detailed configuration (the GCT).
3. The GCT contains a list of services that run on each process, the configuration of each process, and the configuration for dynamic interprocess configuration.

4. Saves the new version of the GCT and the current state of the configuration store for future rollbacks.
5. Sends the new configuration to all hosts on the Exalead CloudView cluster.
6. Dynamically reloads the configuration for the running processes.
 - If Exalead CloudView cannot dynamically load some configuration changes, it restarts the affected processes.
 - If some roles were added, Exalead CloudView starts the new processes.

Note: When applying deployment changes (for example, adding new roles), we recommend restarting Exalead CloudView afterward. If restarting is required, you are prompted to restart when you apply the configuration.

Comparing Configuration Versions

You can select two configuration versions and compare their XML configuration files through the Administration Console. This can be helpful to roll back to a previous configuration version.

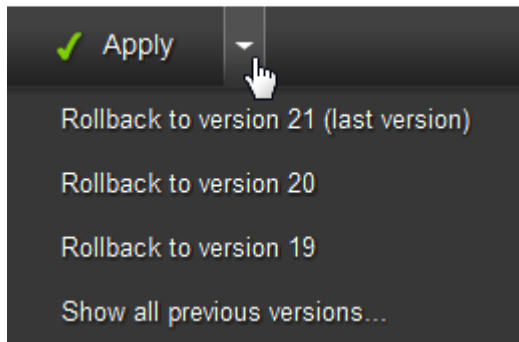
1. From the top navigation bar of the Administration Console, click the down arrow next to **Apply**.
2. Select **Show all previous versions**.
3. Select two configuration versions and click **Compare versions**.
4. The **Changed files** window opens, highlighting the differences between the two selected versions.

Use the Compare file select box to select XML configuration files.

Rolling Back to a Previous Configuration

You can roll back to a previous configuration version through the Administration Console. This replaces the entire configuration store with the configuration from the time the changes were applied.

1. From the top navigation bar of the Administration Console, click the down arrow next to **Apply**.
2. Select **Rollback to version** for the required version.



Editing the Configuration Manually

Most configuration changes are made in the Administration Console, which is an easy to use web interface. However, you must manually modify the configuration store to do some advanced configuration changes.

You can edit the configuration store:

- By using the API Console
- By editing the XML configuration files

Edit a File in the API Console

The API Console is a graphical interface for the Management API (MAMI).

It allows you to edit most configuration files (except for Mashup Builder and Business Console configurations), inspect product status and perform administrative tasks.

Note: For more information about the API Console, see "Perform advanced operations with the API console" in the Exalead CloudView Administration Guide.

1. Go the **API Console** and click **Manage**.
2. Locate the entry point where you want to edit a configuration file. For example, to edit the `IndexSchemaList` configuration:
 - a. **Select MAMI indexing.**
 - b. Click the `set<ConfigName>` method.
 - c. Edit the file.
 - d. Click **Save**.
3. In the **Manage** section of the API Console, click **Apply**.

If the configuration is invalid, an error message displays and your changes are not saved.

Edit the Configuration Files Directly

Before you do this, make sure no one is working in the Administration Console, to avoid editing conflicts.

1. If you have access to the `<DATADIR>`, you can edit the XML files in `<DATADIR>/config`.

When you save your changes to the file, there is no configuration check. This happens only when you apply your changes, as described below.

Apply Changes in the Command Line

If you have modified the configuration manually, **Apply** is not available in the Administration Console.

You have the choice between applying the configuration using the API Console, as described in [Edit a File in the API Console](#), or use the command line.

The `cvcmd` command-line tool provides access to the configuration application. Exalead CloudView must be running for this to work.

1. Go to `<DATADIR>/bin`.
2. Test your configuration: `cvcmd testConfig`
3. If no errors are returned, apply the changes: `cvcmd applyConfig`

Apply changes when Exalead CloudView has stopped

In rare cases, a severe configuration conflict can prevent Exalead CloudView from starting. As a result, the standard tools to apply changes will be unavailable.

For this scenario, use the `buildgct` emergency configuration application tool.

You must follow this procedure exactly as described for it to work.

Important: Do not use `buildgct` except for this procedure, especially for multi-host deployments. The `buildgct` tool does not contact the other hosts to synchronize the configurations, so it can cause severe inconsistencies in the product configuration.

1. Stop Exalead CloudView on all servers.
2. Edit the configuration files to fix the error.
3. Run `<DATADIR>/bin/buildgct`.
4. If it fails, continue editing.
5. Restart the master host.

Apply changes when Exalead CloudView has stopped

6. Wait for the master host to be fully started.
7. Restart the other hosts, if any.

Troubleshooting

This chapter describes how to troubleshoot common document analysis issues and how to analyze queries.

Troubleshooting Document Analysis

Analyzing User Queries with Reporters

Troubleshooting Document Analysis

Below is a list of potential issues with document analysis and their solutions.

Identify the Cause of the Index Crash

Unexpected Search Behavior

Identify the Cause of the Index Crash

The following procedure describes how to identify the cause of the crash.

1. Go to the `log.log` file in `<DATADIR>\run\indexingserver-bg0\`
2. Look for `'ERROR'`.
3. You need to locate:
 - document URI
 - processor type
 - method

The document URI is added to the block list file `indexing_uri_blacklist.txt` in `<DATADIR>\gct`. It is not indexed anymore.

4. Contact support for advanced analysis (file format, character strings etc.).

Example:

```
@@CRITICAL ERROR with "/%2Fdata%2Fcorpus%2Fdepeches%2Fvrac%2Ffr/3_2006-06-21T1226_FAP4251.txt"
in LanguageDetector_process (com.exalead.indexing.analysis.processors.cpp:622):
caught abort (signal 6) from tkill (code -6)
```

Key elements are:

- document URI: `3_2006-06-21T1226_FAP4251.txt`
- processor type: `LanguageDetector`
- method: `process`

Unexpected Search Behavior

Search for Issues in Document Processing

Follow the steps below to identify document processor issues.

Step 1 - to Add log Information

To display detailed chunks in the `log.log` file, you must first add a **Debug Processor** element to your document processors list.

1. From the Administration Console, go to **Index > Data processing > Pipeline name**.
2. In the **Document Processors** tab, click **Other** in the **Processor types** menu.
3. Drag the **Debug Processor** element to the end of the processors list.

Example: `<Debug>` tags are displayed:

```
[2013/09/23-09:54:11.584] [info] [AnalyzerThread-bg0-default_model-1] [analysis.debug
C:\Users\E7G\Downloads\traffic.csv8 source: RATP did: 57 slice: 0: DebugProcessor: d
C:\Users\E7G\Downloads\traffic.csv8DebugProcessor: dumping C:\Users\E7G\Downloads\tr
<DebugChunk type="TextChunk" ctx="ville" deleted="false" part="null" value="Paris" so
</DebugChunk>
<DebugChunk type="TextChunk" ctx="arrondissement" deleted="false" part="null" value="
language="xx"> </DebugChunk>
<DebugChunk type="TextChunk" ctx="source" deleted="false" part="null" value="RATP" so
</DebugChunk>
```

Step 2 - to Submit Document Using cvdebug

1. Submit your document to test processors in your pipeline:

```
cvconsole cvdebug> analysis analyze path=<PATH_TO_DOCUMENT>
```

For example, submit a .CSV file.

```
cvconsole cvdebug> analysis analyze path=/tests/myfile.csv
```

The output is a mapping of contexts and chunk values.

```
<TestAnalysisPipelineOutput xmlns="com.exalead.indexing.analysis.v10" documentProc
semanticAndMappingTimeUS=0>
  <##default>
    <DocumentProcessorsOutput xmlns="com.exalead.indexing.analysis.v10">
      <Document>
        <Document xmlns="com.exalead.ndoc.v10">
          <element>
            [
              <Context xmlns="com.exalead.ndoc.v10" language="xx" name="source"/>,
              <ScoreContext xmlns="com.exalead.ndoc.v10" value=0/>,
              <Chunk xmlns="com.exalead.ndoc.v10" value="sourceTest"/>,
```

```

    <Context xmlns="com.exalead.ndoc.v10" language="xx" name="uri"/>,
    <ScoreContext xmlns="com.exalead.ndoc.v10" value=0/>,
    <Chunk xmlns="com.exalead.ndoc.v10" value="C:\Users\E7G\Downloads\tra
    <Context xmlns="com.exalead.ndoc.v10" language="xx" name="extracted_m
    <ScoreContext xmlns="com.exalead.ndoc.v10" value=0/>,
    <Chunk xmlns="com.exalead.ndoc.v10" value="text/plain"/>,
    <Context xmlns="com.exalead.ndoc.v10" language="xx" name="mime"/>,
    <ScoreContext xmlns="com.exalead.ndoc.v10" value=0/>,
    <Chunk xmlns="com.exalead.ndoc.v10" value="text#plain"/>,
    <Context xmlns="com.exalead.ndoc.v10" language="xx" name="docsrc"/>,
    <ScoreContext xmlns="com.exalead.ndoc.v10" value=0/>,
    <Chunk xmlns="com.exalead.ndoc.v10" value="txt"/>,
    <Context xmlns="com.exalead.ndoc.v10" language="xx" name="text"/>,
    <ScoreContext xmlns="com.exalead.ndoc.v10" value=0/>,
    <Chunk xmlns="com.exalead.ndoc.v10" value="Rang,Reseau,Station,Traffic
c1,c2,c3,c4,Ville,Arrondissement1,M#tro,GARE DU NORD,&quot;48,146,629&quot;;4,5,0,
M#tro,SAINT-LAZARE,&quot;46,790,941&quot;;3,9,12,13,14,Paris,8"/>,
    </element>
  </Document>
</Document>
</DocumentProcessorsOutput>
</##default>
<##default>
  <UnmappedContexts xmlns="com.exalead.indexing.analysis.v10">
    <StringValue>
      [
        <StringValue xmlns="exa.bee" value="docsrc"/>,
        <StringValue xmlns="exa.bee" value="extracted_mime"/>,
        <StringValue xmlns="exa.bee" value="source"/>
      ]
    </StringValue>
  </UnmappedContexts>
</##default>
</TestAnalysisPipelineOutput>

```

Search for Issues in Semantic Processor

You can follow the steps below to identify semantic processor issues.

Step 1 - Display Semantic Processors for Each Document

To display detailed information on semantic processing in an HTML file, you must first add a **Debug Processor** element to your semantic processors list.

Important: The HTML output is verbose. You can use sample data to avoid using large amounts of disk space during indexing.

1. From the Administration Console, go to **Index > Data processing > Pipeline name**.

2. In the **Semantic Processors** tab, drag the **Debug Processor** to the end of the processors list.
3. In the **Input from** field, specify the HTML file in which information is logged.

Example: all semantic processing applied to field 327 of document [0000000031217F90]

DOCUMENT 1 [0000000031217F90] : http://localhost:10001/admin/sdk/java-clients/docs/api/com/exalead/datamodel/v10/DynamicProperty.html - FIELD 327 : excludedcontent

1	All	LOWERCASE all	NORMALIZE all	relatedTermsPreprocessor_stopWord	exalead.nlp.odonymtype Allée
2	■				
3	Implemented	LOWERCASE implemented	NORMALIZE implemented	relatedTermsPreprocessor_stopWord	
4	■				
5	Interfaces	LOWERCASE interfaces	NORMALIZE interfaces	relatedTermsPreprocessor_staticLemma interface	
6	:				

Step 2 - Submit Text or Document

1. Submit text through the semantic pipeline to display all running processors:

- Using a single word:

```
cvconsole cvdebug> semantic annotate language=en context=text value="WORD"
```

- Using a text block (.TXT file):

```
cvconsole cvdebug> semantic annotate-file language=en path=<PATH_TO_TXT_FILE>
```

Example: submit the word 'test'

```
cvconsole cvdebug> semantic annotate language=en context=text value="test"
```

The output displays tokens tagged with annotations:

```
<AnnotatedToken id="0" kind="TOKEN_SEP_PUNCT" lang="en" token="#Ç¥" offset="0" />
<AnnotatedToken id="1" kind="TOKEN_ALPHA" lang="en" token="test" offset="3" >
  <Annotation TID="7" kind="LOWERCASE" id="0" nbTokens="1" display="test" displayK
trustLevel="0" />
  <Annotation TID="8" kind="NORMALIZE" id="1" nbTokens="1" display="test" displayK
trustLevel="0" />
  <Annotation TID="20" kind="phonetic" id="2" nbTokens="1" display="T.E.S.T" displ
trustLevel="100" />
  <Annotation TID="23" kind="relatedTermsPreprocessor_staticLemma" id="3" nbTokens
displayKind="norm" trustLevel="100" />
</AnnotatedToken>
<AnnotatedToken id="2" kind="TOKEN_SEP_PUNCT" lang="en" token="#Ç¥" offset="7" />
<AnnotatedToken token="ö" kind="PUNCT" lang="en" offset="0" >
</AnnotatedToken>
<AnnotatedToken token="test" kind="ALPHA" lang="en" offset="1">
  <Annotation displayForm="test" displayKind="lowercase" tag="LOWERCASE" nbTokens=
  <Annotation displayForm="test" displayKind="normalized" tag="NORMALIZE" nbTokens
  <Annotation displayForm="T.E.S.T" displayKind="normalized" tag="phonetic" nbToke
trustLevel="100" />
  <Annotation displayForm="test" displayKind="normalized" tag="relatedTermsPreproc
```

```
nbTokens="1" trustLevel="100" />
</AnnotatedToken>
<AnnotatedToken token="ö" kind="PUNCT" lang="en " offset="5 ">
</AnnotatedToken>
```

Step 3 - Display All Processors

1. Display the list of semantic processors in the analysis pipeline using `cvdebug`:

```
cvconsole cvdebug> semantic dump-pipe
```

The output (extract) displays each semantic processor and resources:

```
<Processor>
[
  <Normalizer xmlns="com.exalead.mot.components" resource="exalead.891136526.normalizer"
  trustLevel="0"
  normalizeCJ="true"/>,
  <GermanTokenizer xmlns="com.exalead.mot.components"
  normalizerResource="exalead.891136526.normalizer.resource"
  splitPolicy="keepToken3" useCustomNormalizationTID="true" stickTokens="false"
  resource="exalead.subtokenizer.de.resource"/>,
  <DutchTokenizer xmlns="com.exalead.mot.components"
  normalizerResource="exalead.891136526.normalizer.resource"
  splitPolicy="keepToken3" useCustomNormalizationTID="true" stickTokens="false"
  resource="exalead.subtokenizer.nl.resource"/>,
  <NorwegianTokenizer xmlns="com.exalead.mot.components"
  normalizerResource="exalead.891136526.normalizer.resource"
  splitPolicy="keepToken3" useCustomNormalizationTID="true" stickTokens="false"
  resource="exalead.subtokenizer.no.resource"/>,
  <JapaneseCharDetector xmlns="com.exalead.mot.components"/>,
  <CJKProcessor xmlns="com.exalead.mot.components"/>
]
</Processor>
```

Analyzing User Queries with Reporters

CloudView contains several reporters allowing you to collect reporting information related to the behavior of your front-end applications.

For example, reporters can retrieve information related to the execution time or the CPU time of the search service, the suggest service, the mashup components of your applications (page, feed, widget, and trigger), etc. Reporters are therefore useful to analyze and troubleshoot performance issues.

Once you have selected a reporter, you can choose between different output formats to export collected data. To do so, you can select one or several publishers: CSV, JDBC, Reporting Store, PAPI.

Once data is exported to CSV, JDBC or Reporting Store, you can build reports out of it using external reporting tools.

About Reporters

Output Reporting Data to CSV Files

Output Reporting Data to a JDBC Database

Output Reporting Data to the Internal SQLite Database

Index Reporting Data as a Data Source

Available Fields for the Reporting Publishers

About Reporters

This section describes the use of the reporters delivered by default when you install Exalead CloudView.

In the Administration Console, if you select the **Search > Reporting** menu:

- All of them have a Reporting Store Publisher,
- the search-reporting and suggest-reporting reporters also have CSV publishers.

What Can You Do?

You can:

- Choose to add more publishers to the default reporters to output data in different formats as explained in the following "Output data to..." sections.
- Add and configure your own custom reporter if the default ones do not cover your needs.

What Can Reporters Do?

Reporter	Description
search-reporting	Collects user query data submitted to the /search-api/ command of the Search API.
suggest-reporting	Collects query data submitted to the Suggest command of the Search API.
mashup-ui-reporting	Collects data relative to task execution and to CPU activity on the Mashup UI. For example, when a user queries a page, the reporter retrieves data such as the execution and CPU time of pages, widgets, and triggers.

Reporter	Description
	<p>Once configured in the Administration Console, this reporter must be enabled in the Mashup Builder > Application > Application Properties menu.</p> <p>Note: When you enable the mashup-ui-reporting reporter and the Mashup UI debug mode, you can open a Timeline tab in your Mashup UI application. This tab shows a set of reporting fields with bars representing either real or CPU time values.</p>
mashup-api-reporting	<p>Collects data relative to feeds, subfeeds, and triggers execution. This reporter allows you to understand explicitly the feed execution process, with subfeeds and triggers and to identify possible problematic issues.</p> <p>Once configured in the Administration Console, this reporter must be enabled in the Mashup Builder > Application > API Properties menu.</p> <p>Note: When you enable the mashup-api-reporting reporter and the Mashup UI debug mode, you can open a Timeline tab in your Mashup UI application. This tab shows a set of reporting fields with bars representing either real or CPU time values.</p>
opendocs-reporting	<p>Collects data relative to the download and preview of documents on your application pages. In other words, what happens when people click the Download and the Preview links of the Result List widget.</p> <p>Once collected internally, data is displayed in the Business Console's Query Reporting > Top Opened Documents tab.</p>

Output Reporting Data to CSV Files

Exalead CloudView can output reporting data to CSV files.

For all reporters, you can add or remove reporting fields, and adjust the rotation frequency for the output.

Note: The CSV publisher is configured by default for the search-reporting and suggest-reporting services. Reporting fields are already selected but you can edit this default configuration.

Configure the CSV Output (Optional)

1. In the Administration Console, go to **Search > Reporting**.
2. Under **Reporter**, select the **CSV Publisher**.

- a. For **Published fields**, add or remove fields.
- b. Modify the other options if required.

Option	Description
Output to File	Allows you to specify a name for the CSV file. The default names are <code>suggest.csv</code> and <code>search.csv</code> for the default suggest-reporting and search-reporting reporters. You can specify other file names if required.
Max file size (MB)	Allows you to specify the maximum size allowed for the CSV file. If you crossed the max size, the rotation is launched automatically.
Rotate every N months/days/ hours	Allows you to specify when to write the data to a new CSV file. The previous CSV files remain on the server.
Max files to keep	Maximum number of reporting files to keep. The oldest files are discarded at rotation time. 0 means that no limit is enforced, whereas 1 discards all rotated files.
Max days to keep	Maximum file age in days to keep. The oldest files are discarded at rotation time. 0 means that no limit is enforced, whereas 1 only keep today's files.
Max size to keep (MB)	Maximum size allowed for CSV files. The oldest files are discarded at rotation time. 0 means that no limit is enforced.

3. Click **Apply**.

Access CSV Query Data

1. Go to your `<DATADIR>/run/searchserver-ss0/` directory:
2. Then for:
 - **search-reporting:** `/search-reporting/search.csv`
 - **suggest-reporting:** `/suggest-reporting/suggest.csv`
 - **mashup-ui-reporting:** `/mashup-ui-reporting/<FILENAME>.csv`
 - **mashup-api-reporting:** `/mashup-api-reporting/<FILENAME>.csv`
 - **opendocs-reporting:** `/opendocs/<FILENAME>.csv`

Output Reporting Data to a JDBC Database

You can create your own JDBC database to store search or suggest query data, then set up the required connection details and fields to output in Exalead CloudView. For all reporters, you can configure the export to only include a subset of these fields. You can also specify additional fields. For details, see the procedures below.

Create the JDBC Database

1. Create a JDBC database:
 - a. Create a dedicated table for each reporter. A table to store your search query data, a separate table to store suggest query data, etc.
 - b. Define the fields you want to report on.
2. Copy your JDBC driver to `<DATADIR>/javabin`.
3. On the Administration Console **Home** page, restart the `searchserver` and `connector` processes.
4. Add the JDBC reporting publisher. See [Add a JDBC Reporting Publisher](#).

Add a JDBC Reporting Publisher

1. Follow the steps in [Create the JDBC Database](#).
2. In the Administration Console, go to **Search > Reporting**.
3. Expand the configuration for the reporter to export. For example, for search queries, click **search-api**, for suggest queries, click **suggest-reporting**, etc.
4. Click **Add reporting publisher**.
 - a. Select **JDBC Publisher**.
 - b. Click **Accept**.
5. Click the newly added reporting publisher to display its configuration settings, and:
 - a. For **Published fields**, select the fields to include.
 - b. Specify the connection details to the database you created in the previous procedure:

Option	action
Driver	Enter the JDBC driver class name. For MySQL, it is <code>com.mysql.jdbc.Driver</code> .
Connection string	Enter the JDBC URL of the database.
Login	Enter your database login, if any.
Password	Enter your database password, if any.

Option	action
Table	Enter the name of the database table you want to report on.

- Click **Apply**.

Output Reporting Data to the Internal SQLite Database

You can use the Reporting Store Publisher to send data to the embedded CloudView SQLite database.

The Reporting Store Publisher is the default publisher used by all reporters. You can view reporting data for the:

- search-reporting in the Business Console's **Query Reporting** menu.
- mashup-ui-reporting in the **Mashup UI Debug Mode > Timeline** tab.
- mashup-api-reporting, in the **Mashup UI Debug Mode > Timeline** tab.
- opendocs-reporting in the Business Console's **Query Reporting > Top Opened Documents**.

Output Data to the Reporting Store Publisher

- In the Administration Console, go to **Search > Reporting**.
- Under **Reporter**, expand one of the reporters, and select **Reporting Store Publisher**.
- For **Schema**, select one of the predefined schemas of the embedded SQLite database. For example, for the:
 - search-reporting** reporter, select **queries**,
 - suggest-reporting** reporter, select **suggests**,
 - etc.
- For **Rotation cron**, enter a cron command to run the rotation job periodically. By default, `0 * * * *` runs once a day at midnight. A rotation is also triggered every time a collection is queried.
- For **Max records to keep**, enter the maximum number of records that can be accumulated. When you reach the limit, the oldest records are discarded. 0 means that there is no limit to the database size.
- Click **Save**.

Access Reporting Store Data

- Go to `<DATADIR>/reporting_store/<SELECTED_SCHEMA>/collection.db`

Index Reporting Data as a Data Source

Using a PAPI Publisher allows you to use reporting data as a CloudView data source. You are then able to index this reporting data and use the Mashup Builder widgets (charts, tables, etc.), to create your own graphical reports.

Output Data to the PAPI Publisher

1. In the Administration Console, go to **Search > Reporting**.
2. Under **Reporter**, expand one of the reporters, for example, **search-reporting**.
3. Click **Add reporting publisher**.
 - a. Select **PAPI Publisher**.
 - b. Click **Accept**.
4. Click the newly added **PAPI Publisher** to display its configuration settings, and:
 - a. For **Connector name**, select the **Push API (unmanaged)** connector.
 - b. For **Host**, enter the connector hostname.
 - c. For **Port**, enter the Push API port number `<BASEPORT> + 2`.
5. Click **Apply**.

Index Data Collected by the PAPI Publisher

1. Go to **Data Model > Classes**, and select **Trace all metas** to retrieve the reporting fields.
2. Perform a query in your front-end application.
3. In the Administration Console, go to the **Home** page, you see that the default connector is working as it processes the search data.
4. To add the reporting fields as configurable properties in the Data model:
 - a. Go back to **Data Model > Classes**.
 - b. Click **Add class** to create a new class for your reporting properties.
 - c. Click **Add properties from traced metas** and define how reporting fields must be indexed.
5. Click **Apply**.

Once done, you can search for reporting data.

Available Fields for the Reporting Publishers

By default, all the fields listed in the following tables are exported. However, for the CSV publisher, you can configure the export to only include a subset of these published fields.

Search Reporting Fields

search-reporting Query Fields

Field Name	Type	Description
timestamp	datetime	The date and time of the export of the data.
apiclient_ip	string	IP address of the client for this API request.
query_logic	string	Search logic used for the query.
query_target	string	Search target used for the query.
query_querystring	string	The UQL query (q=) entered by the user. This is the same as the <code>_default_</code> value for the query template defined in <code>searchLogicList.xml</code> .
query_language	string	ISO language code
query_start	unsigned integer	First requested full hit.
query_hf	unsigned integer	Number of requested full hits.
query_origin	string	Explains "what" created this request: page load on Mashup UI; AJAX load on Mashup UI; trusted queries; cache warm-up; isAlive; alerting; and so forth.
answer_nmatches	unsigned integer	Total number of matches.
answer_nhits	unsigned integer	Number of hits.
time_total	unsigned integer	Total query time in microseconds.
query_full	string	Full query parameters in URL form.
query_id	unsigned integer	Auto-assigned internal query ID.
spellcheck_enabled	Boolean	Was spellcheck enabled on this query?

Field Name	Type	Description
spellcheck_suggestions	unsigned integer	Number of spellcheck suggestions.
spellcheck_autocorrect	Boolean	Was autocorrect enabled?
spellcheck_autocorrected	Boolean	Was autocorrect triggered?
applicationId	string	Mashup application ID passed by the API client.
user_id	string	User ID passed by the API client.
usersession_id	string	Session ID passed by the API client.
userquery_id	string	Query ID passed by the API client.
processing_indexquery	string	ELLQL query
answer_status	unsigned integer	Answer status. 0=ok, 1=error, 2=timeout, 3=limit reached
time_queue	unsigned integer	Time in query processing queue in microseconds.
time_queryprocessing	unsigned integer	Time for query parsing and processing in microseconds.
time_exec	unsigned integer	Time for partial hits execution in microseconds.
time_synfh	unsigned integer	Time for synthesis and full hits execution in microseconds.
cputime_queryprocessing	unsigned integer	CPU time for query parsing and processing in microseconds.
cputime_exec_searcher	unsigned integer	CPU time for partial hits execution, searcher side, in microseconds.
cputime_exec_index	unsigned integer	CPU time for partial hits execution, index side, in microseconds.
cputime_synthesis_searcher	unsigned integer	CPU time for synthesis execution, searcher side, in microseconds.
cputime_synthesis_index	unsigned integer	CPU time for synthesis execution, index side, in microseconds.

Field Name	Type	Description
cputime_fullhits_searcher	unsigned integer	CPU time for full hits execution, searcher side, in microseconds.
cputime_fullhits_index	unsigned integer	CPU time for full hits execution, index side, in microseconds.
searchserver	string	The search server that processed this query.
expansion_languages	string	Language detected at search-time for the expansion.

Suggest Reporting Fields

suggest-reporting Fields

Field name	Type	Description
timestamp	datetime	The date and time of the export of the data.
apiclient_ip	string	IP address of the client for this API request.
query_service	string	Name of the suggests or dispatchers called.
query_querystring	string	The UQL query (q=) entered by the user. This is the same as the <code>_default_</code> value for the query template defined in <code>searchLogicList.xml</code> .
query_output	string	output format: JSON or XML.
query_full	string	Full query parameters in URL form.
answer_status	Boolean	0 = OK, 1 = error
answer_nhits	unsigned integer	Number of suggestions returned.
answer_blacklisted	unsigned integer	Number of removed suggestions.
time_total	unsigned integer	Total query time in microseconds.
query_distance	unsigned integer	Approximate matching. The greater the distance, the more approximate the match. 0 for exact match.
query_cursor_pos	unsigned integer	The cursor position.
query_recursive	Boolean	Was the query processed recursively?
query_autocomplete	Boolean	Was the original query auto-completed?

Field name	Type	Description
query_min_d1	unsigned integer	If distance >= 1: minimum entry length to perform approximative suggestions with distance set to 1.
query_min_d2	unsigned integer	If distance >= 2: minimum entry length to perform approximative suggestions with distance set to 2.
query_logic	string	Search logic used for the query.
query_callback	string	The javascript callback that was called.
query_exhaustive	Boolean	Did the output contain exhaustive information?
searchserver	string	The search server that processed this query.

Mashup UI & Mashup API Reporting Fields

These are the fields that can be applied for both mashup-ui-reporting and mashup-api-reporting.

mashup-ui-reporting & mashup-api-reporting Fields

Field name	Type	Description
timestamp	datetime	The date and time of the export of the data.
user_id	string	User ID passed by the API client.
application_id	unsigned integer	Auto-assigned internal application ID.
report_id	unsigned integer	Auto-assigned internal report ID.
component_type	string	Indicates the reported component types. For example, Page, PreRequestTrigger, Widget, MashupWidgetTrigger, etc.
component_name	string	Indicates the names of all the components that can be found on the page, that is to say the name of the page itself, the widget names and the trigger names.
component_event	string	Indicates the event types. For example, render, before_query, after_query, before_rendering, after_rendering, etc.
start_cpu	unsigned integer	CPU start time value in microseconds for each page component event.
stop_cpu	unsigned integer	CPU stop time value in microseconds for each page component event.

Field name	Type	Description
start_nanotime	unsigned integer	Real start execution-time value in nanoseconds for each page component event.
stop_nanotime	unsigned integer	Real stop execution-time value in nanoseconds for each page component event.
service_instance	string	Mashup UI or Mashup API instance name (as specified in Deployment > Roles). For example, mu0 for Mashup UI and ac0 for Mashup API.
user_session	string	Auto-assigned internal user session ID.
query_querystring	string	The full query string received by the page or the Mashup API.
client_ip	string	The web client (browser) IP address.
client_user_agent (for mashup-ui-reporting only)	string	The web client (browser) user agent.
client_accept_language (for mashup-ui-reporting only)	string	The web client (browser) default language.
response_size	unsigned integer	The web client (browser) or the Mashup API response size in bytes.

opendocs Reporting Fields

opendocs-reporting Fields

Field name	Type	Description
timestamp	datetime	The date and time of the export of the data.
document_source	string	The connector name.
document_uri	string	The document URI that was downloaded or previewed.
document_filename	string	The document file name that was downloaded or previewed.

Field name	Type	Description
user_id	string	User ID passed by the API client.
application_id	string	The name of the mashup UI application on which documents were downloaded or previewed.
query_querystring	string	The UQL query (q=) entered by the user. This is the same as the <code>_default_</code> value for the query template defined in <code>searchLogicList.xml</code> .
query_queryfull	string	Full query parameters in URL form.
buildgroup	string	The name of the build group in which the document is indexed.
type	string	The type of action that was executed: Download or Preview.

Performance Considerations

This chapter describes what you need to understand before indexing a full corpus.

[About Exalead CloudView Sizing](#)

[The Impact of the Data Model on Performance](#)

[Dealing with Hierarchical Dimensions](#)

About Exalead CloudView Sizing

While developing your search application, you most likely used a small corpus to test indexing. Your focus was on getting relevant results that display according to the specifications for the application.

Once you start the testing phase of your application, indexing with a real corpus, your focus turns to performance, and the sizing you need to support it.

Sizing is a complex topic with so many variables that it is impossible to provide hard & fast rules.

We can, however, explain the main sizing considerations in a project, and the type of resource they impact.

Important: This section does not replace an Exalead professional services engagement. Sizing is complex, with many factors to consider. Consult an Exalead sizing expert before undertaking sizing.

How Project Requirements Impact Sizing

This requirement	Impacts
Project description	
<ul style="list-style-type: none">Do you need exhaustive search? (It guarantees that it searches and retrieves every match, even if it takes longer).	CPU
<ul style="list-style-type: none">What is an acceptable response time? Typically, for dashboards it is 4 or 5 sec; for intranet search it is in milliseconds.	CPU & RAM
Scope: what is the expected lifecycle for this app? Can this app be used for further projects?	Number of machines, or type of hardware

This requirement	Impacts
How many documents can be indexed?	Mainly disk space. May also impact RAM & CPU
How many data sources, and what kind?	CPU & RAM
How many users + what is the estimated QPS (Queries Per Second)?	CPU & RAM
Index freshness - every minute/every hour/every day?	CPU, RAM & disk performance
What is the expected throughput (how many incremental changes to your corpus)?	CPU, RAM & disk performance
Does it need to be an HA deployment?	Number of machines

See also the "Before going live" in the Exalead CloudView Administration Guide for a list of issues to be aware of when going into production.

Disk Requirements

See the General System Requirements in the Exalead CloudView Installation Guide.

RAM Sizing Formula

You can estimate your total RAM requirements like this:

RAM for processes + RAM for RAM-based fields + RAM for document cache.

The table below explains this in more detail.

Memory	is used for	and requires
RAM for processes	running Exalead CloudView processes.	Fixed at 8 GB
RAM for RAM-based fields	storing metas used for faceting, sorting, grouping and in virtual expressions in memory.	Highly variable: $2 * (\text{avg size of all RAM-based meta} * \# \text{ docs})$
RAM for document cache	caching indexed documents in memory.	20% of the size of your index. To estimate index size <ul style="list-style-type: none"> Enterprise search: $20 \text{ kb} * \# \text{ docs}$

Memory	is used for	and requires
		<ul style="list-style-type: none"> Analytical dashboards or eCommerce: $3 \text{ kb} * \# \text{ docs}$

To put all this together:

- For enterprise search

$$\text{Total RAM} = 8 \text{ GB} + 2 (\text{avg size of all RAM-based meta} * \# \text{ docs}) + 0.2(20 \text{ kb} * \# \text{ docs})$$

- For search-based applications (analytical dashboards) or eCommerce

$$\text{Total RAM} = 8 \text{ GB} + 2 (\text{avg size of all RAM-based meta} * \# \text{ docs}) + 0.2(3 \text{ kb} * \# \text{ docs})$$

To see the RAM usage per field, use the `cvdebug` command-line tool located in `<DATADIR>/bin` and start the following command:

```
cvdebug index dump-attribute-group-column-infos
```

The Impact of the Data Model on Performance

The data model serves as a way to group your metas according to business logic using classes. Each property has options that determine how to store the corresponding meta in the index and access it by the application.

You need to keep in mind, though, the impact of:

- selecting unnecessary options for properties
- having too many classes

How Property Options Impact Performance

These are the options we are talking about by 'data model property options'.

attachment *metadata (Alphanum)*

☒ Dedicated field ☐ Dynamic field ☐ Category facet

Data type: Alphanum

Semantic type: metadata

Default value:

▼ Other advanced options

☒ Enumerated

Enable pattern search ☐

Multivalued ☐

Additional meta names:

Compress content ☒

Searchable with prefix ☐

Searchable without prefix ☐

Retrievable ☒

RAM based ☐

► Expansion control

The table below explains what they do and how they impact performance.

Data Model Options, Listed in Order of Performance Impact

Property option	What it does	Impacts	Explanation
Dedicated field	Creates an index field on disk.	Disk, and eventually search latency	The more index fields, the more folders on disk (where an index field = a folder), which in turns means more things to compact, and more things to replicate which in turn increases search latency.
Searchable with prefix	Creates the inverted list, which is the lookup structure used to respond to search queries.	RAM	This also adds folders/files to disk, and at run-time, even when there are no queries sent to it, it consumes some RAM. The files containing the inverted lists is mapped by the index. Map files can be loaded or unloaded in RAM by the OS; this explains why virtual memory in

Property option	What it does	Impacts	Explanation
			the monitoring console may display as higher than available memory.
Searchable without prefix	<p>Copies the meta values into the text field. Typically used for legacy reasons (in v5).</p> <p>By default, the text field is targeted by the text prefix handler, and means that if users enter a query without a prefix, the search targets this text field.</p> <p>If you change the default prefix handler, however, it renders this option useless.</p> <p>In that case, it is better to use the renameContext doc processor and rename your meta to the name of the field targeted by your default prefix handler.</p>	RAM Disk	<p>Virtually the same RAM consumption as for Searchable with prefix.</p> <p>Less disk consumption as values are all stored in the same field.</p>
Retrievable	<p>Copies the entire meta source in the attributes structure.</p> <p>This is the structure used for returning the meta values, whether this appears in search results, facets, or for sorting.</p> <p>Note: If you select this option without selecting "index field", the meta values are stored in the "metas" default attribute.</p>	<p>For search only: Disk.</p> <p>For faceting and search: RAM, since these must be RAM-based.</p>	<p>For faceting and sorting, the property must be both retrievable and RAM-based.</p> <p>This is because we need to ask for this value very frequently when sorting and faceting for the entire result set.</p> <p>Conversely, metas used only for search result display must be stored as retrievable, which</p>

Property option	What it does	Impacts	Explanation
			means they are stored on disk.
RAM-based	<p>This only displays if "retrievable" and "index field" are selected.</p> <p>Important: Only use RAM-based for fields required for sorting, grouping, facet aggregations, or search-time facets.</p>	RAM	<p>The more fields that are RAM-based, the more memory consumed.</p> <p>There is a fixed amount of RAM consumed for each document, regardless whether this property is present or not.</p> <p>Sometimes for numerical properties, the default 64-bit allocated for this property is too much; if you only have a small range of values, you can reduce this value and thereby reduce memory consumption.</p>
Category facet	<p>Selecting this means the original value of my meta is stored in the <code>category</code> field of the index.</p> <p>This was the only method to create facets in v5. However, in v6 there are very few reasons to use this because we can now create search-time facets for numerical, geographic, and date metas.</p> <p>Important: Only select this option when you want, for example, to facet on colors (red, blue, green, etc.), and</p>	RAM Faceting (synthesis) speed	<p>To process category facets efficiently, we have implemented a structure (<code>cdict</code>) to store these in RAM, which assigns an ID to each value.</p> <p>Having too many distinct values in a category field increases with ID range and impact memory consumption and the speed of faceting.</p>

Property option	What it does	Impacts	Explanation
	do NOT need to use full-text search options (wild cards, approximate). You can, however, use this to do exact search, for example color: red returns all documents with the value red.		
Enumerated (Value facet)	Only available on alphanumeric properties with the index field option selected.	RAM and CPU, but less than for category facets	Since it does not manage hierarchical values, it does not perform the calculations required for parent-child relationships, and so consumes less RAM and CPU than a category facet.

How Classes Impact Performance

In the data model, you can define classes to organize your properties according to a business logic. The name of the class appears in any element (index field, facet, hit meta) generated by the property. This allows applications to select only the elements relevant to their business logic.

There is a limitation to adding many different classes and properties, however: the index does not distinguish between class and properties.

This means in the index schema, they are flattened into one list. One document in the index has the properties for all classes, even if there are no values for those properties.

This multiplies the performance impact of the options described in [Table 12](#), for each “redundant” property associated with a document.

Dealing with Hierarchical Dimensions

When dealing with hierarchical dimensions, several use cases can be addressed in two ways. Either at search time, by configuring facets in the Search Logic, or at indexing time, by configuring options in Data Processing and Data Model.

Examples:

- For the search time solution, imagine you have 3 facets: People, Organization, and Role. If you want to be able to switch dimensions from `Organization / Role` to `Role / People` or `Role / Organization`, preparing all possible permutations would lead to important storage and complex field creation. In this case, it is simpler to choose how you want to present values at run time.
- If you have multiple metas, for example type, subtype, model and always use them in the same order, it might be interesting to merge these metas in a single one called type/subtype/model. It is still possible to point on a specific level of the tree, for example, if you want to get model values only. This solution is faster in terms of query latency, but you cannot change the order.

To make it simple:

- Choose the Search/Run time solution when you want flexibility, that is to say, avoid reindexing documents if you need to change the way dimensions have to be presented. This solution can yet lead to longer query latency compared to the indexing time solution when the amount of data is important.
- Choose the indexing time solution when the amount of data to process is important and you need to have low query latency.

Appendix - Configure Document Processors

This section describes how to use and configure document processors in the analysis pipeline.

Chunk Operations

Normalization

Numerical Operations

Text Extraction

Text Operations

Custom

Other

Chunk Operations

Copy Context Chunks

Copies all document chunks from the context specified in **Input from**, and creates new document chunks with the same score, language, and part, in the context specified in **Output to**.

You can apply matching conditions to this processor to refine its behavior. For example, if you have a multivalued field having the following values: `order-1`, `order-2` and `order-A`, and want to numeric orders only (that is, everything but not `order-A`), you can set a condition with a value that equals the `order-d+` regular expression.

Multi-Context Encoder

Creates a DocumentChunk containing the ContextName and the textual value of the DocumentChunks matching 'inputContexts'.

This processor can be used, for example, to store arbitrary (key, value) pairs into one single index field.

The serialization format is the following:

- `"ContextName1"="TextContent1"`
- `"ContextName2"="TextContent2"`
- ...

The double-quote character in name and value is escaped with a backslash.

Note: This storing method is inefficient and must be used with caution.

New Chunk

Creates a new DocumentChunk with 'outputContext' as ContextName, and textual content specified in 'value'.

Remove Contexts

Removes all DocumentChunks with a ContextName matching 'inputContexts'.

Rename Context for Chunks

Renames each DocumentChunk with ContextName matching 'inputContext' with a ContextName 'outputContext'.

Rename Unmapped Contexts

Changes the ContextName for all DocumentChunks associated with a ContextName that does not have a Mapping Configuration.

This avoids extensive renaming using RenameContext.

Replace Values

Compares all DocumentChunks for a given inputContext with the specified KeyValue map.

When the DocumentChunk value is an exact match, the specified string replaces it.

You can use this processor, for example, to normalize different spelling for document metadata.

NOTE: The specified KeyValue map must be an exact match with the complete DocumentChunk.

To replace only a substring of a DocumentChunk, use ReplaceRegexp.

Input: All DocumentChunks associated with the specified 'inputContext' ContextNames.

Output: Same as input.

Value Selector

Takes the input contexts in the specified order, and as soon as one is found, copies the content to the output context and stops.

Normalization

Date Formatter

If a document chunk matches either:

- A custom input format defined with UNIX date syntax, for example, `%Y/%m/%d %H:%M:%S` (see [Indexing Options for Date Properties](#))
- One of the automatically recognized date formats (see [Automatically Recognized Input Formats](#))

The Date Formatter generates three additional document chunks, each with its own context name, using the following naming convention:

- `$inputContext$dateTimeOutputContext` (default format: `%Y/%m/%d-%H:%M:%S`)
- `$inputContext$dateOutputContext` (default format: `%Y/%m/%d`)
- `$inputContext$timeOutputContext` (default format: `%H:%M:%S`)

Note: You can also define specific output formats in the config XML.

Automatically Recognized Input Formats

If no input format is specified, the Date Formatter automatically recognizes dates in the following formats.

Note: Timezones are ignored.

- RFC 822 and 2822
- ISO 8601 and RFC 3339
- Other ('day' and 'month' values are only recognized if written in English):
 - "day, DD month YYYY hh:mm:ss"
 - "day, DD month YYYY hh:mm:ss timezone"
 - "day, DD month YY hh:mm:ss"
 - "day, DD month YY hh:mm:ss timezone"
 - "day month DD YY hh:mm:ss"
 - "day month DD hh:mm:ss timezone YYYY"
 - "DD month YYYY hh:mm:ss"

- "YYYY/MM/DD hh:mm:ss"
- "YYYY/MM/DD-hh:mm:ss"
- "MM/DD/YYYY hh:mm:ss"
- "MM-DD-YYYY hh:mm:ss"
- "MM/DD/YYYY"
- "MM-DD-YYYY"

Output Formats

The default output formats are:

- **date-time:** %m/%d/%Y %H:%M:%S
- **date:** %m/%d/%Y
- **time:** %H:%M:%S

Note: You can also define specific output formats in the XML configuration, using UNIX date syntax.

Numerical Formatter

Creates valid numerical chunks from various number formats.

Public URL Processor

Creates 4 DocumentChunks, each associated with a different ContextName, for each input DocumentChunk associated with the ContextName 'inputContext':

- 'treeOutputContext'
- 'leafOutputContext'
- 'urlOutputContext'
- 'urlCategoryOutputContext'

Units of Measurement Normalizer

Detects the unit symbol if specified in the input value and operates a conversion according to the index unit symbol when required. Then creates a new meta-data with the normalized value.

You must define the following properties:

- **Input from** – Specify the name of your data model measurement property.

- **Measurement type name** – Specify the measurement type defined in your data model measurement property.
- **Unit Symbol** – Specify the Unit symbol to use in the output value.
- **Suffix to add to the meta_name** – (default "_um") suffix name to add to the meta name to create output meta value.

For example, if you have a **volume** index field with a measurement type set to `volume` and a unit symbol set to `millimeter`:

Let us say that Document1 has a **volume** meta-data containing the value `50c1` and Document2 has a **volume** meta-data containing the value `1000`. Then the output for Document1 is a new `volume_um` meta containing the value `500` and for Document2 a `volume_um` meta with the value `1000`.

Numerical Operations

Double to Long

Stores floating point values into signed fields that can then be queried with the `DoublePrefixHandler`.

Fixed Range Numerical Partitioning

Matches numerical values in a range. It transforms a numerical value into a matching range, based on a fixed range size.

For example, with `rangeSize = 100`,

- `101 -> 100_199`
- `234 -> 200_299`

For negative numbers:

- `-20 -> -100_-1`
- `0 -> 0_99`

This helps to create categories (for navigation) from numerical values.

Forced Range Numerical Partitioning

Transforms a numerical value into the text value associated to its matching range from a set of predetermined ranges specified in 'NumericalRange'.

Math Document Processor

Performs mathematical operations on a numerical field. You must preface expressions with a \$.

For example, the expression ``$ht_price * 1.196`` finds the first chunk in the ``ht_price`` context, and replaces all occurrences of ``ht_price`` with the mathematical expression.

The result is a new text chunk, either in the Output context (if specified), or in the original ``ht_price`` context.

Text to Num

Processor to hack an approximate sort on a text field.

Implements a surjection from the set of strings to the set of integers $[0..N]$ with N close but inferior or equal to 18,446,744,073,709,551,615.

You define an ordered alphabet. A first surjection from the set of all strings to the set of finite sequences of symbols taken from this alphabet is applied (strip the string from symbols out of the alphabet).

A partial order relation is inferred on the latter set by the alphabet (lexicographical order).

For obvious cardinal numbers reasons (one set is infinite the other is not), the second surjection cannot be partial-order preserving. The idea is to preserve the relation on the shorter strings, and preserve the relation between shorter strings and longer strings, such as:

- if `STRING2ULONG('shortstring1') <= STRING2ULONG('shortstring2')` then `'shortstring1' <= 'shortstring2'`
- `STRING2ULONG('longstring1') <= STRING2ULONG('longstring2')` does NOT insure `'longstring1' <= 'longstring2'`
- if `STRING2ULONG('shortstring1') <= STRING2ULONG('longstring2')` then `'shortstring1' <= 'longstring2'`

The size of the prefix obviously depends on the size of the alphabet.

Text Extraction

HTML Relevant Content Extractor

Extracts the most relevant parts of an HTML document.

Generally, the relevant part of an HTML document is the article on the middle of the page. The header, the footer and the menus are often the same on all pages and should not be indexed.

The extraction can be tuned using different attributes (see below).

An annotation `readability:concat` is added (on the 1st chunk) when several relevant chunks must be processed together.

Internally, the `HTMLRelevantContentExtractor` assigns a score to each chunk of its input, and keeps only chunks having a score greater than "minScore".

This score is based on different weighting factors:

- word length
- paragraph length
- HTML tags
- CSS classes
- word/hyperlink ratio

It copies relevant HTML chunks to a new context defined by 'relevantChunkContext' and either:

- Copies irrelevant chunks to another context defined by `irrelevantChunkContext` (default).
- Annotates irrelevant chunks with an annotation defined by `irrelevantChunkAnnotation`.

By default, irrelevant chunks are indexed as text with a lower score. Remove the 'excludedcontent' index mapping if you do not want to index them.

Chunks from other types of documents (non-HTML documents) are all copied to the relevant context.

No more 'text' chunks are available after this processor. Use 'htmlcontent' if you want to manipulate the original text.

MIME Detector

Operates on each `DocumentPart` for which a MIME-type is not available.

For each `DocumentPart` in the PAPI, you can specify the MIME-type.

For `DocumentPart`, the 'bytes' and the 'filename' are used to guess the real MIME-type and charset.

The guessed MIME-type and the charset are then set as attributes of the `DocumentPart`.

Input: The `DocumentPart` of the document.

Output: 'mime' and 'encodingToUse' attributes of DocumentParts.

This document processor does not create any document chunks.

Mime Type Setter

Manually specify the mime type.

Semantic Web Document Processor

Extracts microdata/format from indexed documents.

Standard Parts Merger

This processor needs one DocumentPart called the 'Master Part'. If there is only one part, this part is the 'Master Part'.

If there are multiple parts, the part named after the 'masterPart' attribute is the 'Master Part'.

- The DocumentChunks from the 'Master Part' are removed if there is a root DocumentChunk (that is, not within any part) with the same ContextName except for those whose ContextName appears in 'partSpecificContexts'. For example, if there is a DocumentChunk with ContextName 'title' in the root document, any DocumentChunk with ContextName 'title' in the Master Part is deleted. This behavior allows you to keep the meta-data extracted from the master part, unless overridden through a global 'meta' in the PAPI.
- The DocumentChunks from the other DocumentParts are deleted, except for those whose ContextName appears in 'partSpecificContexts'. This avoids the index fields being polluted by additional DocumentPart's metadata. For example, when processing an Email with attachment, we keep the date of the email as a Date, rather than any date extracted as a metadata of any attachment.

Input: All DocumentChunks' from parts.

Output: This processor does not create any new DocumentChunk. It deletes existing DocumentChunks.

Text Extractor (All Mime Types)

Performs text content extraction for all MIME-types (300+ file formats are handled).

Text, HTML, and built-in data types must be processed by the 'NativeTextExtractor'.

Make sure to have a '**NativeTextExtractor**' before the ConvertTextExtractor in your pipeline.

Input: Document Binary Part.

Output: One or more DocumentChunks are created for each part, using the text and metadata of the binary content. Each DocumentChunk is associated with a ContextName. There can be one or more ContextNames depending on the extraction process. The ContextNames created depend on each mime-type.

Text Extractor (text, html, exalead)

Extraction is performed for the following data types:

- text/plain for Text files.
- text/html for HTML Files.
- application/x-exalead-document for CloudView 4.6 document format (com.exalead.document)
- application/x-exalead-ndoc for CloudView 5 internal document format, binary.
- application/x-exalead-ndoc-v10+xml for CloudView internal document format, XML.

Input: Each DocumentPart. The 'mimeType' and 'charset' of the DocumentPart are used by the extractor. Only the DocumentParts for which the mime-type matches the specified list are processed.

Output: Creates one or more chunks for each part, using the text and metadata for the binary content. Each DocumentChunk is associated with a ContextName.

There can be one or more ContextNames depending on the extraction process.

The 'bytes' of each Document Binary Part are deleted.

Xpath Extractor

Extraction is performed for the following data types:

- text/html. HTML Files.
- application/xml. XML Files.

Input: Each DocumentPart. The 'mimeType' and 'charset' of the DocumentPart are used by the extractor. Only DocumentParts for which mime-type matches the specified list are processed.

Output: DocumentChunks are created for each Xpath Rule. Each DocumentChunk is associated with the specified 'Meta name' ContextName.

Note: The output is not processed through the TextExtractors, if you want only the textual content, you must add `//text()` to your rule.

Warning: Must be set before the **NativeTextExtractor** because the 'bytes' of each Document Binary Part are deleted by the **NativeTextExtractor**.

Limitations: This extractor handles node set and string functions, not number and Boolean. You can use number or Boolean functions inside your xpath `<code>//img[starts-with(@src, "http://")]</code>` because this xpath return a set of nodes (`<code></code>`) but xpath `<code>count(//img)</code>` does not work because it returns a number.

Xpath Fragment Extractor

Input: All DocumentChunks associated with the specified 'inputContext' ContextNames. Input can be XML or HTML fragment.

Output: DocumentChunks are created for each Xpath Fragment Rule. Each DocumentChunk is associated with the specified 'Meta name' ContextName.

Warning: To put before the **NativeTextExtractor** because the 'bytes' of each Document Binary Part are deleted by the **NativeTextExtractor**.

Text Operations

Concatenate Values

Concatenates the textual content of DocumentChunks which context names match what is specified in **Input from**, and joins them with the 'join' string.

A single DocumentChunk with the 'outputContext' context name is created as output.

Content Cleanup

Analyzes each DocumentChunk and performs white space removal. The Unicode specification defines 'white spaces'.

This includes ' ' '\r' and '\n'

Input: All DocumentChunks associated with the context names specified as input.

Output: Same as input.

Language Detector

Language detection uses the text of all DocumentChunks associated to the context names specified as input, for which language was not already detected or specified.

The whole text of all these DocumentChunks is taken into account by a statistical algorithm that detects the language. This language is then set as the language for all specified chunks.

For example, the language attribute of a DocumentChunk is used by semantic processing.

Language is represented by its iso639-1 code: `fr`, `en`.

Language Setter

The language is set as the language for all the DocumentChunks associated with the context names specified as input.

For example, the language attribute of a DocumentChunk is used by semantic processing.

The language is represented by its iso639-1 code: `fr`, `en`

Print Values

Prints textual content of DocumentChunks according to a formatting string.

This string contains variables in one of the 3 following formats:

- `$(name)`, the name of a context: output is the textual content of this context.
- `$/name:regexp/`, the name of a context whose chunks must match the regexp: output is the piece of text that has matched.
- `$/name:regexp:format/`, the name of a context whose chunks must match the regexp: output is defined by a sed-like format referencing the regexp subexpressions.

Important: In the regexp and format parts, use a backslash to avoid colons and slashes.

For example: `$(firstname) $(lastname) : $/age:[0-9]+/ $/date:([0-9]{2}) ([0-9]{2}) ([0-9]{4}):day=\\1 month=\\2 year=\\3`

Important: The context used in this method cannot be produced by another processor. It must come from the connector.

Replace Regexp

Substitutes the content substring of all DocumentChunks having the ContextName 'inputContext', using:

- **Pattern** as the matching substring regular expression
- and **Replacement value**, which may have a sed-like output format using references to capture `\\0` through `\\9`.

A new DocumentChunk is created with the substitutions.

Split Values

Splits the textual content of all DocumentChunks containing the context name defined in **Input from** using the specified separator as a separator regular expression.

A new DocumentChunk is created for each segment, with 'outputContext' as the ContextName.

String Hash

Computes a signed hash of the textual input value on 32 bits.

For example, you can use this value in a field used for grouping.

String Transform

Applies textual transformations on chunks from several contexts:

- trims blanks at the beginning and at the end of chunks
- reduces sequences of blanks to only one
- changes text to uppercase/lowercase/normalized/capitalized

Outputs replace inputs.

Custom

Custom Document Processor

Allows you to plug in custom code packaged as a CVPlugin into the document processing pipeline.

For more information, see in the Exalead CloudView Programmer's Guide.

Java Document Processor

Takes Java code either inline or from a file, and executes it on-the-fly.

For production mode, it is best to use the packaging custom code as a Java Plugin (CVPlugin) and using the **Custom Document Processor** to call it. **Plug-ins** allow better packaging and source code maintenance.

For more information, see in the Exalead CloudView Programmer's Guide.

Remote HTTP Transformer

Posts part bytes to the remote HTTP service, and gets the entered resource as a result.

The remote service may return a Document.MIME_V10 document, or any other document that can later be processed in the pipeline.

If the remote service returns a non "OK" HTTP status (!= 200 error code), the corresponding error passes as a regular error.

The service may also advertise a file name, using the standard Content-Disposition's `filename` attribute.

Other

Debug Processor

Dumps all the DocumentChunks specified in **Input from** as standard output. Provides a log of the 'Analysis' process.

Discard Document Processor

DEPRECATED

It does not stop the processing of the document. To do so, add a custom document processor with the following code:

```
document.setProcessingFlag(Operation.DISCARD_AND_DELETE);
((AnalysisDocumentProcessingContext) context).stopProcessingAfterCurrentProcessor();
```

Document Processor Group

Contains a list of document processors, which are executed only if this group document processor condition matches. Avoids condition duplication or distinct pipelines creation when several processors share the same condition.

Format Checker Date

The Format Checker Date processor checks that the chunk matches either:

- A custom input format defined with UNIX date syntax (for example, `%Y/%m/%d-%H:%M:%S`).

- One of the automatically recognized date formats.

Infer File Extension

When the `file_extension` meta is not present, find the file extension based in the file name or the mime meta (if one of these two is present).

Insert Current Date

Adds the current date in an output context.

Precomputed Thumbnails Document Processor

Precomputes thumbnails of the first DocumentPart.

Random DocumentChunks Generator (Uniform Distribution)

Adds a new DocumentChunk for one document out of 'modulo' documents processed.

The textual content of the DocumentChunk is picked out of the list specified in **Values**, with a uniform distribution.

Random DocumentChunks Generator (Zipf Distribution)

Adds a new document chunk for one document out of 'modulo'.

The textual content of the document chunk is picked out of the list specified in **Values**, with a nonuniform discrete Zipf distribution.

Real-Time Alerting

Matches queries defined by end-users and alerts them as soon as possible when a new matching document is indexed.

Semantic Pipe

Instantiates a semantic pipe and creates chunks out of resulting annotations.

It helps instantiate classification processors, and perform document level operations from their output.

Similar String to Part Converter

Converts the signatures in a string format from a meta to a binary part.

Storage Service Document Processor

Queries the storage for any meta to attach to the document.

Multivalued pairs are pushed as multivalued metas.

For example:

- The storage key "nb_comment" is attached as "nb_comment" meta on the document.
- The storage key "tags[]" is attached as "tags" multivalued meta on the document.

UTF8 Checker

Checks that the text passing through is valid UTF-8.

Emits a warning with the document URI and the context name if input is malformed.

Optionally deletes invalid chunks.

Appendix - Configure Semantic Processors

This appendix describes how to use and configure semantic processors in the analysis pipeline.

About Semantic Processors

[Acronym Detector](#)

[Chunker](#)

[Compound Words Splitter](#)

[Fast Rules Matcher \(Rule-Based\)](#)

[Lemmatizer](#)

[Named Entities Matcher](#)

[NGram Extractor](#)

[Normalizer](#)

[Ontology Matcher \(Resource-Based\)](#)

[Phonetizer](#)

[Proximity](#)

[Related Terms](#)

[Rules Matcher \(Rule-Based\)](#)

[Semantic Extractor](#)

[Semantic Query Analysis](#)

[Snowball Stemmer](#)

[Part of Speech Tagger](#)

About Semantic Processors

This appendix assumes that you already have these semantic building blocks in place, and so focuses on how to manually configure semantic processors directly in the analysis pipeline.

Remind that the Analysis pipeline processing is made of several stages:

- The Document processing, performed using Document Processors. See [Appendix - Configure Document Processors](#).
- The Semantic processing stage, performed using the Semantic Processors described in this appendix.

- The mapping stage, consisting of mapping DocumentChunk and Semantic annotations to index fields.

It involves a list of Semantic Processors, which process each DocumentChunk of each document sequentially, except those for which Semantic Processing is disabled in the mapping.

The Semantic Processing stage segments text into 'tokens' and then processes text as a flow of tokens. Semantic annotations are produced for each token.

Acronym Detector

This processor detects acronyms like "U.N.", "I.M.F" or "F.I.F.O" and extracts them in lowercase: "un", "imf", and "fifo".

The annotation tag for the acronym detector is `acronym`.

Specify:

- a list of separators
- whether to index the normalized version of the annotation
- a comma-separated list of context names of the document chunks for which this processor is applied (optional)

Chunker

A chunker detects syntactical groups in a token stream. This processor has no interest in itself, and is mostly used as a preprocessing for advanced syntactical processing.

This processor has a high impact on performance.

It creates output annotations:

- `gadv` – for adverbial group
- `gadj` – for adjectival group
- `gnoun` – for noun group
- `gverb` – for verbal group
- `gprep` – for prepositional group

Compound Words Splitter

The Compound Words Splitter processor splits CamelCase, quiteCamelCase and underscored_case words into separate words.

Example

Input	Output
SearchServer	Search Server
simpleSearchServer	simple Search Server
simple_value	simple value

To allow searching for these words individually, you must use **Tokenize annotations** option. It creates tokens for each root word of the compound word. You need it to index values since annotations are not tokenized (same behavior as the spellchecker).

For example:

Input	Output
SearchServer	<ul style="list-style-type: none">• Search• Server
simpleSearchServer	<ul style="list-style-type: none">• simple• Search• Server
simple_value	<ul style="list-style-type: none">• simple• value

When to Use

The use cases where this processor is useful are manifold. Among others, we could use it for:

- Agglutinated data coming from a database. For example, agglutinated names like JohnSteed, EmmaPeel, JohnGambitt, etc.
- Source code to search for variables and class and function names. Searching is more convenient when these compound names are split into multiple words, for example, when you want the query `search` to retrieve a document containing `SearchServer`.

Note: If you need to index "real" compound words without uppercase and underscores (for example, wheelchair, editor-in-chief, etc.) use a standard tokenization. For more information, see [Customizing the Tokenization Config](#).

Dependencies

Add a **Normalizer** processor in the analysis pipeline if you do not want to index exact forms only, but also support lowercase and normalized forms for uncompound words.

Fast Rules Matcher (Rule-Based)

The Fast Rules Matcher processor provides faster rule processing than the Rules Matcher but with fewer options for defining rules.

One of the advantages of this processor is the efficient algorithm that searches documents literally on the fly, using little memory.

When to Use

Using this processor, you can define rules that support the following matches:

- simple Boolean operators `AND`, `OR`, and `NOT`
- proximity and location operators `NEAR`, `BEFORE`, `AFTER`, `SPLIT`, `BUTNOT`
- prefixes: `text:"foo" AND title:"bar"`
- different word forms, such as normalized, phonetic, and so forth.
- regular expressions: `title:/fo+/'`
- numerical operators: `file_size<10 AND text:"foo bar"`
- dates, with the same supported formats as the **Date Formatter** document processor (RFC 822, RFC 850, asctime, ISO 8601, YYYY/MM/DD-HH:MM:SS). Supported formats are:
 - YYYY/MM/DD-HH: MM:SS
 - YYYY/MM/DD-HH: MM
 - YYYY/MM/DD-HH
 - YYYY/MM/DD
 - YYYY/MM
 - YYYY

In the rules, dates must be enclosed within curly braces:

```
date:>={ 2015/02/02-00:15 }
```

Dependencies

If the matching rules for this processor depend on phonetic, stem, or lemma matching, you must add the corresponding processor above this one in the pipeline.

For example, if your rules require phonetic forms, place the `Phonetizer` processor above this processor in the analysis pipeline.

Rule Nodes

Configure the rules for your Fast Rules semantic processor in an XML file. The root node of the XML file is `FastRulesDefinition`. It contains the `catName` attribute and a set of rules for each value of category, `Category` node.

The `Category` node contains a `value` attribute and a set of rules, `Rule` node.

The `Rule` node contains the following attributes.

Fast Rules Matcher - Rule Node Attributes

Attribute	Description
<code>value</code>	A query. Only a subset of UQL is supported as outlined below.
<code>lang</code>	Restricts the query to documents in a specific language. If <code>lang</code> is <code>xx</code> , then apply to all documents.
<code>exceptionRule</code>	Is equivalent to <code>AND (NOT (value))</code> .

Sample Fast Rules XML Files

Example with Boolean operators

```
<FastRulesDefinition xmlns="exa:com.exalead.mot.components.fastrules" catName="MyCate
  <Category value="MachineLearning" >
    <Rule value="text:&quot;clustering&quot; AND (text:&quot;algorithm&quot; OR text:
OR text:&quot;learning&quot;)" exceptionRule="false" />
  </Category>
  <Category value="Hardware/Cluster" >
    <Rule value="text:&quot;clustering&quot; AND text:&quot;load balancing&quot;" exc
  </Category>
</FastRulesDefinition>
```

Supported Queries

A query specified as a `value`. Only the following subset of UQL is supported:

- `AND`, `OR`, `NOT`
- `BUTNOT`
 - For example, `New BUTNOT "New York"`.
 - Note this is different from `New AND (NOT "New York")`, which eliminates all documents containing "New York" from the search. By contrast, `New BUTNOT "New York"` still returns documents containing `New York` if they also contain the word `new` elsewhere in the document.
- `BEFORE[/N]`, `AFTER[/N]`, `NEAR[/N]`
 - `N` by default = 16
 - For example, `New BEFORE York` means returns documents where `New` occurs no more than 16 words (the default distance) before `York`.
 - `New BEFORE/4 York` returns documents where `New` occurs no more than 4 words before `York`.
- `A SPLIT B`, where `B` must be a terminal node, such as a string, a regular expression, or an annotation.

For example, to search a CSV file you could use `(A AND B) SPLIT ","` returns documents that contain occurrences of both `A` and `B` that are not separated by a comma.

Important: With `SPLIT`, the more complex the left-side expression, the greater the performance impact. In general, searching for regular expressions is more efficient than searching for text. However, when used with the `BUTNOT`, `BEFORE`, `AFTER`, `NEAR`, and `SPLIT` operators, regular expressions no longer have a performance advantage over text.

Rule Syntax

- `ContextName ':' "TextExpr\" |`
- `ContextName ':' "TextExpr\" "{"TextLevel"}" |`
- `ContextName ':' /RegularExpr/ |`
- `ContextName ':' /RegularExpr/ "{"TextLevel"}"; |`
- `ContextName ':' @Tag@`
- `ContextName ':' <comparison operator>42`

Where:

- `ContextName`: : the context name, or meta, where the expression or regexp must be applied, like `text:` or `title:`.
- `"TextExpr"`: any textual expression like `"database administrator"`
- `{TextLevel}`: the matching level, which can be:
 - `{e}` **exact**
 - `{l}` **lowercase**
 - `{n}` **normalized**
 - `{p}` **phonetized**
 - `{s}` **stem**
 - `{a}` **singular lemmas**
 - `{m}` **singular masculine lemmas**
 - **Default value is exact {e}.**
- `/RegularExpr/`: any posix regular expression (without replacement expressions)
- `@Tag@`: the name for the attribute type
- `<comparison operator>` can be one of:
 - `=`
 - `<`; (`<`)
 - `<=`; (`<=`)
 - `>`; (`>`)
 - `>=`; (`>=`)

Example 2. Samples

```

<!-- search the "url" context for all addresses that contain "wikipedia.org/wiki" and
"Thé" in
the "title" context, which is lowercase, or case-insensitive, but does require the a
url:/.wikipedia.org/wiki/ AND title:"Thé">{l}
<!-- Search the "title" context for "Paris" as long as it's not used in the expressio
title:"Paris">{e} BUTNOT title:"Paris Hilton">{n}
<!-- searches the "text" context for the normalized words "Orange" and "Company" -->
text:"Orange">{n} AND text:"Company">{n}
<!-- searches the "text" context for "Optical Zoom" in exact case and "camera" in nor
searches the "title" context
for "camera" in normalized text -->
(text:"Optical Zoom" AND text:"camera">{n}) OR title:"camera">{n}
<!--searches the "text" context for "people" Named Entities annotations that occur wi

```

```
"New York" -->
text:@NE.people@ NEAR/4 "New York"
<!-- search the "price" context for less than 42 -->
price:&lt;=42.0
```

Create the Fast Rules Resource File

Create a Resource File from the Administration Console

The most convenient method consists in creating an empty resources file in the Administration Console and defining its content with the Business Console. See [Create a Resource File from the Administration Console](#) .

To Compile a Resource File from the Command Line

1. Create a rule XML file and save it in a temporary directory. For an example, see [Sample Fast Rules XML Files](#).
2. Compile the Fast Rules XML file.
 - a. Go to <DATADIR>/bin/
 - b. Start the following `cvadmin` command:

```
cvconsole cvadmin> linguistic compile-fastrules input="<PATH TO RULES XML FILE>"
output="<PATH TO OUTPUT FILE>"
```

3. In the Administration Console, select **Index > Data processing > Pipeline name > Semantic Processors**.
4. Drag the **Fast Rules Matcher** processor to the required position in the Current processors list.
5. For **Resource directory**, enter the path to the compiled fast rules file.

Map the Annotation to a Category Facet

Map an Annotation to a Category

1. In the Administration Console, select **Index > Data processing > Pipeline name > Semantic Processors**.
2. On the **Mappings** tab, click **Add mapping source**.
 - **Name:** Enter the annotation name that you created in the rules file, for example, `MyCategory` for the sample file above.
 - **Type:** select **Annotation**.
3. (Optional) In **Input from** field of the mapping, restrict the mapping so it only applies to a subset of comma-separated metas (also known as contexts) associated with this annotation.
4. Click **Add mapping target** and add a category target.

5. Modify the **category mapping** properties. For example, the **Create categories under this root** property must be modified to `Top/MyCategory` in our example.
6. Go to **Search > Search Logics > Your_Search_Logic > Facets** and add a category group.
 - a. Click **Add facet** and enter the name to display in the Mashup UI **Refinements** panel.
 - b. For **Root**, enter the value you have entered for **Create categories under this root** in step 4, for example, `Top/MyCategory`.
7. Click **Apply**.

Lemmatizer

A Lemmatizer produces lemmas for each noun and adjective in the document contexts. A lemma is the semantic root of a word that has different inflections. For example, a lemma of geese is goose.

Use the Lemmatizer to match singular/plural and masculine/feminine versions of words.

The output from the Lemmatizer is typically used by other semantic processors in the document analysis chain.

When to Use

There are two use cases for the lemmatizer query expansion module:

- If you have configured the Ontology Matcher, Rules Matcher, Fast Rules Matcher, or Semantic Extractor for lemma matches, you must define a Lemmatizer before it in the document analysis pipeline.
- To perform lemma query expansion for languages tokenized by the Basis Tech (Extended Languages) tokenizer, you must configure the Lemmatizer semantic processor, and follow some additional steps for tokenization.

Configure Lemmatization Manually

Adapt the steps in [Configure Phonetization Manually](#), replacing the Phonetizer processor with the Lemmatizer processor.

Named Entities Matcher

The Named Entities Matcher is used for named entity detection, typically people, organizations, places, and events.

See [Named Entities Classes and Subclasses](#) for a list of detected entities.

When to Use

The primary goal of named entities extraction is to enrich document with valuable labels. These labels are self-sufficient.

For example, when mapped to facets, they provide useful navigation entry points. But they may also be used as input for further processing such as relation discovery or to highlight some relevant keywords.

Which Entities are extracted?

The **Set of rules to use** option defines the resource that is used to produce the annotations. Each value matches a different resource.

The default value `ne` triggers the extraction of people, organizations, places, and events. It is the only resource that is entirely pre-configured.

For all other resources, you must map allowed NE annotations to categories (see the tooltip corresponding to each resource to know which annotations can be mapped).

See [Named Entities Classes and Subclasses](#) for a list of detected entities.

Note: The value `ne-all` triggers the extraction of all types of entities. The value `ne-basic2` triggers the extraction of extra entities.

Filtering Options

You can also use the following options:

- **Filter out dubious NEs using part-of-speech** – to discard NE annotations for parts of text made of a name followed by a verb or an adverb with the first letter in uppercase. This filter is useful if your documents contain a lot of titles with several capitalized words (what is called "Title Case" or "Headline Style"). It applies to `NE.person`, `NE.place` and `NE.organization`.

For example, let us consider the text "John Make":

"Make" is tagged as a "verb" token by the part-of-speech tagger embedded within this processor, and therefore "John Make" is identified as dubious, and the `NE.person` annotation is discarded.

- **Use resource of known words to disambiguate NE candidates** – This option is only available for English and French. Beware, this option may be very restrictive and is mainly useful to avoid getting too much Named Entity "noise".

It uses a precompiled dictionary resource of known words to disambiguate named entities candidates (the precompiled resource files, `known_words.fr` and `known_words.en` are located under `<INSTALLDIR>/resource/all-arch/namedentities`). These known words are all types of words of a language, nouns, verbs, adverbs, articles, etc.

For example, if this option is selected, "J. Brown" for "James Brown" is not detected as a Named Entity. The initial "J." + the adjective "Brown" is considered as too ambiguous to be considered as a Named Entity.

Yet Disambiguation does not ignore title abbreviations. For example, "Miss Smith", "Mr Smith", "Dr Brown" is detected as Named Entities. It also works if a firstname initial stands between the title abbreviation and the known word, for example, "Dr J. Brown" is detected as a Named Entity too.

Note: These filters involve additional processing, which makes the global process consume more resources.

Example of Named Entities with Filters

Text	Filter using POS	Use Known words	both filters	No filter
J. Brown	NE.person	no annotation	no annotation	NE.person
J. Told	NE.person	no annotation	no annotation	NE.person
Mr Brown	NE.person	NE.person	NE.person	NE.person
Mr Told	no annotation	NE.person	no annotation	NE.person
Mr J. Brown	NE.person	NE.person	NE.person	NE.person
Mr J. Told	NE.person	NE.person	NE.person	NE.person
Teddy Brown	NE.person	NE.person	NE.person	NE.person
Teddy Told	no annotation	NE.person	no annotation	NE.person

Named Entities Classes and Subclasses

These Named Entities classes and subclasses are based on several standard schemas defined on <http://schema.org>.

List of Named Entities Classes and Subclasses Detected by the Named Entity Matcher

NE Type	Annotations	Description	Examples
People	NE.person	Rule-based matching and an ontology of first names, titles.	"John Smith"

NE Type	Annotations	Description	Examples
	subclasses:		
	NE.famousperson	Exact name matching based on an ontology and rules	"Albert Einstein"
	NE.partialperson	Patterns in a rules matcher	"Mr Smith" or "J. Smith"
Organization	NE.organization	Based on ontology and rules	"EXALEAD" "Independant Human Right Commission"
	subclasses:		
	NE.organization.corporate		"EXALEAD" "Walt Disney Company" "Burger King"
	NE.organization.government		"NATO" "Department of Defense" "The Supreme Court"
	NE.organization.nongovernment		"Greenpeace" "Sea Sheperd Conservation Society"
	NE.organization.education		"Harvard" "MIT" "Science-Po Paris"
	NE.organization.sports		"Arsenal" "PSG" "Lakers"
	NE.organization.miscellaneous		"PADI"

NE Type	Annotations	Description	Examples
			"Ju-Jitsu Association"
Place	NE.place	Ontology-based matching	"New Orleans"
	subclasses:		
	NE.place.city		"Cambridge"
	NE.place.country		"United Kingdom"
	NE.place.state		"California"
	NE.place.otheradministrative		"Greater London"
	NE.place.landform		"Mediterranean Sea" "The Highlands"
	NE.place.civicstructure		"Madison Square Garden" "Royal Albert Hall"
Event	NE.event	Rule-based matching	"2nd New York Jazz Festival" "London 2012"
	subclasses:		
	NE.event.cultural		"Avignon Theater Festival" "Asian Regional Meeting" "Cuba's Bishops Conference"
	NE.event.military		"Falklands War" "World-War-II" "Battle of Waterloo"

NE Type	Annotations	Description	Examples
	NE.event.natural		"Hurricane Katrina" "Blizzard of 1993"
	NE.event.political		"French presidential election" "Inauguration of Barack Obama"
	NE.event.religious		"Easter Monday" "Aïd el Kebir" "Pessah"
	NE.event.social		"Independence Day" "World Day for Migrants and Refugees"
	NE.event.sport		"2008 Summer Olympics" "Football World Cup" "Moto GP Championship"
	NE.event.security		"Suicide bombing" "Spinboldak attack"
Date	There are several annotations, see below	Rule-based matching	
	NE.date	Normalized to European numerical standard "day month year" with two-digit days and months	"14 06 1982" "05 12 2003"

NE Type	Annotations	Description	Examples
	NE.date.full	If found, the normalized day of the week is prepended	"Mon 13 02 1977" (English) "Lun 13 02 1977" (French)
	NE.date.uk, NE.date.us	For English text, two annotations are set for ambiguous dates. Use the annotation NE.date.uk for British texts and NE.date.us for American texts	
Price	NE.money	Rule-based matching and ontology for currencies.	"\$2.73" "4,5€" "three hundred million dollars"
	subclasses:	The following subclasses aim at simplifying currency conversions	
	currency.unity		dollar US
	currency.quantity		150
French postal address	NE.address.fr	Rule-based matching and ontology of French cities	"10 place de la Madeleine, 75008 Paris"
French phone number	NE.phone.fr	Rule-based matching	"(+33) 6.82.33.15.12" "05 64 222 222"
Time, duration, and time ranges in French	NE.time	Rule-based matching	"13h45" "3 h 56 min 12 sec" "de 7h03 à 17h28"
Email	NE.email	Rule-based matching	john.smith@gmail.com
URL	NE.url	Rule-based matching	"https://www.exalead.com"

NE Type	Annotations	Description	Examples
IP v4 address	NE.ip	Rule-based matching	"192.168.204.120"
Credit card	NE.creditcard	Generated by Basis Tech tokenizer and rule-based matching	"378282246310005" (American Express) Note: The following formats are not supported: <ul style="list-style-type: none"> Australian BankCard: 5610591081018250 Some VISA Number pattern: 4222222222222 Dankort (PBS): 76009244561 Dankort (PBS): 5019717010103742 Switch/Solo (Paymentech): 6331101999990016

For an example of Named Entities processing, see [Test the Semantic Processing of your Analysis Pipeline](#).

Extract Your Own Named Entities

The rules for the Named Entity matcher are packaged in the product. Depending on the entity type, matching is based on:

- a predefined ontology,
- predefined rules,
- or a combination of the two.

To enrich matches for certain entities, you can add your own Ontology Matcher and Rules Matcher processor to the analysis pipeline. The following table helps you defining the best configuration.

Extract entities with...	when...	for example
rules	<ul style="list-style-type: none"> Entities are either numerical or textual and values are countless. 	dates, phone numbers, emails, URLs, addresses, prices, etc.

Extract entities with...	when...	for example
	<ul style="list-style-type: none"> Context can be identified. For example, in "\$100", the "\$" symbol shows us that 100 is a price. Some parts of your entities are already annotated by the Named Entities Matcher or another resource. 	
ontology resources	<ul style="list-style-type: none"> Entities are textual and values can be listed (not infinite) A resource already exists (employees, categories) The context does not help to identify them Listing them is not a big challenge You need to normalize output values. 	<ul style="list-style-type: none"> first names, cities, days, months, etc. To normalize group of entities like: USA, United States, United States of America, Etats Unis, Estados Unidos -> in United States of America
both rules and ontology resources	The number of values to extract is countless but parts of these entities are a clue to recognize it	<p>Mr Obama</p> <p>We need:</p> <ul style="list-style-type: none"> a resource to annotate Mr, Mister, Miss, etc. a rule to extract persons' names when we have an annotation next to a capitalized word.

Set Block Lists and Allow Lists for Named Entities Extraction

Block List People's Names

- In the Administration Console, add an **Ontology Matcher** to the semantic pipeline after the **Named Entities Matcher**.
 - Expand the Ontology Matcher configuration panel, and click **Create new** to create a new ontology resource.

- b. Click **Apply**.
- c. Click **Edit**.

The Business Console opens to let you configure the ontology resource.

2. Create a `blocklist.person` ontology annotation that lists all the names you do not want to index.
 - a. Click **Add annotation**, and give it a name, for example, `blocklist.person`.
 - b. For this annotation, click **Add display form**, enter the name to block list in **Match text form** and click **Add text form**.
 - c. Repeat the previous substep for all the names you want to block list.
 - d. Click **Go Live**.
3. Create an xml file in your `DATADIR/resource` directory, for example `myannotationmanager.xml` and edit the file to copy the following code:


```
<AnnotationManager name="blocklist remover" xmlns="exa:com.exalead.linguistic.v10"
  ignoreInvalidOperations="true">
  <Remove annotation="NE.person" ifMatchWith="blocklist.person" />
</AnnotationManager>
```
4. Go back to the Administration Console, and add an **Annotation Manager** to the semantic pipeline after the **Ontology Matcher**.
 - a. Expand the **Annotation Manager**.
 - b. Click **Browse** and select the path of the xml file.
 - c. Click **Apply**.
5. Reindex all data.

Test Your Block List Configuration

1. In the Business Console, select **Semantic > Resources**.
2. Select your ontology block list.
3. Use the **Test** tab to test the semantic pipeline behavior.
 - a. Enter one of the block listed names in the **text** field.
 - b. The **Annotations** panel displays the `blocklist.person` tag.

NGram Extractor

An NGram Extractor extracts between min and max length word-grams.

This processor has no interest in itself, and mostly serves as a preprocessing for spell checking or suggestions.

It is supported by all languages.

Normalizer

We usually put the normalizer right after the tokenizer in analysis pipelines. The normalizer computes lowercase and unaccentuated (normalized) forms for each alphanumeric token.

For each alphanumeric token, you must have in output: a `LOWERCASE` annotation and one or two `NORMALIZE` annotations.

All languages where there is a distinction between lowercase vs uppercase and accentuated vs unaccentuated.

Moreover, normalization exceptions are defined for:

- German
- Spanish
- French
- Italian

And normalization alternatives are defined for:

- German

For example, in German, `grüne` (green) has an alternative normalized form `gruene`. You get the following annotations:

- `LOWERCASE grüne`
- `NORMALIZE grune`
- `NORMALIZE gruene`

Ontology Matcher (Resource-Based)

The Ontology Matcher is a semantic processor used for detecting expressions, or text forms in documents.

Detected text forms are then tagged with an annotation name and a corresponding display form. If you map this annotation to a category field, you can see its display forms as a new category in your list of facets. It gathers all the documents containing the text forms you linked to the display form.

Dependencies

If the matching rules for this processor depend on phonetic, stem, or lemma matching, you must add the corresponding processor above this one in the pipeline.

For example, if your rules require phonetic forms, place the **Phonetizer** processor above this processor in the analysis pipeline.

Rules for Ontology Matching

The Ontology Matcher detects expressions. Each expression belongs to an annotation package, which can be seen as a namespace.

- Results are tagged by annotations spanning the range of tokens that has been matched.
- Each annotation package creates an annotation tag.
- Each tagged expression can have several text forms.

The rules for the Ontology Matcher are defined in an XML file saved in a resource directory. They are specified during the configuration of the Ontology Matcher semantic processor in the analysis configuration.

Sample Ontology Matcher XML File

In this example, we use the Ontology Matcher to create an annotation, with value `my.annotation` for a document when there is a reference to the brand **Coca-Cola**.

```
<Ontology xmlns="exa:com.exalead.mot.components.ontology">
  <Pkg path="my.annotation">
    <Entry display="Coca-Cola">
      <Form level="exact" />
    </Entry>
    <Pkg path="subannotation">
      <Entry display="Albert Einstein" lang="en">
        <Form value="Albert E." level="normalized" />
      </Entry>
    </Pkg>
  </Pkg>
  <Pkg path="my.second.annotation">
    <Entry display="Recherche et Développement" lang="fr">
      <Form level="exact" distance="0" />
      <Form level="lowercase" distance="1" />
      <Form level="normalized" distance="2" />
      <Form value="R&D" level="normalized" distance="3" />
      <Form value="R & D" level="exact" distance="4" />
    </Entry>
  </Pkg>
</Ontology>
```

```
</Pkg>
</Ontology>
```

These Ontology rules would create the following annotations:

Input: **"Always Coca-Cola..."** Result: The annotation created is `displayForm="Coca-Cola"`
`tag="my.annotation" level="exact" distance="0"`

Input: **"always coca-cola..."** Result: No annotation is created since the match level is set to exact
 and `"Coca-Cola" != "coca-cola"`

Input: **"the famous albert e." (lang=en)** Result: The annotation created is
`displayForm="Albert Einstein" tag="my.annotation.subannotation" level="normalized" distance="0"`

Input: **"le célèbre albert e." (lang=fr)** Result: No annotation is created since the token is tagged
 as French.

Input: **"recherche et développement"** Result: The annotation created is `displayForm="Recherche et Développement" tag="my.second.annotation" level="lowercase" distance="1"`

Input: **"R&D"** Result: The annotation created is `displayForm="Recherche et Développement" tag="my.second.annotation" level="normalized" distance="3"`

Ontology Rules Syntax

An annotation package is characterized by a path and can contain:

- Subannotations whose path are concatenated using the "." separator.
- A set of expressions to detect.

For example:

```
<Pkg path="subannotation">
  <Entry display="Albert Einstein" lang="en">
    <Form value="Albert E." level="normalized" />
  </Entry>
</Pkg>
```

Display Forms

An expression, or display form, is characterized by a value and an optional language. When a language is specified, only tokens of this language are used for detection. For example,

```
<Entry display="Coca-Cola">      <Form level="exact" /></Entry>
```

A display form can have one or more matches with text forms. A text form contains a value, a normalization level, and an optional distance. For example,

```
<Form value="Albert E." level="normalized" />
```

The distance attribute can be used for scoring the annotation depending on which alternative text form has matched. When the text form's `value` is not specified, the display form for the associated expression is used instead.

Available Matching Normalization Levels

The `level` attribute specifies which form must be matched. Certain levels rely on other semantic processors, which you must place above the Ontology Matcher in the analysis pipeline.

Ontology Matcher Level Attribute and Possible Values

Level	Description
<code>exact</code>	Matches using exact form (the token).
<code>lowercase</code>	Matches using the lowercase form. Requires a Normalizer.
<code>normalized</code>	Matches using the normalized form. Requires a Normalizer.
<code>lemmaSingularMasculine</code>	Matches using the singular/masculine/normalized form. Requires a Lemmatizer.
<code>stem</code>	Matches using the stemmed form. Requires a Snowball Stemmer
<code>phonetic</code>	Matches using the phonetic form. Requires a Phonetizer.

Multilevel Ontology Example

```
<Ontology xmlns="exa:com.exalead.mot.components.ontology">
  <Pkg path="organization">
    <Entry display="company/computer/maker/Lenovo Group">
      <Form value="Lenovo Group" level="exact" />
      <Form value="Lenovo" level="exact" />
    </Entry>
    <Entry display="company/computer/maker/Dell Inc">
      <Form value="Dell Inc" level="exact" />
      <Form value="Dell" level="exact" />
    </Entry>
    <Entry display="company/computer/maker/Hewlett Packard Co">
      <Form value="Hewlett Packard Co" level="exact" />
      <Form value="Hewlett Packard" level="exact" />
    </Entry>
  </Pkg>
</Ontology>
```

```

    <Form value="HP" level="exact" />
  </Entry>
</Pkg>
<Pkg path="IT">
  <Entry display="company/computer/IT/IBM Corp">
    <Form value="IBM" level="exact" />
    <Form value="International Business Machines" level="lowercase" />
    <Form value="IBM Corp" level="exact" />
    <Form value="International Business Machines Corporation" level="lowercase" />
  </Entry>
</Pkg>
</Ontology>

```

Create the Ontology Matcher Resource File

Create a Resource File from the Administration Console

The most convenient method consists in creating an empty resources file in the Administration Console and defining its content with the Business Console. See [Create a Resource File from the Administration Console](#) .

To Compile a Resource File from the Command Line

This procedure describes how to manually create and compile your resources from the command line.

1. Create a rule XML file and save it in a temporary directory. For an example, see [Sample Ontology Matcher XML File](#).
2. Compile the ontology rules XML file:
 - a. Go to <DATADIR>/bin/
 - b. Open the `cvadmin` command tool and start the following command.
3. In the Administration Console, select **Index > Data processing > Pipeline name > Semantic Processors**.
4. Drag the **Ontology Matcher** to the required position in the Current Processors list, expand it and:
 - a. For **Resource directory**, enter the path to the compiled ontology file.
 - b. Select the parameters **Restrict language** and **Keep longest match**

For more information about available parameters, see in the Exalead CloudView XML Configuration Reference Guide.

Map an Annotation to a Category Facet

Once your Ontology Matcher resource file is defined, you can map an annotation to a category field. You are then able to see its display forms as a new category in your list of facets. It gathers all the documents containing the text forms you linked to the display form.

1. In the Administration Console, select **Index > Data processing > Pipeline name > Semantic Processors**.
2. On the **Mappings** tab, click **Add mapping source**.
 - a. **Name:** Enter the annotation name that you created in the rules file, for example, `my.annotation` for the sample file above.
 - b. **Type:** select **Annotation**.
3. (Optional) In **Input from** field of the mapping, restrict the mapping so it only applies to a subset of comma-separated metas (also known as contexts) associated with this annotation.
4. Click **Add mapping target** and add a category target.
5. Modify the category-mapping properties.

For example, the **Create categories under this root** property could be modified to `Top/Product` to contain a `Coca-Cola` category (corresponding to the `Coca-Cola` display form).

6. Go to **Search > Search Logics > Your_Search_Logic > Facets** and add a category group.
 - a. Click **Add facet** and enter the name to display in the Mashup UI **Refinements** panel.
 - b. For **Root**, enter the value you have entered for **Create categories under this root** in step 5, for example, `Top/Product`.
7. Click **Apply**.

Phonetizer

A Phonetizer can improve spell-checking at search-time, by building a phonetic form for each word in the contexts specified.

When to Use

This semantic processor is a prerequisite for the following scenarios:

- To perform search-time phonetic expansion using the Phonetic query expansion module, you must extract phonetic forms from the relevant fields at index time. This creates the dictionary used by the phonetic expansion module at search-time.

Typically, you would set up phonetic extraction through the data model. See [Phonetize a Field Created from a Data Model Property](#).

- If you have configured the Ontology Matcher, Rules Matcher, Fast Rules Matcher, or Semantic Extractor for phonetic matches, you must add a Phonetizer processor above it in the analysis pipeline as described in [Configure Phonetization Manually](#).

Phonetize a Field Created from a Data Model Property

- In **Data Model > Properties for the class name**, expand the property to find out which semantic type this property uses.
- Next to the **Semantic type** list, click the icon to edit the semantic type.
This takes you to the appropriate section on the **Semantic Types** tab.
- Select **Extract phonetic forms**. This adds a **Phonetizer** semantic processor to your analysis pipeline.
- Click **Apply**.
- Reindex the documents that need to phonetic extraction.
- On the **Home** page, locate the appropriate connectors and click **Scan**.

Extracted phonetic forms are saved to the dictionary, so they can be accessed at search time for phonetic expansion. For details, see [Configuring Query Expansion](#).

Configure Phonetization Manually

Follow this procedure when you need to extract phonetic forms that need to be used by other processors, such as the Ontology Matcher.

- In the Administration Console, select **Index > Data Processing > Pipeline name > Semantic Processors**.
- Drag the **Phonetizer** processor to the pipeline.
- In the pipeline, expand the Phonetizer:
 - **Language**: specify a comma-separated list of language ISO codes. Leave blank to process all languages.
 - **Input from**: specify a comma-separated list of document context names to be processed. Leave blank to process all input contexts.
- Drag the processor that requires the extracted phonetic forms so it is below the Phonetizer in the analysis pipeline.
- Expand this dependant processor:
 - **Language**: specify the same comma-separated list of language ISO codes as specified for Phonetizer. Leave blank to process all languages.

- **Input from:** specify the same comma-separated list of document context names as specified for Phonetizer. Leave blank to process all input contexts.
- Finish configuring the dependant processor as described in:
 - [To Compile a Resource File from the Command Line.](#)
 - [Configure the Semantic Extractor.](#)
 - [Create the Fast Rules Resource File.](#)
 - [Create a Rules Matcher Resource File.](#)

Proximity

The Proximity processor is a MOT processor that spots pieces of text where a number of annotations appear close to each other.

The behavior is similar to that of a `NEAR` operator in query language, with more options to express distance constraints.

How Is the Best Match Selected?

When faced with the choice of the best match, that is to say, when several candidate matches overlap, the following criteria are used:

The shortest match in terms of token is preferred. For example, searching for `A NEAR B` in the text `A A B B` selects `A [A B] B`.

The greater the sum of the lengths of element annotations the better. For example, if more than one annotation `A` appear on a token, the longest is chosen.

Configure the Proximity Processor

The configuration includes:

- A list of `<ProximityElement>` defining the set of annotations to search for. These elements have the following attributes:
 - `annotation`: the annotation tag
 - `value`: the annotation display form to match. Regular expressions can be used by enclosing the string in slashes.
 - `mandatory` (default `true`): when set to `false`, the processor can report matches where this annotation is missing.

- `name`: a name that can be referenced in the output annotation format using `$name`. Names must be made of characters in `[0-9a-zA-Z]`.
- An output `annotation` tag and an optional `displayForm` string defining a format where named elements can be referenced to build the output display form. If `displayForm` is undefined, the output annotation does not have any value.
- An `ordered` Boolean attribute forcing the annotations to be matched in their definition order when set to `true`. If `false`, any permutation of the annotations list is allowed to match.
- An `allowElementOverlap` Boolean attribute (default= `false`) allowing the annotations of elements to overlap when set to `true`.
- A number of nonmutually exclusive distance constraints:
 - `sentenceScope` (default `false`) – if `true`, each match is to be contained in a sentence, no match spans several sentences.
 - `paragraphScope` (default `false`) – if `true`, each match is to be contained in a paragraph, no match spans several paragraphs.
 - `windowSize` (default 2048, max 8192) – maximum size of a match in terms of token.
 - `minDistance` (default `none`, max 4096) – the minimum distance between two elements in terms of token.
 - `maxDistance` (default `none`, max 4096) – the maximum distance between two elements in terms of token.

Proximity processor XML configuration file sample

```
<Proximity xmlns="exa:com.exalead.linguistic.v10" annotation="output" displayForm="wh
  ordered="false"
windowSize="100" sentenceScope="true" >
  <ProximityElement annotation="NE.date" value="/[1-5] 201./" mandatory="true" name=
  <ProximityElement annotation="NE.famouspeople" value="Olivier Panis" name="people"
</Proximity>
```

Related Terms

Related terms are a list of nouns or adjectives separated by link words, and shared by at least N documents of your corpus (N=5 by default). This setting can be configured in **Index > Linguistics > Dictionaries > Your_Search_Logic > Related Terms**.

An internal, language-specific resource file that cannot be edited identifies these links.

Related terms are flagged at index time as semantic annotations, based on the configuration of the Related Terms Extractor semantic processor.

Note: You can also add text directly to the dictionary using the dedicated annotation `relatedTermCustom` when defining annotations (**Kind** or **Name** field).

Required Settings

For the Related Terms Extractor you must specify:

Setting	Description
Allow list	Sends the specified expression to the dictionary as a possible related term.
Block list	Blocks the specified expression from displaying as a related term in the Refinements panel.
Related terms min. span	The minimum number of words (excluding stop words) in a generated related term. This parameter is used only when Extract new related terms is enabled. The default is 2.
Related terms max. span	The maximum number of words (excluding stop words) in a generated related term. This parameter is used only when Extract new related terms is enabled. The default is 3.
Max. related terms per doc	The maximum number of related terms per doc; the default is 64.
Keep longest match	Keeps only the longest match. For example, if you have 5 tokens ('a', 'b', 'c', 'd', 'e') and 4 related terms 'a', 'a-c', 'b-c-d' and 'd-e', this option only keeps 'b-c-d' and removes all other related terms; the default is <code>true</code> .

Optional Settings

You can also specify these optional settings:

- **Input from** (optional)
a comma-separated list of context names of the document chunks for which this processor is applied.
- Additional attributes that do not appear in the Administration Console. For a full list and descriptions, see "RelatedTermsSynthesisConfig" in the Exalead CloudView XML Configuration Reference Guide.

To set these options, you must do one of the following:

- `<DATADIR>/config/analysis.xml`, OR
- In the API Console, go to **Indexing > setAnalysisConfigList**.

Search-Time Configuration

For related terms to display on the Refinements panel at search time, they must meet the following criteria:

- Not be shared by more than X% of your hits (X=25 by default).
- Be in at least Y hits (Y=3 by default).
- Have a corpus frequency of at least Z (Z=0 by default).

To configure related terms behavior at search time

- In the Administration Console, select **Search > Search Logics > Your_Search_Logic > Facets**.
- Under **Related terms** (at the bottom), select **Enable**.
- Adjust the options that appear to control which related terms appear in the Refinements panel at search-time.

Rules Matcher (Rule-Based)

The Rules Matcher allows you to define complex rules for matching patterns against a token stream. You can use this processor when you need to define proximity matching rules or define complex rules involving `Opt` or `Iter`.

Unlike the Fast Rules Matcher, the Rules Matcher does not support numerical operators or matching on prefixes.

Results are tagged by annotations spanning the range of tokens that have been matched. At indexing time, these annotations are mapped to category or index fields.

Dependencies

If the matching rules for this processor depend on phonetic, stem, or lemma matching, you must add the corresponding processor above this one in the pipeline.

For example, if your rules require phonetic forms, place the `Phonetizer` processor above this processor in the analysis pipeline.

Basics of Creating Rules

Rules matching is based on an XML file that lists patterns to identify. It is defined using regexp-like expressions.

- The XML file may contain several rules. Each rule has a priority and a specific annotation.
- Use priority to disambiguate multiple matches for the same content.

The best match is determined by the criteria below, applied in this order:

1. Leftmost match
2. Longest match
3. Highest priority value, where 0 = lowest priority.
4. The order in which the rules are defined in the rules file.

- The annotation marks the matched tokens.

For example, a Named-Entity matcher could tag people's names with the annotation (NE, people) and then the Rules Matcher tags the first names with (sub, 1) and the last names with (sub, 2) thus allowing match normalization.

Note: For an explanation of the available Boolean and operators, see [Rules Syntax](#).

Sample Rules Matcher XML File

In this example, we want to identify email addresses, long dates, and short dates in French and English. We use the Rules Matcher to create an annotation for a document when it identifies an email address, based on the character sequence defined in the Rules XML file.

Important: This example also relies on a day names ontology provided by the Named Entities matcher: `<Annotation kind="exalead.nlp.date.days"/>`. For this example to work, you must place the Named Entities matcher before the Rules Matcher processor.

```
<TRules xmlns="exa:com.exalead.mot.components.transducer">
  <!-- 1st rule tags emails with the annotation NE.email -->
  <TRule priority="0">
    <MatchAnnotation kind="NE.email"/>
    <Seq>
      <TokenRegex value="[0-9A-Za-z_]+"/>
      <Noblank/>
      <Iter min="0" max="4">
        <Or>
          <Word value="-" level="exact"/>
          <Word value="." level="exact"/>
        </Or>
      <Noblank/>
    </Seq>
  </TRule>
</TRules>
```

```

    <TokenRegexp value="[0-9A-Za-z_]+" />
    <Noblank />
</Iter>
<Word value="@" level="exact" />
<Noblank />
<TokenRegexp value="[0-9A-Za-z_]+" />
<Noblank />
<Iter min="0" max="6">
    <Or>
        <Word value="-" level="exact" />
        <Word value="." level="exact" />
    </Or>
    <Noblank />
    <TokenRegexp value="[0-9A-Za-z_]+" />
    <Noblank />
</Iter>
<Word value="." level="exact" />
<Noblank />
<Or>
    <TokenRegexp value="[A-Za-z][A-Za-z]" />
    <Word value="gov" />
    <!-- government entities in the US -->
    <Word value="com" />
    <!-- commercial entities -->
    <Word value="net" />
    <!-- network providers -->
    <Word value="org" />
    <!-- non-profit organizations -->
    <Word value="edu" />
    <!-- educational institutions -->
    <Word value="info" />
    <!-- informative websites -->
    <Word value="biz" />
    <!-- business -->
    <Word value="pro" />
    <!-- business use by qualified professionals -->
    <Word value="name" />
    <!-- individuals' real names, nicknames, pseudonyms -->
    <Word value="aero" />
    <!-- aviation-related businesses -->
    <Word value="asia" />
    <!-- region of Asia, Australia, and the Pacific -->
    <Word value="cat" />
    <!-- Catalan language and culture -->
    <Word value="coop" />
    <!-- cooperatives -->
    <Word value="int" />
    <!-- international treaty-based organizations -->

```

```

    <Word value="jobs"/>
  <!-- employment-related sites -->
    <Word value="mil"/>
  <!-- US Department of Defense -->
    <Word value="tel"/>
  <!-- publishing contact data -->
    <Word value="museum"/>
<!-- museums -->
    <Word value="travel"/>
<!-- travel industry -->
  </Or>
</Seq>
</TRule>
<!-- 2nd rule tags dates with the annotations NE.date.full and NE.date.compact -->
<TRule priority="9">
  <!-- All that's been matched by the captures, French way -->
  <MatchAnnotation kind="NE.date.full" value="%1 %2 %3 %4"/>
  <!-- Don't use day of week -->
  <MatchAnnotation kind="NE.date.compact" value="%2/%3/%4"/>
  <!-- matches dates like Samedi 31 janvier, 2004 or 16th of November 2003 -->
  <Seq>
    <!-- 1st capture: optional week day -->
    <Opt>
      <Sub no="1">
        <!-- Day names ontology, taken from Named Entities matcher -->
        <Annotation kind="exalead.nlp.date.days"/>
      </Sub>
    </Opt>
    <Word value=", "/>
  </Opt>
    <Opt>
      <Word value="the"/>
    </Opt>
  </Opt>
  <!-- 2nd capture: day number/ordinal -->
  <Sub no="2">
    <Or>
      <!-- day ordinal : 16, 28, 15th, 1st, 3rd ... -->
      <Annotation kind="exalead.nlp.date.ordinals"/>
      <!-- day number with no ordinals: 16, 28, 1 ... -->
      <TokenRegex value="0?[1-9]"/>
      <TokenRegex value="[12][[:digit:]]" />
      <TokenRegex value="3[01]" />
    </Or>
  </Sub>
  <Opt>
    <Or>
      <Word value="of"/>

```

```

        <Word value="-"/>
    </Or>
</Opt>
<!-- 3rd capture: month name -->
<Sub no="3">
    <Annotation kind="exalead.nlp.date.months"/>
</Sub>
<Opt>
<Word value="-"/> </Opt>
<!-- 4th capture: year -->
<Sub no="4">
    <Or>
        <!-- year like 06 -->
        <TokenRegexp value="[:digit:]{2}"/>
        <!-- full year [1000, 2999] -->
        <TokenRegexp value="[12][[:digit:]]{3}" />
    </Or>
</Sub>
</Seq>
</TRule>
</TRules>

```

Rules Syntax

Booleans

Booleans express constraints on a single token. These constraints can be combined in a tree using the classic operators **AND**, **OR**, and **NOT**. The individual leaf conditions must be met to continue matching. These conditions can be about the format of the token, its possible annotations, its type or language, etc.

Even if Booleans express constraints on a single token, the annotation may match more than one token. A match on a token bearing a specific annotation results in a match of all the tokens that are delimited by the annotation. This can be more than one token long. This is valid for the conditions concerning the keywords `annotation` and `path` (See table below). All others match at most one token.

Boolean Operators	Description
Boolean OR	<p>An <code>Or</code> matches if at least one of its sub expressions matches.</p> <p>The length of the annotation matched is the longest of the sub expression matches.</p>
Boolean AND	An <code>And</code> matches a token if all its sub expressions match.

Boolean Operators	Description
	The length of the annotation matched is the longest of all sub expression matches.
Boolean NOT	A <code>Not</code> matches a token if its sub expression does not match. The length of the annotation matched is 1.
Boolean NOR	A <code>Nor</code> matches a token for the combined Boolean operators <code>Not</code> <code>Or</code> . The length of the annotation matched is 1.
Boolean Atoms	Description
TokenRegexp	<p>A <code>TokenRegexp</code> matches if the exact anchored token string is matched. This is the default behavior. It is, however, possible to define the match as normalized or case-insensitive. The following regexp expressions are not implemented:</p> <ul style="list-style-type: none"> • assertions like <code>\b</code>, <code>\B</code>, <code>?=</code>, <code>?!</code>, <code>?<=</code>, <code>?<!</code> • back references <code>\1</code>, <code>\2</code>, ... • support for UNICODE like <code>\u0020</code> or <code>\p{name}</code> • nongreedy repeat operators like <code>??</code>, <code>*?</code>, <code>+?</code> • octal notation like <code>\0333</code> <p>For example,</p> <pre><TokenRegexp value="0?[1-9] 12 [:digit:]] 3[01]"/></pre>
Word	<p>A <code>Word</code> matches if its value matches the normalized form of the token string. This is the default behavior. It is, however, possible to define the match as "exact" or "case-insensitive".</p> <p>For example,</p> <pre><Word value="-" level="exact"/></pre>
Annotation	<p>An <code>Annotation</code> matches if the token bears an annotation matching the specified kind and possibly a nonrequired value.</p> <p>For example,</p> <pre><And> <Annotation kind="some"/> <Annotation kind="other"/> </And></pre>
Path	A <code>Path</code> matches a path value in an ontology. The implementation relies on annotations emitted by an <code>OntologyMatcher</code> somewhere upstream in the analysis pipeline.

Boolean Atoms	Description
AnyToken	AnyToken matches any token.
Noblank	An assertion matching a nonblank token. Its use is restricted to the root of a Boolean expression.
Digit	A Digit matches a token whose kind is <code>TOKEN_NUMBER</code> (set by the tokenizer for tokens made of a sequence of one or more digits). This is semantically equivalent to using the regular expression <code>[0-9]+</code> but is more efficient since the work has already been done by the tokenizer.
Alpha	Alpha matches a token made of uppercase or lowercase letters.
Alnum	Alnum matches a token made of uppercase/lowercase letters or digits.
Paragraph	paragraph matches a token whose kind is <code>TOKEN_SEP_PARAGRAPH</code> (set by the tokenizer).
TokenLanguage	<TokenLanguage> matches a token with a specific language id. This allows you to write rules, which are triggered for certain languages only.
Punct	A Punct matches a token whose kind is <code>TOKEN_SEP_PUNCT</code> (set by the tokenizer).
Dash	A Dash matches a token whose kind is <code>TOKEN_SEP_DASH</code> (set by the tokenizer).
Sentence	A Sentence matches a token whose kind is <code>TOKEN_SEP_SENTENCE</code> (set by the tokenizer).
TokenKind	<p>A TokenKind matches a token whose kind matches the specified value (set by the tokenizer). Allowed values are:</p> <ul style="list-style-type: none"> • <code>SEP_PARAGRAPH</code> • <code>SEP_SENTENCE</code> • <code>SEP_PUNCT</code> • <code>SEP_QUOTE</code> • <code>SEP_DASH</code> • <code>NUMBER</code> • <code>ALPHANUM</code>. Note, this means alphabetical and numerical, not alphabetical or numerical (use <Alnum> instead) • <code>ALPHA</code>

Operators

Rules operators resemble those of standard regular expressions but with an XML syntax. Only the `<near>` operator has been added to the usual set.

Operators	Description
Concatenation	<code><Seq></code> is a concatenation pattern.
Disjunction	<code><Or></code> is a disjunction pattern.
Proximity	<p><code><Near></code> matches subpatterns at a maximum distance of <code>n</code> nonblank tokens. By default, the order is free but may be imposed by setting the Boolean attribute <code>ordered</code> to <code>true</code>. This pattern matches the longest possible match.</p> <p>Use the <code>slop</code> attribute to set the number of nonblank tokens allowed between A and B (default 0).</p>
Option	<code><Opt></code> matches its subpattern zero or one time.
Bounded Repetition	<p><code><Iter></code> matches the sequence of its subpatterns between min and max times. The maximum is 128.</p> <pre> <Iter min="0" max="6"> <Or> <Word value="-" level="exact"/> <Word value="." level="exact"/> </Or> <Noblank/> <TokenRregex value="[0-9A-Za-z_]+"/> <Noblank/> </Iter> </pre>
Capture	<code><Sub></code> defines a function to tag subparts of an annotation (kind, value) for later retrieval. For example, the day of the week is annotated (sub, 1), the day of the month (sub, 2), the month (sub, 3), and the year (sub, 4). By concatenating subs in increasing order, we can get normalized dates.
Pattern referencing	Each operator can have a name (attribute name) used later for referencing the operator <code><PatternRef></code> .
Pattern reuse	The operator <code><TImport></code> allows the reuse of existing patterns from a file (attribute file name). To be reusable, a pattern must have a name.
Including Rules	The operator <code><TInclude></code> works like a <code>#include</code> in C/C++. It adds to the current TRules set all the TRule objects found in the specified file (attribute file name).

Rules Best Practices

Do the following:

- Use the `<Digit>` operator to match tokens made of a sequence of digits instead of a regular expression. Indeed, it uses the token kind computed by the tokenizer and is therefore more efficient.
- Use a normalizer; it is likely that you need a normalizer somewhere upstream in the analysis pipeline because `<Word>` and `<TokenRegexp>` operators may match against the lowercase or normalized forms of tokens.
- Use the `<Noblank>` operator if you do NOT want to skip spaces. The rules matcher skips spaces and tabs so that the rules are not littered with hundreds of references to blanks. It is however possible to assert that there is no blank token between two tokens at a precise position in the stream with the operator `<Noblank>`.

Avoid the following:

- Using the `<Near>` operator too often. It uses repeat operators, and so matches the longest possible match. This can be costly in terms of compilation time and RAM consumption, so you must try to keep A and B as simple as possible and limit NEAR overlapping.

Caveats

The Rules Matcher has the following caveats:

- It does not report overlapping or embedded matches; the earliest and longest match is reported. If there are ambiguities, the tokens matched by the highest-priority rules are kept. If there are two or more rules with similar priorities, the first rule in the declaration order has highest priority. For the repeat operators (`<Opt>`, `<Iter>`) and `<Near>`, the longest possible match is preferred.
- The operator `<Sub>` allows for sub expression matches retrieval. It has two attributes `kind` and `value` defining the annotation emitted each time the sub expression matches. These "numerical subs" are useful in match normalization. They are defined so that concatenating the text they have matched in increasing order of their value yields normalized matches. For example, people's names detection rules could mark first names with (sub, 1) and last names with (sub, 2) thus giving equal results after concatenation for "John Smith" and "Smith John". Of course, these submatch annotations are only emitted when the overall pattern matches.
- During the compilation, the RulesMatcher performs a number of optimizations, in particular for Boolean ORs that are, whenever possible, replaced with a single regular expression that indicates if any of the conditions in the list match. For example, the following example is

automatically rewritten to the second simplified expression, which is more efficient. The same thing is done with words `<word>` are transformed to regular expressions.

```
<Or>
  <TokenRegexp value="0?[1-9]" />
  <TokenRegexp value="[12][[:digit:]]" />
  <TokenRegexp value="3[01]" />
</Or>
<TokenRegexp value="0?[1-9]|[12][[:digit:]]|3[01]" />
```

- The `<And>` operator does not impose that lengths of submatches be equal. A match is found if all its subpatterns match, irrespective of the length of the matches. The following example matches even if one annotation does not have the same length as another annotation, provided that they are both present on the same token. The match length is the length of the longest annotation. For example:

```
<And>
  <Annotation kind="some" />
  <Annotation kind="other" />
</And>
```

Limitations

The Rules Matcher has the following limitations:

- The window size is 200 tokens. This is the maximum length of a match.
- The upper bound for the operator `<iter>` must remain as low as possible as it is costly in terms of resources. The maximum is limited to 128.
- UNICODE is not handled and matching is done on UTF-8 strings without specific processing (at the byte level). Consequently:
 - Accentuated characters do not match if they are used in case-insensitive or normalized mode.
 - The dot wildcard matches a byte and therefore not all UTF-8 characters.

Create a Rules Matcher Resource File

Create a Resource File from the Administration Console

The most convenient method consists in creating an empty resources file in the Administration Console and defining its content with the Business Console. See [Create a Resource File from the Administration Console](#) .

To Compile a Resource File from the Command Line

1. Create a rule XML file and save it in the resource directory. For more information, see [Basics of Creating Rules](#).
2. In the Administration Console, select **Index > Data processing > Pipeline name > Semantic Processors**.
3. Drag the **Rules Matcher** to the required position in the Current Processors list.
4. Enter the **Resource file** path.

Map the Annotation to a Category Facet

We now need to configure the Rules Matcher processor to map the `NE.email` annotation to a category facet that represents the email address. This allows the document to be related to all email addresses found in it.

1. In the Administration Console, select **Index > Data processing > Pipeline name > Semantic Processors**.
2. On the **Mappings** subtab, click **Add mapping source**.
 - a. **Name:** Enter the annotation name that you created in the rules file, for example, `NE.email` for the sample file above.
 - b. **Type:** select **Annotation**.
3. (Optional) In **Input from** field of the mapping, restrict the mapping so it only applies to a subset of comma-separated metas (also known as contexts) associated with this annotation.
4. Click **Add mapping target** and add a category target.
5. Modify the category-mapping properties.

For example, the **Create categories under this root** property must be modified to `Top/Email` in our example.

6. Go to **Search > Search Logics > Your_Search_Logic > Facets** and add a category group.
 - a. Click **Add facet** and enter the name to display in the Mashup UI **Refinements** panel.
 - b. For **Root**, enter the value you have entered for **Create categories under this root** in step 4, for example, `Top/Email`.
7. Click **Apply**.

Semantic Extractor

The Semantic Extractor performs the following:

- Extracts the following entities based on triggers and units.

- Interprets matched entities with rewrite rules.

Entities and Attributes

The following entities are available:

Entity type	Description
<TextEntity>	extracts string values
<BooleanEntity>	extracts Boolean values
<IntegerEntity>	extracts integer values
<FloatingPointEntity>	extracts floating point values
<RangeEntity>	extracts ranged string values
<RegexEntity>	extracts string values following a pattern

Entities are defined with the following common attributes and sets of specific attributes:

Common Entity Attributes

Attribute	Type	Description
trigger or (s)	string	An optional left context triggering the entity detection.
triggers	NamedObjectList	An optional list of left contexts triggering the match
leftContext	string	Alias for trigger
leftContexts	NamedObjectList	Alias for triggers
annotation	string	The annotation set by the processor in case of a match
display	string	The annotation value in case of a match. Each ? in your custom display form is replaced by the matching value.
matchMode	string	The level used to match the feature: <code>normalized</code> , <code>lowercase</code> , or <code>exact</code>
name	string	Unique entity reference. Only used by the GUI.
unit	String	An optional condition on right context
units	NamedObjectList	An optional list of conditions on right context
rightContext	String	Alias for unit

Attribute	Type	Description
<code>rightContexts</code>	<code>NamedObjectList</code>	Alias for units

Text Entity Specific Attributes

Attribute	Type	Description
<code>value</code>	<code>string</code>	The values to match, separated with ' '
<code>maxValueSize</code>	<code>int</code>	Maximum number of tokens for the value
<code>lang</code>	<code>iso code</code>	Restricts matching to a specific input language.

Boolean Entity Specific Attributes

Attribute	Type	Description
<code>yes</code>	<code>string</code>	The value for true
<code>no</code>	<code>string</code>	The value for false

Integer and Floating Point Entity Specific Attributes

Attribute	Type	Description
<code>step</code>	<code>int</code>	If defined, normalizes the value to the nearest step. default=0
<code>min</code>	<code>int</code>	Minimum value for a match. default=-2147483648
<code>max</code>	<code>int</code>	Maximum value for a match. default=2147483647
<code>coefficient</code>	<code>double</code>	The normalization coefficient used to multiply the matched value. default=1
<code>precision</code>	<code>int</code>	The floating-point precision in output. default=0
<code>handleOutOfBoundVa</code>	<code>Boolean</code>	If false, ignores values lower than min or greater than max. default=True
<code>addExactValue</code>	<code>Boolean</code>	If true, adds the original value (before normalization) in an extra annotation <code>TAG.exact</code>

Attribute	Type	Description
		default=False
<code>truncateTrailingZeros</code>	Boolean	If true, removes the trailing zeros after point. default=False

Range Entity Specific Attributes

Attribute	Type	Description
<code>dimension</code>	int	Dimension count. default=2
<code>delimiter</code>	string	Numbers delimiter. default="x"

Regex Entity Specific Attributes

Attribute	Type	Description
<code>value</code>	string	Perl-5 regular expression
<code>endTrigger</code>	string	An optional left context
<code>endTriggers</code>	NamedObjectList	An optional list of left contexts

Rule Attributes

A Rule is defined by the following attributes:

- `name`
- `mode`
- `value` (pattern)
- `output` string
- `trustLevel`

Dependencies

If the matching rules for this processor depend on phonetic, stem, or lemma matching, you must add the corresponding processor above this one in the pipeline.

For example, if your rules require phonetic forms, place the **Phonetizer** processor above this processor in the analysis pipeline.

Sample Semantic Extractor XML File

The Semantic Extractor configuration is made of two nonmandatory parts: a list of entity definitions and a list of rules. There are also two operators for macro definitions and inclusion of external configurations.

```
<SemanticExtractorConfig xmlns="exa:com.exalead.mot.components.semanticextractor" xml:lang="en">
  <!-- define macros usable in the rules' values -->
  <Define name="id" value="/REF-[[[:alnum:]]+/{name=id}" />
  <Define name="near" value=":WORD{max=10,name=near}" />
  <!-- entities definition -->
  <TextEntity value="SD" annotation="sdcard" display="SD Card"/>
  <TextEntity annotation="reference" value="ref|reference|references" matchMode="normal"/>
  <IntegerEntity annotation="weight" unit="g" display="? grammes" step="100" min="0" max="1000"/>
  <triggers>
    <bee:StringValue value="poids"/>
    <bee:StringValue value="weight"/>
  </triggers>
</IntegerEntity>
  <TextEntity annotation="hddtype" value="SATA|SSD" matchMode="exact" />
  <IntegerEntity annotation="capacity" min="0" max="2000" step="100" unit="GB" display="GB"/>
  <FloatingPointEntity trigger="withPrecision" annotation="prec" precision="2" truncate="true"/>
  <RangeEntity trigger="resolution" delimiter="x" annotation="resolution" />
  <!-- rules definition -->
  <Rule name="type" value="capacity hddtype" output="Hard Drive" mode="match"/>
  <Rule name="type" value="capacity hddtype{not} sdcard" output="Camera" mode="match"/>
  <Rule name="reference_number" value="[ reference #near #id ]" output="$ (id)" mode="match"/>
</SemanticExtractorConfig>
```

Each matching entity generates a semantic annotation, which can be mapped using the standard annotation mappings. Similarly, each matching rule generates a semantic annotation identified by the rule's name.

Let us use the above sample to process the following text: "SD Card 4 GB". This generates the following annotations:

- "sdcard", because the processor detected the expression "SD". It displays as "SD Card".
- "capacity", because the processor initially detected the integer "4". It displays as "4 GB".
- "type", because the processor detected both the "sdcard" and "capacity" annotations, but did not detect the "hddtype" annotation. It displays as "Camera".

Entities Syntax

Text

Extract values according to the triggers (left context), values and units (right context) you specified, as defined by:

```
(trigger1 | trigger2 ...)? (separators)* VALUE_TO_EXTRACT (separators)* (unit1 | unit2 ...)?
```

A simple example:

```
<!-- match the expressions "SD card", "sd card", "SD CaRd", ... -->
<TextEntity value="SD card" annotation="sd_card" matchMode="normalized" />
```

To specify multiple value attributes, use the '|' character as a separator:

```
<!-- match the expression "SD card" or "Carte SD" -->
<TextEntity value="SD card|Carte SD" annotation="sd_card" matchMode="exact"/>
```

To restrict matching, use triggers to specify a left context:

```
<!-- match the expressions "Operating System: Mac OS X" or "Operating System: Windows X" -->
<TextEntity trigger="Operating System" annotation="os" matchMode="normalized" value="Mac OS X|Windows X"/>
```

To specify multiple triggers:

```
<!-- match the expressions "Operating System: Mac OS X", "Operating System: Windows X",
"Systeme d'exploitation: Mac OS X" or "Systeme d'exploitation: Windows XP" -->
<TextEntity annotation="os" matchMode="normalized" value="Mac OS X|Windows XP">
  <triggers>
    <bee:StringValue value="Operating System" />
    <bee:StringValue value="Systeme d'exploitation" />
  </triggers>
</TextEntity>
```

To specify how many tokens are used to build the value, omit the `value` attribute and you use the `maxValueSize` attribute:

```
<!-- match the expressions "Operating System: something really unknown" -->
<TextEntity trigger="Operating System" annotation="os_unknown" matchMode="normalized" maxValueSize=10 />
```

Boolean

This condition performs a Boolean match. For example, this condition matches the expression "SD card: YES".

```
<!-- match the expressions "SD card: YES" -->
<BooleanEntity trigger="SD card" annotation="sd_card" matchMode="exact" yes="YES" no="NO" />
```

Integers

The integer entity is often used with normalization. It uses the following attributes:

Integer Entity Attributes

Attribute	Description
min	The minimum value the extracted number must have.
max	The maximum value the extracted number must have.
coefficient	The coefficient applied to the extracted number, default: 1 (no coefficient).
precision	The precision used to generate the display form (optional).
step	The step used to generate the display form (optional).
handleOutOfBoundVa	Generates display form with "<" or ">". If false, skips numbers that are not in the [min-max] range. default: true
addExactValue	If true, adds the value of the extracted number before any normalization in another annotation suffixed by ".exact"; default: false

```
<!-- match the expressions "Port USB: 2", and generate the display form "2" with an
<IntegerEntity trigger="port usb" annotation="port_usb" matchMode="normalized" />
<!-- match the expressions "Size: 1 Ko", and generate the display form "1024 octet" -
<IntegerEntity trigger="size" unit="Ko" annotation="size" matchMode="normalized" coef
display="? octet" />
<!-- match the expressions "HDD: 320 Go", and generate the display form "300-400 Go"
<IntegerEntity trigger="HDD" unit="Go" annotation="hdd_capacity" matchMode="normalize
step="100" display="? Go" />
<!-- match the expressions "HDD: 320 Go", and generate the display form "300-400 Go"
annotation with display form "320" -->
<IntegerEntity trigger="HDD" unit="Go" annotation="hdd_capacity" matchMode="normalize
step="100" display="? Go" addExactValue="true" />
```

Floating-Point

```
<!-- match the expressions "Port USB: 2", and generate the display form "2" with an
display =?"-->
<IntegerEntity trigger="port usb" annotation="port_usb" matchMode="normalized" />
<!-- match the expressions "Size: 1 Ko", and generate the display form "1024 octet" -
<IntegerEntity trigger="size" unit="Ko" annotation="size" matchMode="normalized" coef
display="? octet" />
<!-- match the expressions "HDD: 320 Go", and generate the display form "300-400 Go"
<IntegerEntity trigger="HDD" unit="Go" annotation="hdd_capacity" matchMode="normalize
max="2000" step="100" display="? Go" />
```

```
<!-- match the expressions "HDD: 320 Go", and generate the display form "300-400 Go"
annotation with display form "320" -->
<IntegerEntity trigger="HDD" unit="Go" annotation="hdd_capacity" matchMode="normalized"
max="2000" step="100" display="? Go" addExactValue="true" />
```

`<FloatingPointEntity>` has the same attributes as the `<IntegerEntity>`, but use floats instead of integers.

Ranged

The `<RangeEntity>` is used to extract ranged expressions.

Range Entity Attributes

Attribute	Description
dimension	The number of dimensions in your entity
delimiter	The delimiter used to separate dimensions

```
<!-- match the expressions "Resolution: 800 x 600" -->
<RangeEntity trigger="resolution" annotation="resolution" matchMode="normalized" dimension="2" />
<!-- match the expressions "Size: 42 x 12 x 32" -->
<RangeEntity trigger="size" annotation="size" matchMode="normalized" dimension="3" delimiter="x" />
```

RegExp

The `<RegexEntity>` extracts expressions matching a pattern.

```
<!-- will match the expressions "Opening hours: 09:00 - 18:00" -->
<RegexEntity trigger="Opening hours" annotation="opening" matchMode="normalized"
value="[:digit:]+[:digit:]+ - [:digit:]+[:digit:]+" />
<!-- will match the expressions "Start at: 09:00 am", "Start at: 01:30 pm" -->
<RegexEntity trigger="Start at" annotation="start_at" matchMode="normalized"
value="[:digit:]{2}[:digit:]{2}" />
  <endTriggers>
    <bee:StringValue value="am" />
    <bee:StringValue value="pm" />
  </endTriggers>
</RegexEntity>
```

Rules Syntax

Mode

The mode specifies if the rule is a positive one (value match) or an exception (value filter). A matching exception prevents any rule from matching.

Pattern

A pattern is made of atoms and operators. An atom defines a basic matching element in a rule, it can take the following forms:

- A reference to an entity through its annotation (`country`).
- A regular expression surrounded by slashes possibly capturing parts of a match (`/[pP].*(ern)/`).
- A full-text string surrounded by quotes (`"sequence of words"`).
- A reference to a MOT annotation (`NE.people`).
- An assertion enforcing match constraints:
 - `:WORD` matches any word in a sequence.
 - `:ENDLINE` matches carriage returns.
 - `:START` matches at the beginning of the input text.
 - `:END` matches at the end of the input text.

Each atom may have a set of options specified in curly braces immediately following it. That set is a combination of 0 or more elements separated with commas.

Atom Options

Option	Description
<code>name</code>	allows the atom to be referenced in the output format of a rule through <code>\$(country)</code> .
<code>opt</code>	matching of this atom is not required.
<code>not</code>	the pattern matches if this atom does not.
<code>max</code>	maximum number of words allowed for a <code>:WORD</code> atom to match.
<code>original</code>	use the matching text when building the output rather than the output value of an entity or the display form of an annotation.
<code>default</code>	defines a default value for an optional atom (option <code>opt</code>) to use when building the output.
<code>exact</code>	requires exact matching level.
<code>lower</code>	requires lowercase matching level.
<code>norm</code>	requires normalized matching level.

Option	Description
<code>skipPunct</code>	requires punctuation-insensitive matching
<code>fuzzy</code>	requires approximative matching.

Atom Operators

Operator	Description
<code>AND</code>	Default operator. Matches atoms in any order and positions.
<code>[]</code>	Matches the enclosed sequence of atoms in the order specified.
<code>()</code>	Matches the enclosed atoms at the same position (cumulative constraints on a single position).

Output

The final annotation has the name of the matching rule and the rule trust level if any (default is 100). The display form is defined with an output format that may contain:

- A reference to what has been matched by atom through its name (`$ (country)`).
- A reference to a regular expression capture through the atom name and an array access syntax specifying the capture number (`$ (regexp[0])`).
- A carriage return `\n`.
- Any character sequence, which is used as-is in the output (`the street is:$(street)` and `the city is:$(city)`).

If defined, the processor's parameter prefix forces the output annotation to be prefixed with its value.

Macros

For often-used rule parts, use the `<Define>` element to write macros that you can then reference in a rule value.

A macro has a name and a value. These are substituted each time the macro is referenced with a `#`, followed by the macro's name.

- This name can only contain the characters `[a-z][A-Z][0-9]`.
- The macro can be used anywhere in a rule, such as inside a regular expression.
- To disable macro substitution, insert a backslash before the `#` symbol.

Create the Semantic Extractor Resource File

This section explains how to compile the semantic extractor's resource file, and describes the parameter options available in the Administration Console.

Create a Resource File from the Administration Console

The most convenient method consists in creating an empty resources file in the Administration Console and defining its content with the Business Console. See [Create a Resource File from the Administration Console](#).

To Compile a Resource File from the Command Line

1. Create a rule XML file and save it in a temporary directory. For an example, see [Sample Semantic Extractor XML File](#).
2. Compile the XML file.
 - a. Go to <DATADIR>/bin/
 - b. Run the following `cvadmin` command:


```
cvconsole cvadmin> linguistic compile-semantic-extractor input="<PATH TO XML FILE>"
output="<PATH TO OUTPUT FILE>"
```
3. In the Administration Console, select **Index > Data processing > Pipeline name > Semantic Processors**.
4. Drag the **Semantic Extractor** processor to the required position in the **Current Processors** list, then specify the parameters:

Option	Description
Resource directory (required)	enter the URL to the compiled semantic extractor file. Use the format <code>data://</code> , <code>file://</code> , or <code>resource://</code> .
Break on sentence	maximum of one match per sentence, no match intersentence if true (default false).
Break on paragraph	maximum one match per paragraph, no match interparagraph if true (default true).
Break on line	maximum one match per line, no match interlines if true (default false).
Match all rules	returns the matches for all rules if true, otherwise stops after the first matching rule (default true).

Map the Annotation to a Category Facet

1. In the Administration Console, select **Index > Data processing > Pipeline name > Semantic Processors**.
2. On the **Mappings** subtab, click **Add mapping sources**.
 - a. **Name:** Enter the annotation name that you created in the rules file.
 - b. **Type:** select **Annotation**.
3. (Optional) In **Input from** field of the mapping, restrict the mapping so it only applies to a subset of comma-separated metas (also known as contexts) associated with this annotation.
4. Click **Add mapping target** and add a category target.
5. Modify the **Category Mapping** properties. For example, the **Create categories under this root** property must be modified to `Top/Megapixel` in the example above.
6. Go to **Search > Search Logics > Your_Search_Logic > Facets** and add a category group.
 - a. Click **Add facet** and enter the name to display in the Mashup UI **Refinements** panel, for example `Megapixel`.
 - b. For **Root**, enter the value you entered for **Create categories under this root** in step 4, for example, `Top/Megapixel`.
7. Click **Apply**.

Semantic Query Analysis

Semantic query analysis is a special use case for the Semantic Extractor that allows you to rewrite queries without having to write any custom code. This analyzer runs on the raw user query, making use of a Semantic Extractor. The output is used as a new UQL query processed by the search logic in the usual way.

Configure Semantic Query Analysis

The configuration consists of:

- a semantic extractor's compiled resource
- an optional list of semantic processors, which runs before the semantic extractor
- options to set the analyzer's behavior

Add a Semantic Query Analysis to Your Searcher

1. Go to **Administration Console > Search Logics > Query Language** tab.
2. In **Semantic query analysis**, select **Enable**.

3. In **Resource directory**, create a new semantic resource extractor or select an existing one.
4. In **Semantic processors**, select the semantic processors (if any) linked to the semantic resource extractor selected previously.
5. In **Language**, select the languages for which the analyzer is activated.
6. In **Unused word policy**, select one of the following options:
 - **mandatory**: all query words that have not been used by the matching rule to build the output are added to the output query using a AND.
 - **optional**: all query words that have not been used by the matching rule to build the output are added to the output query using an OPT.
 - **remove**: all query words that have not been used by the matching rule to build the output are discarded.

You can also set the unused word policy at the rule level:

- a. Select **Single match mode**.
 - b. In the Business Console, edit your rule and select the appropriate option for **Unused word policy**. This rule setting overrides the **Unused word policy** set at the Semantic Query Analysis level.
7. In **Debug log file**, enter the path to an HTML file for debug purposes.
 8. Click **Save** and apply the configuration.

Configure Query Processing

You can add a list of comma-separated query names that defines which parts of the query are processed. The default value is `_default_`. This means that by default processing is only applied on the query entered by the user, and not on refinements and restrictions applied by query expansion.

1. Open the API Console.
2. Click **Manage**.
3. Select **search** in the list.
4. In **Configuration**, select **setSearchLogicList**.
5. Search for `queryNames`.
6. Replace `_default_` with the list of query names.

Example 1: Define "Cheap" for an E-Commerce Site

Let us say that you have an online store. You have an index of all product names and characteristics, including a numeric `price` field, and you want to make sense of queries such as `cheap USB flash drive` or `inexpensive USB flash drive`.

With the following configuration, you can rewrite such queries to `USB AND flash AND drive AND price<10`.

Create a Resource File from the Administration Console

1. In the **Resource directory** field, click **Create new** and enter the name of your resource file.
2. Then you can define its content with the Business Console. For more information, see the Exalead CloudView Business Console User's Guide.

Compile a Resource File from the Command Line

1. Create a semantic extractor configuration as shown below:

```
<SemanticExtractorConfig xmlns="exa:com.exalead.mot.components.semanticextractor"
xmlns:bee="exa:exa.bee">
  <!-- entities definition -->
  <TextEntity annotation="cheap" value="cheap|inexpensive|low cost|lowcost|affordable"
display="price<10" matchMode="normalized"/>
  <!-- rules definition -->
  <Rule name="cheap product rule" value="cheap{name=matched}" output="$(matched)" />
</SemanticExtractorConfig>
```

2. Compile the resource:

- a. Go to `<DATADIR>/bin/`
- b. Run the following `cvadmin` command:

```
cvconsole cvadmin> linguistic compile-semantic-extractor input="<PATH TO XML SOURCE>"
output="<PATH TO BINARIES DIRECTORY>"
```

Configure Semantic Query Analysis

1. Specify the **Unused word policy** parameter to **mandatory**. It rewrites `cheap` to `price<10` and relies on the **mandatory** parameter to get a big AND.

The resulting syntax tree:

```
AND
NUM: document_price OP_LT 10
AND NATURAL
  ALPHA: text: usb k=2 (form: normalized)
  ALPHA: text: flash k=2 (form: normalized)
  ALPHA: text: drive k=2 (form: normalized)
```

Here is the final ELLQL query sent to the index:

```
#query{nbdocs=0, score.expr="@term.score * @proximity + @b", proximity.maxDistance=10,
term.score=RANK_TFIDF}
(#and(#num(document_price,<,10)#and(#alphanum{source="MOT",seqid=0,groupid=0,k=2}{
#alphanum{source="MOT",seqid=1,groupid=0,k=2}(text,"flash")
#alphanum{source="MOT",seqid=2,groupid=0,k=2}(text,"drive") ) ) )
```

Example 2: Define "Cheap" for Different Products

Now that you understand the basic principle of semantic query analysis, let us look at an example where you want to define different criteria for "cheap", depending on the product type. It does not make sense for a query for "cheap tv set" to search for TVs with a price of 10€ or lower.

The solution is to create text entities for products, and associate a definition of "cheap" for that particular product.

Configure the Semantic Extractor

1. Create a semantic extractor configuration as shown below:

```
<SemanticExtractorConfig xmlns="exa:com.exalead.mot.components.semanticextractor"
  <TextEntity annotation="product" value="usb flash drive" display="10" matchMode="normalized"
  <TextEntity annotation="product" value="tv set" display="100" matchMode="normalized"
  <TextEntity annotation="product" value="computer|laptop" display="300" matchMode="normalized"
  <TextEntity annotation="cheap" value="cheap|inexpensive|low cost|lowcost|affordable"
  matchMode="normalized"/>
  <Rule name="cheap product rule" value="cheap product{name=threshold,original=text}
  output="$(text) AND price<$(threshold)" mode="match"/>
</SemanticExtractorConfig>
```

- `name=threshold` defines `$(threshold)` as the product annotation's display form (the price threshold)
- `original=text` defines `$(text)` as the input text annotated by product (the product name as entered by the user)

Note: We could have built an ontology instead of writing TextEntities and add an ontology matcher in the SemanticQueryAnalysisConfig, externalizing the information.

Configure Semantic Query Analysis

We now want to remove words matched by the text entity "cheap", since it is not referenced in the output.

1. Specify the **Unused word policy** parameter to **remove** to get rid of them:

The resulting syntax tree for "cheap tv set" is:

```
AND
  AND NATURAL
    ALPHA: text: tv k=2 (form: normalized)
    ALPHA: text: set k=2 (form: normalized)
  NUM: document_price OP_LT 100
```

Snowball Stemmer

A Snowball Stemmer builds a stem, or root, for each word in the document contexts. The root is determined by cutting off word endings. For example, a stem of geese is goose.

The output from the Snowball Stemmer is used by other semantic processors in the document analysis chain.

When to Use

If you have configured the Ontology Matcher, Rules Matcher, Fast Rules Matcher or Semantic Extractor for stem matches, you must define a Snowball Stemmer before it in the document analysis pipeline.

Configure Stemming Manually

Adapt the steps in [Configure Phonetization Manually](#), replacing the Phonetizer processor with the Snowball Stemmer processor.

Part of Speech Tagger

The Part of Speech Tagger allows you to enable part of speech processing that is usually only activated when you perform related terms processing.

How to use

Specify:

- the path to the part of speech resource file for the specific language to configure,
- a language for which the part of speech is activated that corresponds to the resource file specified,
- a comma-separated list of context names of the document chunks for which this processor is applied.

When to use

This processor is required by the Related Term processor.

Appendix - Semantic Resources Reference

Ontology

- `com.exalead.mot.components.ontology.Ontology`
- No documentation for this element.
- Attributes:

Name	Type	Default value	Description
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>matchOnSeparators</code>	<code>boolean</code>	<code>True</code>	If you want to skip separators, set this Boolean to false
<code>matchOnSeparatorsBut</code>	<code>string</code>		If you want to skip only a set of separators, specify them here ex: <code>matchOnSeparatorsBut="-_"</code> matches on separators but skip '-' and '_'

- Nested elements:

Name	Type	Description
<code>OInclude</code>	<code>com.exalead.mot.components.ontology.OInclude</code>	
<code>Pkg</code>	<code>com.exalead.mot.components.ontology.Pkg</code>	

OInclude

- `com.exalead.mot.components.ontology.OInclude`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.ontology.Ontology` (**as** `Ontology`)
 - `com.exalead.mot.components.ontology.Pkg` (**as** `Pkg`)
- Attributes:

Name	Type	Default value	Description
fileName	string		Path to external ontology to include

Pkg

- `com.exalead.mot.components.ontology.Pkg`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.ontology.Pkg` (**as** Pkg)
- Attributes:

Name	Type	Default value	Description
modifiedBy	string		
modifiedAt	nullablelong		
path	string		Package name, used as annotation tag
disabled	boolean		

- Nested elements:

Name	Type	Description
OInclude	<code>com.exalead.mot.components.ontology.OIn</code>	
PkgOrEntry	<code>com.exalead.mot.components.ontology.Pkg</code>	

Entry

- `com.exalead.mot.components.ontology.Entry`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.ontology.Pkg` (**as** Pkg)
- Attributes:

Name	Type	Default value	Description
modifiedBy	string		

Name	Type	Default value	Description
modifiedAt	nullablelong		
display	string		Display form of the annotation
lang	iso code		Language of forms in this entry when not specified
kind	int		Kind of the annotation
disabled	boolean		

- Nested elements:

Name	Type	Description
Form	com.exalead.mot.components.ontology.Form	

Form

- `com.exalead.mot.components.ontology.Form`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.ontology.Entry` (as `Entry`)
- Attributes:

Name	Type	Default value	Description
modifiedBy	string		
modifiedAt	nullablelong		
value	string		Matching expression
level	string		Matching level
lang	iso code		Used to restrict a match to a specific language
distance	int		Specifies the distance to apply to the default annotation trustLevel (that is, 100-distance)
disabled	boolean		

FastRulesDefinition

- `com.exalead.mot.components.fastrules.FastRulesDefinition`
- A set of categories together with their associated rules
- Attributes:

Name	Type	Default value	Description
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>catName</code>	<code>string</code>		The category name

- Nested elements:

Name	Type	Description
<code>Category</code>	<code>com.exalead.mot.components.fastrules.Category</code>	
<code>DateFormat</code>	<code>com.exalead.mot.components.fastrules.DateFormat</code>	

Category

- `com.exalead.mot.components.fastrules.Category`
- A set of rules and a category value for matching documents
- Parent elements:
 - `com.exalead.mot.components.fastrules.FastRulesDefinition` (as `FastRulesDefinition`)
- Attributes:

Name	Type	Default value	Description
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>value</code>	<code>string</code>		The category value

- Nested elements:

Name	Type	Description
Rule	<code>com.exalead.mot.components.fastrules.Rule</code>	

Rule

- `com.exalead.mot.components.fastrules.Rule`
- A rule expressing constraints on a document content
- Parent elements:
 - `com.exalead.mot.components.fastrules.Category` (**as** `Category`)
- Attributes:

Name	Type	Default value	Description
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>value</code>	<code>string</code>		A query defined with a subset of Exalead User Query Language
<code>exceptionRule</code>	<code>boolean</code>		Makes the rule an exception rule instead of a normal rule. When any of the exception rules of a category matches, the category is not assigned to the document, even if some of its normal rules match.
<code>lang</code>	<code>iso code</code>	<code>xx</code>	Restrict this query to a specific language

DateFormat

- `com.exalead.mot.components.fastrules.DateFormat`
- A date format definition conforming to C function `strptime`
- Parent elements:
 - `com.exalead.mot.components.fastrules.FastRulesDefinition` (**as** `FastRulesDefinition`)
- Attributes:

Name	Type	Default value	Description
modifiedBy	string		
modifiedAt	nullablelong		
value	string		The date format value

LemmaDictionary

- `com.exalead.mot.components.lemmatizer.LemmaDictionary`
- No documentation for this element.
- Attributes:

Name	Type	Default value	Description
modifiedBy	string		
modifiedAt	nullablelong		
lang	iso code		

- Nested elements:

Name	Type	Description
Lemma	<code>com.exalead.mot.components.lemmatizer.L</code>	

Lemma

- `com.exalead.mot.components.lemmatizer.Lemma`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.lemmatizer.LemmaDictionary` (as `LemmaDictionary`)
- Attributes:

Name	Type	Default value	Description
value	string		
pos	string		
trustLevel	int	100	

- Nested elements:

Name	Type	Description
inflecteds	<code>com.exalead.mot.components.lemmatizer.I</code>	Inflected forms of the word

Inflected

- `com.exalead.mot.components.lemmatizer.Inflected`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.lemmatizer.Lemma` (**as** `inflecteds`)
- Attributes:

Name	Type	Default value	Description
value	string		Inflected form of the word.
number	<code>enum(singular,unnumbered plural,unnumbered)</code>		Number of the inflected form
gender	<code>enum(masculine,neutral feminine,neutral)</code>		Gender of the inflected form

NormalizationOverwrites

- `com.exalead.mot.components.normalizer.NormalizationOverwrites`
- No documentation for this element.
- Nested elements:

Name	Type	Description
NormalizationOver	<code>com.exalead.mot.components.normalizer.N</code>	rite*

NormalizationOverwrite

- `com.exalead.mot.components.normalizer.NormalizationOverwrite`

- overwrite normalization for a specific letter, for example, umlaut exceptions: ä -> ae, ü -> ue, ö -> oe ...
- Parent elements:
 - `com.exalead.mot.components.normalizer.NormalizationOverwrites` (as `NormalizationOverwrites`)
- Attributes:

Name	Type	Default value	Description
lang	string		
origChr	string		
replaceString	string		

NormalizationAlternatives

- `com.exalead.mot.components.normalizer.NormalizationAlternatives`
- No documentation for this element.
- Nested elements:

Name	Type	Description
NormalizationAlte	<code>com.exalead.mot.components.normalizer.N</code>	nativ

NormalizationAlternative

- `com.exalead.mot.components.normalizer.NormalizationAlternative`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.normalizer.NormalizationAlternatives` (as `NormalizationAlternatives`)
- Attributes:

Name	Type	Default value	Description
lang	string		
origChr	string		

Name	Type	Default value	Description
replaceString	string		

NormalizationExceptions

- `com.exalead.mot.components.normalizer.NormalizationExceptions`
- No documentation for this element.
- Nested elements:

Name	Type	Description
NormalizationExce	<code>com.exalead.mot.components.normalizer.N</code>	tion*

NormalizationException

- `com.exalead.mot.components.normalizer.NormalizationException`
- set normalization exception: for this word, the annotation is added with a trust level of 0 instead of 100. This is useful to index thé, maïs, ... as lowercase word and as normalized words
- Parent elements:
 - `com.exalead.mot.components.normalizer.NormalizationExceptions` (as `NormalizationExceptions`)
- Attributes:

Name	Type	Default value	Description
lang	string		
word	string		

RegexpMatches

- `com.exalead.mot.components.regexpmatcher.RegexpMatches`
- No documentation for this element.
- Attributes:

Name	Type	Default value	Description
modifiedBy	string		

Name	Type	Default value	Description
modifiedAt	nullablelong		

- Nested elements:

Name	Type	Description
RegexMatch	com.exalead.mot.components.regexmatcher.RegexpMatch	Regular expressions to recognize.

RegexMatch

- `com.exalead.mot.components.regexmatcher.RegexpMatch`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.regexmatcher.RegexpMatches` (as `RegexpMatches`)
- Attributes:

Name	Type	Default value	Description
regexp	string		Regular expression to recognize.
annotation	string		Tag of the annotations to add on matched tokens.
level	enum(exact, lowercase, normalized)	normalized	Level of the regular expression. For example, a regexp with <code>level=lowercase</code> matches case-insensitively.
lang	iso code	xx	Lang of the regular expression. The regexp does not match token in other languages. If <code>lang=xx</code> , token may be matched, no matter the language they are in.
displayForm	string		Value of the annotations to add on matched tokens. By default, the display form is that of the matched tokens, but the user can override it. Captures may be used. For

Name	Type	Default value	Description
			example if the regexp is "(foo)bar", and the displayForm is "\1baz", then annotations are added on "foobar" with value "foobaz".

SemanticExtractorConfig

- `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
- No documentation for this element.
- Attributes:

Name	Type	Default value	Description
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>lettersForDist1</code>	<code>int</code>	4	number of letters to enable maxDist=1 (letter level)
<code>lettersForDist2</code>	<code>int</code>	8	number of letters to enable maxDist=2 (letter level)
<code>wordsForDist1</code>	<code>int</code>	3	number of words to enable maxDist=1 (word level)
<code>wordsForDist2</code>	<code>int</code>	5	number of words to enable maxDist=2 (word level)

- Nested elements:

Name	Type	Description
<code>AbstractEntity</code>	<code>com.exalead.mot.components.semanticextr</code>	ity*
<code>Define</code>	<code>com.exalead.mot.components.semanticextr</code>	
<code>Include</code>	<code>com.exalead.mot.components.semanticextr</code>	
<code>Rule</code>	<code>com.exalead.mot.components.semanticextr</code>	

Entity

- `com.exalead.mot.components.semanticextractor.Entity`
- backward compatibility (?), must be mapped to `TextEntity` by the builder
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)
- Attributes:

Name	Type	Default value	Description
<code>name</code>	<code>string</code>		Unique entity reference. Only used by the GUI.
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>type</code>	<code>string</code>		
<code>entity</code>	<code>string</code>		
<code>output</code>	<code>string</code>		
<code>matchMode</code>	<code>string</code>	<code>normalized</code>	

TextEntity

- `com.exalead.mot.components.semanticextractor.TextEntity`
- Matches strings
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)
- Attributes:

Name	Type	Default value	Description
<code>unit</code>	<code>string</code>		An optional condition on right context
<code>rightContext</code>	<code>string</code>		Alias for unit

Name	Type	Default value	Description
trigger	string		An optional left context triggering the match
leftContext	string		Alias for trigger
annotation	string		The annotation set in case of a match
display	string		The annotation value in case of a match
matchMode	string	normalized	Match level
name	string		Unique entity reference. Only used by the GUI.
modifiedBy	string		
modifiedAt	nullablelong		
value	string		The values to match, separated with ' '
maxValueSize	int	10	Maximum number of tokens for the value
lang	iso code	xx	Optional language constraint

- Nested elements:

Name	Type	Description
units	exa.bee.StringValue*	An optional list of conditions on right context
rightContexts	exa.bee.StringValue*	Alias for units
triggers	exa.bee.StringValue*	An optional list of left contexts triggering the match
leftContexts	exa.bee.StringValue*	Alias for triggers

BooleanEntity

- `com.exalead.mot.components.semanticextractor.BooleanEntity`
- Matches binary values true/false, yes/no, ...
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)

- Attributes:

Name	Type	Default value	Description
unit	string		An optional condition on right context
rightContext	string		Alias for unit
trigger	string		An optional left context triggering the match
leftContext	string		Alias for trigger
annotation	string		The annotation set in case of a match
display	string		The annotation value in case of a match
matchMode	string	normalized	Match level
name	string		Unique entity reference. Only used by the GUI.
modifiedBy	string		
modifiedAt	nullablelong		
yes	string		The value for true
no	string		The value for false

- Nested elements:

Name	Type	Description
units	<code>exa.bee.StringValue*</code>	An optional list of conditions on right context
rightContexts	<code>exa.bee.StringValue*</code>	Alias for units
triggers	<code>exa.bee.StringValue*</code>	An optional list of left contexts triggering the match
leftContexts	<code>exa.bee.StringValue*</code>	Alias for triggers

IntegerEntity

- `com.exalead.mot.components.semanticextractor.IntegerEntity`
- Matches integral numbers
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)
- Attributes:

Name	Type	Default value	Description
unit	<code>string</code>		An optional condition on right context
rightContext	<code>string</code>		Alias for unit
trigger	<code>string</code>		An optional left context triggering the match
leftContext	<code>string</code>		Alias for trigger
annotation	<code>string</code>		The annotation set in case of a match
display	<code>string</code>		The annotation value in case of a match
matchMode	<code>string</code>	<code>normalized</code>	Match level

Name	Type	Default value	Description
name	string		Unique entity reference. Only used by the GUI.
modifiedBy	string		
modifiedAt	nullablelong		
step	int		If defined, normalizes the value to the nearest step
min	int	-2147483648	Minimum value for a match
max	int	2147483647	Maximum value for a match
coefficient	double	1	The normalization coefficient used to multiply the matched value
precision	int		The floating-point precision in output
handleOutOfBoundsValue	boolean	True	If false, ignores values lower than min or greater than max
addExactValue	boolean		If true, adds the original value (before normalization) in an extra annotation TAG.exact
truncateTrailingZeros	boolean		If true, removes the trailing zeros after point

- Nested elements:

Name	Type	Description
units	exa.bee.StringValue*	An optional list of conditions on right context
rightContexts	exa.bee.StringValue*	Alias for units
triggers	exa.bee.StringValue*	An optional list of left contexts triggering the match
leftContexts	exa.bee.StringValue*	Alias for triggers

FloatingPointEntity

- `com.exalead.mot.components.semanticextractor.FloatingPointEntity`
- Matches floating-point numbers
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)
- Attributes:

Name	Type	Default value	Description
<code>unit</code>	<code>string</code>		An optional condition on right context
<code>rightContext</code>	<code>string</code>		Alias for unit
<code>trigger</code>	<code>string</code>		An optional left context triggering the match
<code>leftContext</code>	<code>string</code>		Alias for trigger
<code>annotation</code>	<code>string</code>		The annotation set in case of a match
<code>display</code>	<code>string</code>		The annotation value in case of a match
<code>matchMode</code>	<code>string</code>	<code>normalized</code>	Match level
<code>name</code>	<code>string</code>		Unique entity reference. Only used by the GUI.
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>step</code>	<code>double</code>		If defined, normalizes the value to the nearest step
<code>min</code>	<code>double</code>	<code>-2147483648</code>	Minimum value for a match
<code>max</code>	<code>double</code>	<code>2147483647</code>	Maximum value for a match
<code>coefficient</code>	<code>double</code>	<code>1</code>	The normalization coefficient used to multiply the matched value

Name	Type	Default value	Description
precision	int	2	The floating-point precision in output
handleOutOfBoundValue	boolean	True	If false, ignores values lower than min or greater than max
addExactValue	boolean		If true, adds the original value (before normalization) in an extra annotation TAG.exact
truncateTrailingZeros	boolean		If true, removes the trailing zeros after point

- Nested elements:

Name	Type	Description
units	exa.bee.StringValue*	An optional list of conditions on right context
rightContexts	exa.bee.StringValue*	Alias for units
triggers	exa.bee.StringValue*	An optional list of left contexts triggering the match
leftContexts	exa.bee.StringValue*	Alias for triggers

RangeEntity

- `com.exalead.mot.components.semanticextractor.RangeEntity`
- Matches n-dimension integral numbers
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig` (as `SemanticExtractorConfig`)
- Attributes:

Name	Type	Default value	Description
unit	string		An optional condition on right context

Name	Type	Default value	Description
rightContext	string		Alias for unit
trigger	string		An optional left context triggering the match
leftContext	string		Alias for trigger
annotation	string		The annotation set in case of a match
display	string		The annotation value in case of a match
matchMode	string	normalized	Match level
name	string		Unique entity reference. Only used by the GUI.
modifiedBy	string		
modifiedAt	nullablelong		
dimension	int	2	Dimension count
delimiter	string	x	Numbers delimiter

- Nested elements:

Name	Type	Description
units	exa.bee.StringValue*	An optional list of conditions on right context
rightContexts	exa.bee.StringValue*	Alias for units
triggers	exa.bee.StringValue*	An optional list of left contexts triggering the match
leftContexts	exa.bee.StringValue*	Alias for triggers

RegexpEntity

- `com.exalead.mot.components.semanticextractor.RegexpEntity`

- Matches a regular expression
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)
- Attributes:

Name	Type	Default value	Description
<code>trigger</code>	<code>string</code>		An optional left context triggering the match
<code>leftContext</code>	<code>string</code>		Alias for trigger
<code>annotation</code>	<code>string</code>		The annotation set in case of a match
<code>display</code>	<code>string</code>		The annotation value in case of a match
<code>matchMode</code>	<code>string</code>	<code>normalized</code>	Match level
<code>name</code>	<code>string</code>		Unique entity reference. Only used by the GUI.
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>value</code>	<code>string</code>		Perl-5 regular expression
<code>endTrigger</code>	<code>string</code>		An optional left context

- Nested elements:

Name	Type	Description
<code>triggers</code>	<code>exa.bee.StringValue*</code>	An optional list of left contexts triggering the match
<code>leftContexts</code>	<code>exa.bee.StringValue*</code>	Alias for triggers
<code>endTriggers</code>	<code>exa.bee.StringValue*</code>	An optional list of left contexts

Define

- `com.exalead.mot.components.semanticextractor.Define`
- Macro definition
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)
- Attributes:

Name	Type	Default value	Description
name	string		
value	string		

Include

- `com.exalead.mot.components.semanticextractor.Include`
- Includes an external `SemanticExtractorConfig` rules file
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)
- Attributes:

Name	Type	Default value	Description
path	string		The absolute path to the <code>SemanticExtractorConfig</code> XML file.

Rule

- `com.exalead.mot.components.semanticextractor.Rule`
- A semantic extraction rule.
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.SemanticExtractorConfig`
(as `SemanticExtractorConfig`)

- Attributes:

Name	Type	Default value	Description
modifiedBy	string		
modifiedAt	nullablelong		
id	string		Unique rule reference. Only used by the GUI.
name	string		Rule name.
mode	enum (match, filter)	match	Rule mode.
pattern	string		Rule pattern
output	string		Rule output name.
unusedWordPolicy	string		Unused query word policy. If defined, overrides general value when this rule fires
trustLevel	int	100	Rule trust level. A percentage showing this rule's relevance compared to others.
language	string		Which language the rule must be applied to.
value	string		Rule value.

Synonyms

- com.exalead.mot.grewrite.v10.Synonyms
- The Synonyms feature allows you to define synonym resources
- Attributes:

Name	Type	Default value	Description
modifiedBy	string		
modifiedAt	nullablelong		
equivalenceClass	boolean	True	A synonym set is defined with a master expression and a set

Name	Type	Default value	Description
			of associated expressions (an expression is defined by many words with a space as separator) originalExpr = {alternativeExpr1, ..., alternativeExprN}. When query is parsed we expand originalExpr with {alternativeExpr1, ..., alternativeExprN}. When equivalenceClass Boolean is set to true, we also expand: - alternativeExpr1 by originalExpr, alternativeExpr2, ..., alternativeExprN - alternativeExpr2 by originalExpr, alternativeExpr1, alternativeExpr3, ..., alternativeExprN - ... - alternativeExprN by originalExpr, alternativeExpr1, ..., alternativeExprN-1
matchOnSeparators	boolean	True	If false, synonym matching is punctuation-insensitive.
stopwordsResource	string	resource:///stopwords/ontology.bin	Path to the compiled ontology containing stopwords used at build time when generating permutations.
permutations	boolean		If true, adds for each synonym some extra forms made of words permutations after removing stopwords.
addStopwordFreeFor	boolean		If true, adds for each synonym an extra form from which stopwords have been removed.

- Nested elements:

Name	Type	Description
SynonymSet	com.exalead.mot.qrewrite.v10.SynonymSet	

SynonymSet

- `com.exalead.mot.grewrite.v10.SynonymSet`
- A set of synonym terms.
- Parent elements:
 - `com.exalead.mot.grewrite.v10.Synonyms` (**as** Synonyms)
- Attributes:

Name	Type	Default value	Description
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>originalExpr</code>	<code>string</code>		String to match in user query
<code>level</code>	<code>enum(exact, lowercase, normalized)</code>	<code>normalized</code>	Term level of these expressions.
<code>lang</code>	<code>iso code</code>		Only match the original expression in this language.
<code>equivalenceClass</code>	<code>enum(true, false, SynonymSetToSy, SynonymToSynor</code>		override equivalenceClass Boolean if null or different from true/false, keep Synonyms.equivalenceClass value

- Nested elements:

Name	Type	Description
<code>Synonym</code>	<code>com.exalead.mot.grewrite.v10.Synonym*</code>	The list of synonyms of the original expression.

Synonym

- `com.exalead.mot.grewrite.v10.Synonym`
- No documentation for this element.
- Parent elements:

- `com.exalead.mot.grewrite.v10.SynonymSet (as SynonymSet)`

- **Attributes:**

Name	Type	Default value	Description
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>alternativeExpr</code>	<code>string</code>		Synonym string: can be a multiword expression (separated by space only)
<code>level</code>	<code>enum(exact, lowercase, normalized, custom, sameasset)</code>	<code>sameasset</code>	display level
<code>customLevel</code>	<code>byte</code>		(only when <code>level == "custom"</code>) when <code>level = "custom"</code> , this <code>customLevel</code> is used
<code>distance</code>	<code>int</code>		distance applied to this synonym
<code>lang</code>	<code>string</code>		if not null, override language of <code>SynonymSet</code>

TRules

- `com.exalead.mot.components.transducer.TRules`

- A set of transducer rules

- **Attributes:**

Name	Type	Default value	Description
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		

- **Nested elements:**

Name	Type	Description
<code>RulePattern</code>	<code>com.exalead.mot.components.transducer.R</code>	

Seq

- `com.exalead.mot.components.transducer.Seq`
- **Abstract class common to all patterns whose children have to be interpreted as a sequence.**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`

- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
<code>priority</code>	<code>int</code>		
<code>name</code>	<code>string</code>		Optional name of the pattern (used for pattern references).
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		

- **Nested elements:**

Name	Type	Description
<code>RulePattern</code>	<code>com.exalead.mot.components.transducer.R</code>	

Iter

- `com.exalead.mot.components.transducer.Iter`
- `@b iter pattern ...`
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`

- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- Attributes:

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
min	int		
max	int	128	
greedy	boolean	True	

- Nested elements:

Name	Type	Description
RulePattern	com.exalead.mot.components.transducer.R	

Star

- com.exalead.mot.components.transducer.Star
- @b star pattern == iter(min=0, max=this.max, greedy=true)
- Parent elements:
 - com.exalead.mot.components.transducer.And (as And)
 - com.exalead.mot.components.transducer.And (as And)
 - com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)
 - com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)
 - com.exalead.mot.components.transducer.Iter (as Iter)
 - com.exalead.mot.components.transducer.Iter (as Iter)
 - com.exalead.mot.components.transducer.Near (as Near)
 - com.exalead.mot.components.transducer.Near (as Near)
 - com.exalead.mot.components.transducer.Nor (as Nor)

- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

Name	Type	Default value	Description
max	int	2147483647	

- Nested elements:

Name	Type	Description
RulePattern	com.exalead.mot.components.transducer.R	

Plus

- `com.exalead.mot.components.transducer.Plus`
- `@b plus pattern == iter(min=1, max=this.max, greedy=true)`
- Parent elements:
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`

- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
max	int	2147483647	

- **Nested elements:**

Name	Type	Description
RulePattern	<code>com.exalead.mot.components.transducer.R</code>	

Opt

- `com.exalead.mot.components.transducer.Opt`

- `@b opt pattern == iter(min=0, max=1, greedy=true)`
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
 - `com.exalead.mot.components.transducer.Star (as Star)`
 - `com.exalead.mot.components.transducer.Star (as Star)`
 - `com.exalead.mot.components.transducer.Sub (as Sub)`

- `com.exalead.mot.components.transducer.Sub` (**as** `Sub`)
- `com.exalead.mot.components.transducer.TRule` (**as** `TRule`)
- `com.exalead.mot.components.transducer.TRule` (**as** `TRule`)
- `com.exalead.mot.components.transducer.TRules` (**as** `TRules`)
- `com.exalead.mot.components.transducer.TRules` (**as** `TRules`)

- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

- **Nested elements:**

Name	Type	Description
RulePattern	<code>com.exalead.mot.components.transducer.R</code>	

Sub

- `com.exalead.mot.components.transducer.Sub`
- **@b** sub pattern denotes submatches that are retrieved.
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And` (**as** `And`)
 - `com.exalead.mot.components.transducer.And` (**as** `And`)
 - `com.exalead.mot.components.transducer.DisjunctivePattern` (**as** `DisjunctivePattern`)
 - `com.exalead.mot.components.transducer.DisjunctivePattern` (**as** `DisjunctivePattern`)
 - `com.exalead.mot.components.transducer.Iter` (**as** `Iter`)
 - `com.exalead.mot.components.transducer.Iter` (**as** `Iter`)
 - `com.exalead.mot.components.transducer.Near` (**as** `Near`)

- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).

Or

Name	Type	Default value	Description
modifiedBy	string		
modifiedAt	nullablelong		
no	int		
kind	string	sub	
value	string		
trustLevel	int	100	

- Nested elements:

Name	Type	Description
RulePattern	com.exalead.mot.components.transducer.R	

Or

- `com.exalead.mot.components.transducer.Or`
- `@b` or pattern matches ...
- Parent elements:
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`

- `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
 - `com.exalead.mot.components.transducer.Star (as Star)`
 - `com.exalead.mot.components.transducer.Star (as Star)`
 - `com.exalead.mot.components.transducer.Sub (as Sub)`
 - `com.exalead.mot.components.transducer.Sub (as Sub)`
 - `com.exalead.mot.components.transducer.TRule (as TRule)`
 - `com.exalead.mot.components.transducer.TRule (as TRule)`
 - `com.exalead.mot.components.transducer.TRules (as TRules)`
 - `com.exalead.mot.components.transducer.TRules (as TRules)`
- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

- **Nested elements:**

Name	Type	Description
RulePattern	<code>com.exalead.mot.components.transducer.R</code>	

Near

- `com.exalead.mot.components.transducer.Near`
- A BINARY near matching subexpressions in any order at a max distance defined by slop in terms of nonblank tokens
- Parent elements:
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`

- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
<code>priority</code>	<code>int</code>		
<code>name</code>	<code>string</code>		Optional name of the pattern (used for pattern references).
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		
<code>slop</code>	<code>int</code>		
<code>ordered</code>	<code>boolean</code>		

- **Nested elements:**

Name	Type	Description
<code>RulePattern</code>	<code>com.exalead.mot.components.transducer.R</code>	

Noblink

- `com.exalead.mot.components.transducer.Noblink`
- **Assert that there is no space between two tokens.**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`

- `com.exalead.mot.components.transducer.And (as And)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`

- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

PatternRef

- `com.exalead.mot.components.transducer.PatternRef`
- **Abstract class common to pattern that matches a word or an annotation.**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`

- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

And

- `com.exalead.mot.components.transducer.And`
- **Abstract class common to pattern that matches a word or an annotation.**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`

- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

- **Nested elements:**

Name	Type	Description
RulePattern	<code>com.exalead.mot.components.transducer.R</code>	

Not

- `com.exalead.mot.components.transducer.Not`
- Abstract class common to pattern that matches a word or an annotation.
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`

- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

- **Nested elements:**

Name	Type	Description
RulePattern	<code>com.exalead.mot.components.transducer.R</code>	

Nor

- `com.exalead.mot.components.transducer.Nor`
- **Abstract class common to pattern that matches a word or an annotation.**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`

- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

- Nested elements:

Name	Type	Description
RulePattern	com.exalead.mot.components.transducer.R	

TokenKind

- `com.exalead.mot.components.transducer.TokenKind`
- Matches a specific token kind as set by the tokenizer Allowed values are: SEP_PARAGRAPH SEP_SENTENCE SEP_PUNCT SEP_QUOTE SEP_DASH NUMBER ALPHANUM (** Warning **, this means alpha and num, not alpha or num) ALPHA.
- Parent elements:
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`

- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
value	string		

Paragraph

- `com.exalead.mot.components.transducer.Paragraph`
- Matches a token with kind `SEP_PARAGRAPH`
- Parent elements:
 - `com.exalead.mot.components.transducer.And (as And)`

- `com.exalead.mot.components.transducer.And (as And)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`

- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

Sentence

- `com.exalead.mot.components.transducer.Sentence`
- **Matches a token with kind SEP_SENTENCE**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`

- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

Dash

- `com.exalead.mot.components.transducer.Dash`
- **Matches a token with kind SEP_DASH**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`

- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

Punct

- `com.exalead.mot.components.transducer.Punct`
- **Matches a token with kind SEP_PUNCT**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`

- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

Digits

- `com.exalead.mot.components.transducer.Digits`
- **Matches a token with kind NUMBER**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`

- `com.exalead.mot.components.transducer.Or` (**as** `Or`)
- `com.exalead.mot.components.transducer.Plus` (**as** `Plus`)
- `com.exalead.mot.components.transducer.Plus` (**as** `Plus`)
- `com.exalead.mot.components.transducer.Seq` (**as** `Seq`)
- `com.exalead.mot.components.transducer.Seq` (**as** `Seq`)
- `com.exalead.mot.components.transducer.SequentialPattern` (**as** `SequentialPattern`)
- `com.exalead.mot.components.transducer.SequentialPattern` (**as** `SequentialPattern`)
- `com.exalead.mot.components.transducer.Star` (**as** `Star`)
- `com.exalead.mot.components.transducer.Star` (**as** `Star`)
- `com.exalead.mot.components.transducer.Sub` (**as** `Sub`)
- `com.exalead.mot.components.transducer.Sub` (**as** `Sub`)
- `com.exalead.mot.components.transducer.TRule` (**as** `TRule`)
- `com.exalead.mot.components.transducer.TRule` (**as** `TRule`)
- `com.exalead.mot.components.transducer.TRules` (**as** `TRules`)
- `com.exalead.mot.components.transducer.TRules` (**as** `TRules`)

- **Attributes:**

Name	Type	Default value	Description
<code>anchor</code>	<code>boolean</code>		
<code>priority</code>	<code>int</code>		
<code>name</code>	<code>string</code>		Optional name of the pattern (used for pattern references).
<code>modifiedBy</code>	<code>string</code>		
<code>modifiedAt</code>	<code>nullablelong</code>		

Alnum

- `com.exalead.mot.components.transducer.Alnum`
- **Matches a token made only of letters or digits (case-insensitive)**

- **Parent elements:**

- `com.exalead.mot.components.transducer.And (as And)`
- `com.exalead.mot.components.transducer.And (as And)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`

- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

Alpha

- `com.exalead.mot.components.transducer.Alpha`
- **Matches a token made only of letters (case-insensitive)**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`

- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

TokenLanguage

- `com.exalead.mot.components.transducer.TokenLanguage`
- **Matches a token with specified language**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`

- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
value	string		

AnyToken

- `com.exalead.mot.components.transducer.AnyToken`
- **Matches any word.**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`

- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- Attributes:

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

TokenRegexp

- `com.exalead.mot.components.transducer.TokenRegexp`
- Matches a regexp.
- Parent elements:
 - `com.exalead.mot.components.transducer.And` (**as** And)
 - `com.exalead.mot.components.transducer.And` (**as** And)
 - `com.exalead.mot.components.transducer.DisjunctivePattern` (**as** DisjunctivePattern)
 - `com.exalead.mot.components.transducer.DisjunctivePattern` (**as** DisjunctivePattern)
 - `com.exalead.mot.components.transducer.Iter` (**as** Iter)
 - `com.exalead.mot.components.transducer.Iter` (**as** Iter)
 - `com.exalead.mot.components.transducer.Near` (**as** Near)
 - `com.exalead.mot.components.transducer.Near` (**as** Near)
 - `com.exalead.mot.components.transducer.Nor` (**as** Nor)
 - `com.exalead.mot.components.transducer.Nor` (**as** Nor)
 - `com.exalead.mot.components.transducer.Not` (**as** Not)
 - `com.exalead.mot.components.transducer.Not` (**as** Not)
 - `com.exalead.mot.components.transducer.Opt` (**as** Opt)
 - `com.exalead.mot.components.transducer.Opt` (**as** Opt)

- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
value	string		Regular expression to recognize.
level	enum(exact, lowercase, normalized)	exact	Level of the regular expression. For example, a regexp with

Name	Type	Default value	Description
			level=lowercase matches case-insensitively.

Word

- `com.exalead.mot.components.transducer.Word`
- Matches a word.
- Parent elements:
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Not (as Not)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Opt (as Opt)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Or (as Or)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Plus (as Plus)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`
 - `com.exalead.mot.components.transducer.Seq (as Seq)`

- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
value	string		Word to recognize.
level	enum(exact, lowercase, normalized)	normalized	Level of the word. For example, a word with level=lowercase matches case-insensitively.

Annotation

- `com.exalead.mot.components.transducer.Annotation`
- Matches an annotation kind and possibly its display form if specified.
- Parent elements:

- `com.exalead.mot.components.transducer.And (as And)`
- `com.exalead.mot.components.transducer.And (as And)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`

- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
kind	string		
value	string		
useDisplayForm	boolean	True	
required	boolean		
error	enum(ignore, warn, error)	warn	

Ctx

- `com.exalead.mot.components.transducer.Ctx`
- **backward compatibility classes**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`

- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
kind	string		

Name	Type	Default value	Description
value	string		
useDisplayForm	boolean	True	
required	boolean		
error	enum(ignore, warn, error)	warn	
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		

AnnotationRegexp

- `com.exalead.mot.components.transducer.AnnotationRegexp`
- Matches an annotation with specified kind and whose display form matches the specified regular expression. If defined, "capture" contains an output format a la sed used for generating the final match annotation
- Parent elements:
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`

- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
anchor	boolean		
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		

Name	Type	Default value	Description
modifiedAt	nullablelong		
kind	string		Annotation kind that the annotation regular expression is tested against.
value	string		Regular expression of the annotation.
level	enum(exact, lowercase, normalized)	exact	Matching level, can be normalized, lowercase, or exact.
useDisplayForm	boolean	True	Activate this option to use the display form of this annotation to build the final output annotation.
required	boolean		
error	enum(ignore, warn, error)	warn	
capture	string		Output format in sed format used to generate the final match annotation

TRule

- `com.exalead.mot.components.transducer.TRule`
- **A transducer rule**
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`

- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).

Name	Type	Default value	Description
modifiedBy	string		
modifiedAt	nullablelong		
kind	string		
value	string		
trustLevel	int	100	

- Nested elements:

Name	Type	Description
MatchAnnotation	com.exalead.mot.components.transducer.M	
RulePattern	com.exalead.mot.components.transducer.R	

MatchAnnotation

- `com.exalead.mot.components.transducer.MatchAnnotation`
- Match generation An annotation kind and a format
- Parent elements:
 - `com.exalead.mot.components.transducer.TRule` (**as** `TRule`)
 - `com.exalead.mot.components.transducer.TRule` (**as** `TRule`)
- Attributes:

Name	Type	Default value	Description
kind	string		
value	string	%0	
trustLevel	int	-1	

TInclude

- `com.exalead.mot.components.transducer.TInclude`
- Include an XML rules file
- Parent elements:

- `com.exalead.mot.components.transducer.And (as And)`
- `com.exalead.mot.components.transducer.And (as And)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Iter (as Iter)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Near (as Near)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Nor (as Nor)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`

- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
filename	string		

TImport

- `com.exalead.mot.components.transducer.TImport`
- Import an XML rules file so that rules and patterns defined in it can be referenced by a `PatternRef`. This is not quite equivalent to a `TInclude`, which contains at least a `TRule`. Here, the `TRule` is not created.
- **Parent elements:**
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.And (as And)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.DisjunctivePattern (as DisjunctivePattern)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Iter (as Iter)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Near (as Near)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`
 - `com.exalead.mot.components.transducer.Nor (as Nor)`

- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Not (as Not)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Opt (as Opt)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Or (as Or)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Plus (as Plus)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.Seq (as Seq)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.SequentialPattern (as SequentialPattern)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Star (as Star)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.Sub (as Sub)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRule (as TRule)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`
- `com.exalead.mot.components.transducer.TRules (as TRules)`

- **Attributes:**

Name	Type	Default value	Description
priority	int		
name	string		Optional name of the pattern (used for pattern references).
modifiedBy	string		
modifiedAt	nullablelong		
filename	string		

Remove

- `com.exalead.linguistic.v10.Remove`
- Removes the specified annotations, possibly when some condition is met
- Attributes:

Name	Type	Default value	Description
<code>annotation</code>	<code>string</code>		Tag of the annotations to be removed
<code>ifMatchWith</code>	<code>string</code>		Removes the annotation if the annotated text span matches that of this one
<code>ifOverlapWith</code>	<code>string</code>		Removes the annotation if it overlaps with this one
<code>displayFormsMustMatch</code>	<code>boolean</code>		If <code>ifMatchWith</code> is <code>TRUE</code> , removes the annotation only if both display forms match

Copy

- `com.exalead.linguistic.v10.Copy`
- Copies a source annotation along with its display form and display kind to a target annotation.
- Attributes:

Name	Type	Default value	Description
<code>annotation</code>	<code>string</code>		The source annotation to be copied
<code>target</code>	<code>string</code>		The target annotation
<code>unless</code>	<code>string</code>		Copies the source annotation unless this annotation is present

KeepLongestLeftMost

- `com.exalead.linguistic.v10.KeepLongestLeftMost`

- When several annotations overlap, it keeps the longest (removes all others); if there are several longest annotations, then keep the leftmost ones. For example, there are 5 tokens "tow truck driver license requirements" and 3 annotations on "tow truck driver", "truck driver license requirements" and "license requirements" with the same tag. It keeps the annotation on "truck driver license requirements" and removes the other two.
- Attributes:

Name	Type	Default value	Description
annotations	string		List of comma-separated annotations to process
interTags	boolean		Keep the longest-leftmost among all the tags. If FALSE, one annotation per tag is kept.

- Nested elements:

Name	Type	Description
AnnotationProcessed	com.exalead.linguistic.v10.AnnotationPr	Alternative way to specify the list of annotations to process

AnnotationProcessed

- com.exalead.linguistic.v10.AnnotationProcessed
- Alternative way to specify the list of annotations to be processed by the operation `KeepLongestLeftMost`
- Parent elements:
 - com.exalead.linguistic.v10.KeepLeftMostLongest (as `KeepLeftMostLongest`)
 - com.exalead.linguistic.v10.KeepLongestLeftMost (as `KeepLongestLeftMost`)
- Attributes:

Name	Type	Default value	Description
name	string		

KeepLeftMostLongest

- `com.exalead.linguistic.v10.KeepLeftMostLongest`
- When several annotations overlap, it keeps the leftmost (removes all others); if there are several leftmost annotations, then keep the longest ones. For example, there are 5 tokens "tow truck driver license requirements" and 3 annotations on "tow truck driver", "truck driver license requirements" and "license requirements" with the same tag. It keeps the annotations on "tow truck driver" and "license requirements".
- Attributes:

Name	Type	Default value	Description
annotations	string		List of comma-separated annotations to process
interTags	boolean		Keep the leftmost-longest among all the tags. If FALSE, one annotation per tag is kept

- Nested elements:

Name	Type	Description
AnnotationProcess	<code>com.exalead.linguistic.v10.AnnotationPr</code>	Alternative way to specify the list of annotations to process

KeepFirst

- `com.exalead.linguistic.v10.KeepFirst`
- Selects the first N occurrences or values of an annotation and remove all others
- Attributes:

Name	Type	Default value	Description
annotation	string		The annotation to find
contexts	string		Keeps the first N annotation occurrences/values in each of these contexts

Name	Type	Default value	Description
howMany	int	1	How many annotation occurrences/values are kept
what	enum (occurrences, values)		Defines what must be kept: first N annotation 'occurrences' or 'values'.

SelectMostFrequentValue

- `com.exalead.linguistic.v10.SelectMostFrequentValue`
- Selects the N most frequent values of a given annotation and annotates the document with them.
- Attributes:

Name	Type	Default value	Description
annotation	string		The annotation to find
documentAnnotation	string		Annotates the document with this annotation instead of the selected annotation
truncate	boolean		Keeps only one value when there are multiple candidates
howMany	int	1	How many values must be kept

SelectMostFrequentAnnotation

- `com.exalead.linguistic.v10.SelectMostFrequentAnnotation`
- Selects the most frequent annotation and annotates the document with it.
- Attributes:

Name	Type	Default value	Description
annotations	string		Selects among these annotations
documentAnnotation	string		Annotates the document with this annotation

SelectByContexts

- `com.exalead.linguistic.v10.SelectByContexts`
- Selects annotations appearing in the first context of a list sorted by decreasing priority. For example, selecting an annotation from (title, text) looks up title context and then, if the annotation is not found, text context.
- Attributes:

Name	Type	Default value	Description
annotation	string		The annotation to find
contexts	string		The contexts to select the annotation from, sorted by decreasing priority
documentAnnotation	string		Annotates the document with this annotation
firstOnly	boolean		Selects only the first occurrence of the annotation

StringValue

- `exa.bee.StringValue`
- No documentation for this element.
- Parent elements:
 - `com.exalead.mot.components.semanticextractor.RegexpEntity` (as `endTriggers`)
 - `com.exalead.mot.components.semanticextractor.BooleanEntity` (as `leftContexts`)
 - `com.exalead.mot.components.semanticextractor.FloatingPointEntity` (as `leftContexts`)
 - `com.exalead.mot.components.semanticextractor.IntegerEntity` (as `leftContexts`)
 - `com.exalead.mot.components.semanticextractor.RangeEntity` (as `leftContexts`)

- `com.exalead.mot.components.semanticextractor.RegexpEntity (as leftContexts)`
- `com.exalead.mot.components.semanticextractor.SimpleEntity (as leftContexts)`
- `com.exalead.mot.components.semanticextractor.TextEntity (as leftContexts)`
- `com.exalead.mot.components.semanticextractor.ValuedEntity (as leftContexts)`
- `com.exalead.mot.components.semanticextractor.BooleanEntity (as rightContexts)`
- `com.exalead.mot.components.semanticextractor.FloatingPointEntity (as rightContexts)`
- `com.exalead.mot.components.semanticextractor.IntegerEntity (as rightContexts)`
- `com.exalead.mot.components.semanticextractor.RangeEntity (as rightContexts)`
- `com.exalead.mot.components.semanticextractor.TextEntity (as rightContexts)`
- `com.exalead.mot.components.semanticextractor.ValuedEntity (as rightContexts)`
- `com.exalead.mot.components.semanticextractor.BooleanEntity (as triggers)`
- `com.exalead.mot.components.semanticextractor.FloatingPointEntity (as triggers)`
- `com.exalead.mot.components.semanticextractor.IntegerEntity (as triggers)`
- `com.exalead.mot.components.semanticextractor.RangeEntity (as triggers)`
- `com.exalead.mot.components.semanticextractor.RegexpEntity (as triggers)`
- `com.exalead.mot.components.semanticextractor.SimpleEntity (as triggers)`
- `com.exalead.mot.components.semanticextractor.TextEntity (as triggers)`

- `com.exalead.mot.components.semanticextractor.ValuedEntity (as triggers)`
 - `com.exalead.mot.components.semanticextractor.BooleanEntity (as units)`
 - `com.exalead.mot.components.semanticextractor.FloatingPointEntity (as units)`
 - `com.exalead.mot.components.semanticextractor.IntegerEntity (as units)`
 - `com.exalead.mot.components.semanticextractor.RangeEntity (as units)`
 - `com.exalead.mot.components.semanticextractor.TextEntity (as units)`
 - `com.exalead.mot.components.semanticextractor.ValuedEntity (as units)`
- **Attributes:**

Name	Type	Default value	Description
value	string		

Appendix - ELLQL Language

ELLQL (for EXALEAD Low Level Query Language) exposes all advanced features of the EXALEAD search and index stacks.

Internally, all UQL (for User Query Language) queries entered by users are transformed into ELLQL. ELLQL is mostly used by custom programs to enrich user queries. In the Administration Console, you can write ELLQL in the **Search Logics > Search Logic Name > Query Template** tab. You can also directly enter ELLQL in search fields, using the `eq` parameter instead of the `q` parameter (used for UQL queries).

ELLQL Language Features

Simple Operators

Compound Operators

ELLQL Language Features

Structure of the Language

ELLQL queries look like this:

```
#operator{options} (argument)
```

where:

- `operator` - specifies the type of the field on which the query is applied. For example, `operator` may have one of the following values (all possible values are described afterward): `#alphanum`, `#num`, `#category`, etc. `operator` may also represent the way fields are compounded (`#and`, `#near`, etc.).
- `options` - related to the operator. They may be written as a comma-separated list of values or `key=value`. `options` are optional.
- `argument` - may be any kind of string composed of: lower/upper case letters, digits, and underscore. The only restriction is that an argument cannot begin with a digit. For simple queries, `argument` generally represents the name of the field and the value of the query. `argument` may also be a query itself, and in that case, we talk of a compound operator.

Options

The following option is applicable to all ELLQL nodes:

`hl=0` - deactivates search result highlighting and summary for a specific node and its children if any.

An `operator` may take as option any **(key, value)** pair, in the form of **key=value**, where:

- **key** formatting is the same as for operator arguments (see above)
- and value may be a numerical value (int or float) or a string

Then, two cases may occur:

- **key** is a known option of operator. In that case, `operator` uses the **(key, value)** pair to perform its action. Known options are listed afterward for each operator. While these known options differ from one operator to another, the **(slice, int)** option is common to every operator. It restricts the scope of the query to a specific slice for the current node.
- **key** is not known. **(key, value)** is then called by a node property and may be used for scoring operations.

Simple Operators

Fields Search

Each type of field is associated with a query operator.

Alphanumeric Fields Operators

Alphanum

Syntax	Options	Examples
<code>#alphanum{alphanum_option (field, "term")}</code>	<ul style="list-style-type: none"> • k=int (required) - specifies on which field matching mode (indexing level) the operator is applied. k may take any value that is described within the <code>Linguistic.xml</code> configuration file. Classical ones are the following: <ul style="list-style-type: none"> ◦ 0 - exact indexing level, 	<p>Search the term <code>house</code> in the title normalized field (k=2)</p> <pre>#alphanum{k=2} (title, "house")</pre> <p>Search the term <code>house</code> in the title normalized field (k=2), knowing that there are only 3 documents in the corpus containing the term <code>house</code></p> <pre>#alphanum{k=2, nbdocs=3} (title, "house")</pre>

Syntax	Options	Examples
	<ul style="list-style-type: none"> ◦ 1 - lowercase indexing level, ◦ 2 - normalized indexing level. Note: In the Administration Console, the default value for every new field is 2, but you are free to define your own values in the <code>Linguistic.xml</code> file. • <code>nbdocs=int</code> - specifies the number of documents that contain the term. This value is then used to compute IDF (Inverse Document Frequency), which is required to calculate the TFIDF values used for ranking. Note: This value is generally automatically set when ELLQL is generated from UQL queries. • <code>source=value</code> - used by the debugger to specify the node's origin. It may be really helpful when debugging complex ELLQL expressions. 	

Anumpattern

Possible patterns:

- `prefix/suffix/substring` search
- `missing characters` search, for example, `?i?e` must match `bite` and `rime`. In that case, `?` is called the missing character marker.

- full pattern search, for example, `*?i*?e` must match `bite`, `rime`, `resize`, `linearize`, but not `image` nor `wheelie`. `*` is the wildcard.

Syntax	Options	Examples
<code>#anumpattern{options}(field, "term")</code>	<ul style="list-style-type: none"> • all the <code>alphanum_options</code> • <code>type={prefix, suffix, substring, missingChars, fullPattern}</code> - default is <code>fullPattern</code> • <code>mcmarker=CHAR</code> - default is <code>?</code> • <code>wildcard=CHAR</code> - default is <code>*</code> • <code>missingchars</code> <p>Note: pattern search is activated for a prefix handler when enabled at the data model level</p>	<p>Search all terms with substring <code>lot</code>, for example, <code>lottery</code>, <code>Camelot</code>, <code>slotted</code>, in the text normalized field (<code>k=2</code>):</p> <pre>#anumpattern{k=2,type="substring"(text, "lot")</pre> <p>Search all terms with missing chars <code>?i?e</code>, for example, <code>rime</code>, <code>rite</code>, <code>mime</code>, in the title normalized field (<code>k=2</code>):</p> <pre>#anumpattern{k=2,type="missingchars"(title, "?i?e")</pre> <p>or:</p> <pre>#anumpattern{k=2,type="missingchars",mcmarker="X"}(title, "XiXe")</pre>

Numerical Fields

Num

Syntax	Options	Examples
<code>#num{num_option}(field, operator, value)</code>	<p>Since fields may be multivalued, the <code>num_option</code> allows you to customize the comparison policy.</p> <p>The following options are accepted:</p> <ul style="list-style-type: none"> • <code>any</code> - If any value matches the condition. • <code>all</code> - If all values need to match the condition. 	<ul style="list-style-type: none"> • Search for documents with a size greater than 100. <pre>#num(document_file_size, >, 100)</pre> • Search for documents where the <code>prices</code> field value is contained between \$2.50 and \$9.99. <pre>#num(prices, <=>, 2.5, 9.99)</pre>

Syntax	Options	Examples
	<ul style="list-style-type: none"> <code>none</code> - If no value has to match the condition. <p>operator possible values:</p> <ul style="list-style-type: none"> Comparison operators: <code><</code>, <code><=</code>, <code>></code>, <code>>=</code>, <code>==</code> or <code>=</code> (equals to), <code>!=</code> Range operator that requires two values: the syntax then becomes <code>#num{num_options} (field, <=>, from_value, to_value).</code> 	<ul style="list-style-type: none"> Search for documents where one of the values of the <code>prices</code> multivalued field is equal to \$5. <code>#num{any} (prices, ==, 5)</code> Search for documents where all the values of the <code>prices</code> multivalued numerical field are under \$5. <code>#num{all} (prices, <=, 5)</code>

Around

Syntax	Options	
<code>#around{around_opt (field, value)</code>	<p>Search for document where the field value is around the value argument.</p> <ul style="list-style-type: none"> Same options as <code>#num</code> apply Additional options are <code>lowerBound</code> and <code>upperBound</code>, with defaults to negative infinity and positive infinity. Only values within these bounds can be returned. The generic <code>name</code> option allows you to calculate the difference, <code>diff</code>, between the value and the returned value. For example, if <code>#around{name=NAME} (field, 10)</code> returns 5, then <code>NAME.diff = -5</code>. 	<p>Search any value of <code>field</code> around 50 ranged between 0 and 100. The resulting <code>diff</code> between the found value and 50 is stored in the <code>around.diff</code>.</p> <pre>#around{lowerBound=0,upperBound=100 name="around"} (field, 50) <Hit> [...] <metas> <Meta name="field"> <MetaString name="value">75</Met </Meta> </metas> <infos> <HitInfo key="ranking.around.diff value="25"/> </infos> </Hit></pre>

Attrnum

Syntax	Use	Example
<code>#attrnum{options}(field, operator, value)</code>	Allows you to search NumericalFields that are not searchable but only retrievable and RAM-based.	Search for a numerical field with a value greater than 100 <code>#attrnum(myfield, > , 100)</code>
<code>#attrnum{options}(field, <=>, from_value, to_value)</code>	Allows you to search for ranges on NumericalFields; where field is the name of numerical field, from_value is the start value, and to_value the end value of the range.	Search for fields where the prices field value is contained between \$2.50 and \$9.99 <code>#attrnum(myfield, <=>, 2.5, 9.99)</code>
<code>#attrnum{options}(field, meta, operator, value)</code>	Allows you to search DynamicNumericalFields that are not searchable but only retrievable and RAM-based.	Search for documents where the "myfield" dynamic numerical field has a "mymeta" meta with a value greater than 100 <code>#attrnum(myfield, mymeta, > , 100)</code>
<code>#attrnum{options}(field, meta, <=>, from_value, to_value)</code>	Allows you to search for ranges on DynamicNumericalFields; where field is the name of dynamic numerical field, meta is the name of meta to compare, from_value is the start value, and to_value the end value of the range.	Search for documents where the "myfield" dynamic numerical field has a "mymeta" meta with a range of values between 20 and 100 <code>#attrnum(myfield, mymeta, <=> , 20, 100)</code>

Category Fields

Category

Syntax	Example
<code>#category{options}(field, "value")</code>	Search documents in the Top/Source/default category: <code>#category(categories, "Top/Source/default")</code>

Uid Fields

Uid

Syntax	Use
<code>#uid{options}(field, "value1" "value2" "value3")</code>	Search for documents where the Uid field value is one of value1, value2, value3. All possible values are listed separated with quotes.

Geographical Search

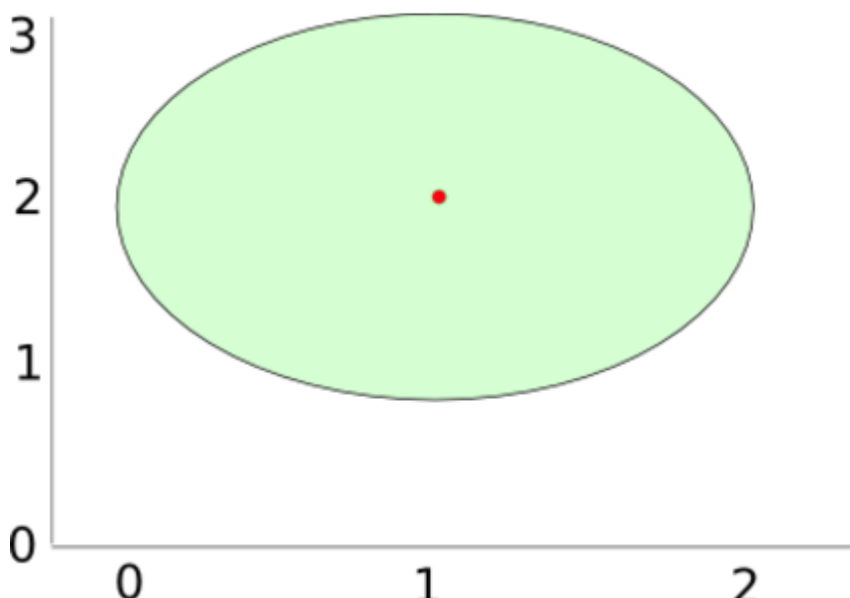
Geographical search may be applied as soon as fields of type **point** are stored in the index.

Distance

Syntax	Use
<code>#distance{options}(field, lat, lng, distance_in_meters)</code>	Retrieve all the documents with a field describing a position (a point) within distance_in_meters meters from (lat, lng)

Example:

The following search area:



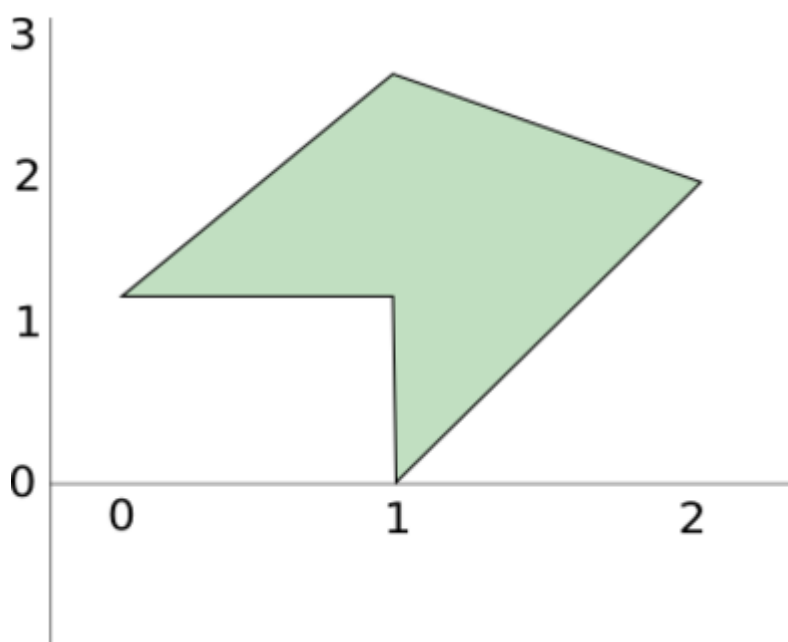
is expressed with: `#distance(field, 1, 2, 1)`

Within

Syntax	Use
<code>#within{options}(field, polygon)</code>	Search for positions stored in the <code>field</code> that are contained within the <code>polygon</code>
Simple polygon: (lat1, lng1; lat2, lng2; lat3, lng3, ...)	
Multiple polygons: [polygon1 polygon2 ...]	When specifying more than one polygon, the search zone is composed of subtracting zones described by polygons <code>polygon2</code> , <code>polygon3</code> , etc. to <code>polygon1</code> Important: When two polygons share a common segment, the behavior is undefined.

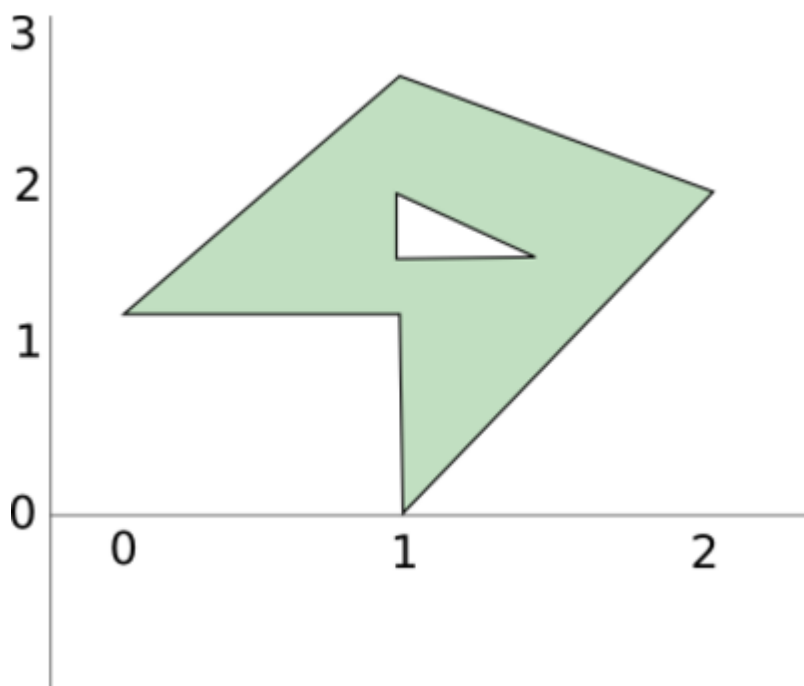
Examples:

- The following search area:



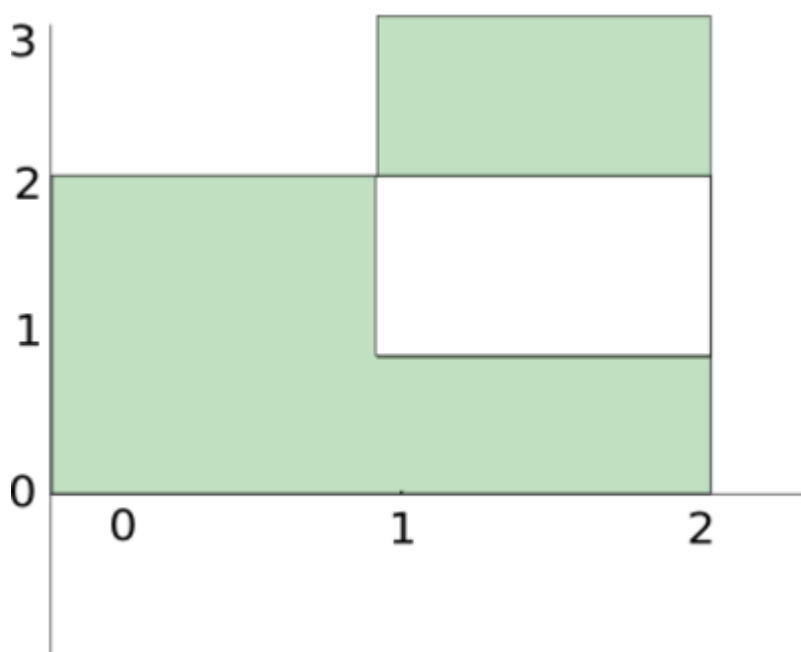
is expressed with: `#within(field, (0, 1; 1, 1; 1, 0; 2, 2; 1, 3; 0, 1))`

- The following area:



is expressed with: `#within(field, [(0, 1; 1, 1; 1, 0; 2, 2; 1, 3; 0, 1) (1, 2; 1, 1.5; 1.5, 1.5; 1, 2)])`

- Two overlapping polygons produce a symmetric difference as illustrated with:



expressed as: `#within(field, [(0, 0; 2, 0; 2, 1; 0, 2) (1, 1; 2, 1; 2, 3; 1, 3)])`

Important: When two polygons share a common segment, the behavior is undefined.

Specials

- `#true{options}()` - returns all documents (`#all` in UQL becomes `#true` in ELLQL)
- `#false{options}()` - returns nothing
- `#did{options}(5 10 45)` - returns the specified DIDs (here 5, 10 and 45)

Delimiters

Some operators allow you to detect delimiters within a document.

- `#page{options}(field)` - page delimiter
- `#paragraph{options}(field)` - paragraph delimiter
- `#sentence{options}(field)` - sentence delimiter

They are mostly used with the `#split` operator, that is itself the expansion of the `split` prefix handler.

For example, if you define a `split` prefix handler of type `insentence`, you can then enter a query such as `insentence:(texta AND textb)`. This query corresponds to the search of documents where `texta` and `textb` appear within the same sentence in the `text` field.

This UQL query is expanded as: `#split(#and(#alphanum(text, "a") #alphanum(text, "b")), #sentence(text))`

Compound Operators

All these operators hold other operators. The options of these operators, called below `internal_options`, can be used, for example, for scoring purposes:

- `positions.merge:`
 - `KEEP` - the positions of all children operators are never merged.
 - `MERGE` - the positions of all children operators are merged when they are the same.
- `<nodePropertyName>.policy={ADD,MAX,MIN}` - specifies how the node properties of children are merged.
- `*.policy=` - specifies the same policy for all node properties.

Unary Operators

Syntax	Use	Examples
<code>#not{internal_options} (query)</code>	Returns the documents that do not match the query	Retrieve all the documents that do not contain <code>toto</code> <code>#not(#alphanum(text, "toto"))</code>
<code>#opt{internal_options} (query)</code>	Makes optional query parts (only-useful for ranking)	
<code>#autocache{internal_opti (query)</code>	Caches the documents returned by query, and uses the cache for next queries. The scoring information associated to each document in the cache is discarded.	<ul style="list-style-type: none"> Load all the documents in the cache and return the documents: <code>#autocache{expectedSize=LAF (#category(categories, "Top/Source/ default"))</code> Use the cache: <code>#autocache(#category(catego "Top/Source/ default"))</code>
<code>#at(position, query)</code>	<p>Only works on alphanum fields. It applies query at an exact position in the field. The position is expressed in terms of indexed tokens; which means that usually this position does not take into account spaces, punctuation, etc.</p> <p><code>position</code> can be a positive value from 0, or a negative value (backward position, with -1 meaning the last position).</p> <p>When the position is negative, only <code>#alphanum</code> or <code>#anumpattern</code> can be used.</p>	<ul style="list-style-type: none"> Retrieve all the documents beginning with <code>toto</code>: <code>#at(0, #alphanum{k=2} (text, "toto"))</code> Retrieve all the documents ending with <code>toto</code>: <code>#at(-1, #alphanum{k=2}(text, "toto"))</code> Retrieve all the documents with <code>toto</code> as third word: <code>#at(2, #alphanum{k=2} (text, "toto"))</code>

Syntax	Use	Examples
	Important: Be careful when using <code>#at</code> with a <code>text</code> field since by default, several contexts are mapped into <code>text</code> (including <code>title</code> , <code>htmlcontext</code> , etc.)	<ul style="list-style-type: none"> Retrieve all the documents with exactly one word: <code>toto</code> (the same <code>toto</code> at the end and the beginning of the document):<code>#at(0, #at(-1, #alphanum{k=2}(text, "toto")))</code>
<code>#filter{internal_options("virtual_expr", query)}</code>	Returns the results of query only if "virtual_expr" is true.	-

Binary Operators

Syntax	Use	Examples
<code>#butnot{internal_options(search_query, avoid_query)}</code>	Returns all the documents matching at least one <code>search_query</code> with different positions that <code>avoid_query</code>	New BUTNOT "New York" is: <code>#butnot(#alphanum{k=2}(text, "New"), #seq(#alphanum{k=2}(text, "New") #alphanum{k=2}(text, "York")))</code>
<code>#split{internal_options(search_query, separator_query)}</code>	Applies <code>search_query</code> , taking care that all results are contained in a same <code>separator_query</code> (that may be a page, a sentence, or a paragraph)	Searching for A & B in the same page may be written in as: <code>#split(#and(#alphanum{k=2}(text, "A") #alphanum{k=2}(text, "B")), #page(text))</code>
<code>#innerjoin{internal_options(join_id, lead_query, filter_query)}</code>	Inner join has a few specific node options (<code>internal_options</code>) in addition to common options: <code>joinPolicy</code> <ul style="list-style-type: none"> <code>BITSET_JOIN</code> - Join with a compact bitset. Be careful, 	Return the emails with foo in the subject and a PDF in attachment: <code>#innerjoin(mail, #alphanum{k=2}(subject, "foo"),</code>

Syntax	Use	Examples
	<p>its size is limited to 1GB. If the limit is reached, users does not get any results. Yet, the error is reported in the Indexing Server log as: "Key for innerjoin is too large for a bitset policy, unable to execute the query.". To solve this issue, consider changing to SPARSESET_JOIN.</p> <ul style="list-style-type: none">SPARSESET_JOIN - Default value. Join with a sparse bitset.MERGE_JOIN - Compute all matching dids at initialization. Ranking keys are not propagated in this mode. <p><rankingKeyName>.join={JO</p> <p>- specifies how the ranking key value is merged.</p> <ul style="list-style-type: none">JOIN_LEFT - Default value. The ranking key is propagated from (and only from) left member of join node.JOIN_RIGHT - For a given join id, the ranking key is merged from all documents that match right member according to the innerjoin ranking key merge policy.JOIN_LEFT_RIGHT - For a given join id, the ranking key is merged from left member and all documents that match right member according to	<pre>#category(attachement_file_ty 3))</pre>

Syntax	Use	Examples
	the innerjoin ranking key merge policy.	

Nary Operators

Syntax	Use	Examples
<code>#seq{internal_options} (query1 query2 ...)</code>	Searches for a sequence. Each query must have its position following the previous query	"New York" is: <code>#seq(#alphanum{k=2} (text, "New") #alphanum{k=2} (text, "York"))</code>
<code>#and{internal_options} (query1 query2 ...)</code>	Searches for documents matching all the queries	New York is: <code>#and(#alphanum{k=2} (text, "New") #alphanum{k=2} (text, "York"))</code> New and York can be at any position in the document.
<code>#or{internal_options} (query1 query2 ...)</code>	Searches for documents matching at least one query	banana OR apple is: <code>#or(#alphanum{k=2} (text, "banana") #alphanum{k=2} (text, "apple"))</code>
<code>#bor{internal_options} (query1 query2 ...)</code>	Searches for documents matching at least one query. To be used only for a fast OR on many documents (no expansion, no ranking)	banana BOR apple is: <code>#bor(#alphanum{k=2} (text, "banana") #alphanum{k=2} (text, "apple"))</code>
<code>#fuzzyand{fuzzyand_option internal_options} (query1 query2 ...)</code>	Searches for documents matching at least X queries, where X is determined according to the <code>fuzzyand_option</code> . The score is adjusted according to the number of matching queries.	-

Syntax	Use	Examples
	<p><code>fuzzyand_option</code> can be:</p> <ul style="list-style-type: none"> • either <code>maxFailure=X</code> which means that up to X queries can fail, • or <code>minSuccess=X</code> means that at least X queries are expected to succeed. 	
<pre>#consecutive{internal_op (query1 query2 ...)</pre>	<p>Executes the queries in order (query1, then query2, etc.) and acts as an OR.</p> <p>However, when a timeout occurs, the first queries are more likely to have been fully completed than the last queries.</p>	-

Proximity Operators

Syntax	Use
<pre>#prox{internal_options} (query1 minDistance12 maxDistance12 query2 minDistance23 maxDistance23 ... maxDistance(N-1)N queryN)</pre>	<p>Each position of query <i>i</i> must be between <code>minDistance(i-1) i</code> and <code>maxDistance(i-1) i</code> positions from the query (i-1)</p> <p><code>minDistance</code> and <code>maxDistance</code> are signed, which enables several matching strategies for <code>#prox(A minDistance maxDistance B)</code>:</p> <ul style="list-style-type: none"> • <code>minDistance>0, maxDistance>0</code> - B is between <code>minDistance</code> and <code>maxDistance</code> positions after A. • <code>minDistance=maxDistance=distance>0</code> - B is exactly at distance positions after A. This can also be written in a more concise way as <code>#prox(A distance B)</code>. • <code>minDistance<0, maxDistance<0</code> - B is between <code>minDistance</code> and <code>maxDistance</code> positions before A. • <code>minDistance=maxDistance=distance<0</code> - B is exactly at distance positions before A. This can also be

Syntax	Use
	<p>written in a more concise way as <code>#prox(A distance B)</code>.</p> <ul style="list-style-type: none"> • <code>minDistance<0, maxDistance>0</code> - B is near A. <p>If all min (resp. max) distances are the same, the operator can be written in a more concise way, by specifying the distance before all query nodes:</p> <pre>#prox{internal_options}(minDistance maxDistance, query1 query2 ...)</pre> <p>These operators can use an optional <code>sameposok</code> option to indicate that a distance of 0 between children matches.</p>

Appendix - Search API Parameters

This page lists all parameters that can be used:

- in the commands of the Exalead CloudView Search API
- directly as parameters to the HTTP mount points, when using directly the HTTP REST API
- from the Search Clients (see SearchAPIClient)

Note: The .NET clients handle many of the Search API commands by the built-in properties. Others can be specified using custom parameters.

For reference documentation, see the related API javadoc and .NET reference documentation delivered in the SDK. For programmer documentation, see the Exalead CloudView Programmer's Guide.

Common notations:

- Many Search API parameters take a "KV map" as an argument. A KV map is a list of key values, written in the form: key:val,key2:val2.
- For each "Boolean" argument, the following values are accepted (case-insensitive): 0, 1, true, false, yes, no, disabled, enabled.

The search Command

This Search API command performs searches on the Exalead CloudView index. It is available at `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/search`. You can use it from the Java SearchClient and the .NET SearchClient classes.

Global Parameters

sl or logic

- String, monovalued.
- Must reference a valid logic, defined in the search logics list.

Example: print the query ast and not the ellq

```
q=job&sl=s10
q=job&logic=s10
```

st or target

- String, monovalued.
- Must reference a valid search target

Example: print the query `ast` and not the `ellql`

```
q=job&st=st0
q=job&target=st0
```

c or context

The context contains the encoded version of previous arguments to chain queries.

d or debug

Turns on various debugging flags for this query.

Value is a comma-separated list of the following parameters:

Parameter	Description
<code>query</code>	Logs query parsing and execution info
<code>synthesis</code>	Logs synthesis info
<code>ph</code>	Logs partial hit info (for example, scoring/collapsing keys)
<code>fh</code>	Logs full hit info (metas, hit categories)
<code>ast</code>	Puts the query <code>ast</code> in answer info
<code>ellql</code>	Puts the <code>ellql</code> in answer info (enabled by default)
<code>context</code>	Puts the context in answer info (enabled by default)
<code>executor</code>	Puts the executor in answer info. When a query execution requires several executions (disjunctive facets refinements, spell) they are all displayed.
<code>all</code>	All of the above
<code>qp:int</code>	queryprocessing debug level (0 to 3=all)

Note: You can disable flags using the `-` prefix.

Example: print the query `ast` and not the `ellql`

```
d=ast,-ellql
```

Example: print everything except the logs full hit info (`fh`)

```
d=all,-fh
```

Example: print everything except the context (`context`)

`d=all,-context`

[hit_infos; hit_info, hi](#)

Puts additional info in the hit to debug the ranking. The following info is supported:

Parameter	Description
<code>ast</code>	Computes the index AST for each hit.
<code>rankings</code>	Displays the ranking keys for each hit.
<code>all</code>	All of the above

Note: You can disable flags using "-" prefix.

Example: search for word "vacation" and enable ranking info for ast. You can cumulate the options.

`q=vacation&hi=ast`

`q=vacation&hi=ast,rankings`

[applicationId](#)

In V6.x, you can create multiple applications using the Mashup Builder. This parameter is the Mashup application ID passed by the API client.

Important: It is also required by the Business Console, since the Business Logic is directly embedded as a prelinguistic Business Processor into the Search Server.

To trigger the Business Processor at runtime, you have to pass the `applicationId` parameter and optionally, the `stagingMode` parameter:

Parameter	Description
<code>applicationId</code>	Represents your application identifier (for example, "default")
<code>stagingMode</code>	<p>Accepts two values:</p> <ul style="list-style-type: none"> <code>true</code> - to use the application configuration and compile the Business Console resources at query time (so you do not have to apply your configuration). <code>false</code> - to use the application configuration and precompiled Business Console resources.

Examples:

- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/search?q=test&applicationId=default&stagingMode=false` - The Business Processor uses the "default" application configuration and precompiled resources.
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/search?q=test&applicationId=default&stagingMode=true` - The Business Processor uses the "default" application configuration and compile resources at query-time.
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/search?q=test&applicationId=default` - The Business Processor will use the "default" application configuration and precompiled resources.
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/search?q=test` - The Business Processor is not triggered.

Checkpoints

The list of checkpoint values to retrieve on each slice, stored in the answer's slice info.

Value is a comma-separated list of checkpoint names.

Origin

This parameter has no impact on query treatment. However, it is logged in the `search.csv` file and can help you identify the origin of the query, when you analyze user queries using the **search-reporting** reporter with a CSV publisher. It is useful for better traceability and debugging.

Sorting and Grouping Parameters

s or sort

Sort is declared through several parameters:

Parameter	Description
<code>sort=STRING</code>	Comma-separated list of <code>asc([sort id])</code> or <code>desc([sort id])</code> , that defines the order of the hits.
<code>s=STRING</code>	Same as <code>sort</code> .
<code>sort.[sort id].expr=STRING</code>	An existing index field, an existing virtual field or a symbolic name, used to compare hits. Set to <code>[sort id]</code> by default.
<code>sort.[sort id].lsb=INTEGER</code>	Position of the least significant within <code>[0, 63]</code> to be considered. For integer field only. 0 by default.

Parameter	Description
<code>sort.[sort id].msb=INTEGER</code>	Position of the most significant within <code>[lsb, 63]</code> to be considered. For integer field only. 63 by default.
<code>sort.[sort id].limit=INTEGER</code>	Maximum number of byte used to compare character string. 0 means that the strings are compared till the end. 0 by default.
<code>sort.[sort id].min=INTEGER</code>	If the field is an integer, only keeps hits for which the <code>expr</code> is at least INT.

Example: sort alphabetically by region and then by price

```
s=asc(document_region),desc(document_price)
```

Example: sort alphabetically by increasing price, decreasing score and then by increasing file size. 'myprice' and 'myscore' are '[sort id]' [sort id] can not contain any space, or comma, or closing parenthesis, nor point. If such characters are required, use the expanded syntax.

```
s=asc(myprice),desc(myscore),asc(document_file_size)
s.myprice.expr=price-discount
s.myscore.expr=@term.score*@proximity*#round(document_double,4)
```

Example: sort by color index defined on 4 bits in the index field `document_bitfield`.

```
s=asc(color)
s.color.expr=document_bitfield
s.color.lsb=12
s.color.msb=15
```

is the same as: `s=asc(document_bitfield[12:15])`

The following syntax is also supported for backward compatibility. It allows only mono field sort.

- `s=field` - Sort descending on field.
- `s=-field` - Sort ascending on field.
- `s=field[lsb,msb]` - Sort descending on field, restricted to a given bitrange.
- `s=-field[lsb,msb]` - Sort ascending on field, restricted to a given bitrange.

Group

Theses parameters control the GroupBy feature.

Parameter	Description
<code>group=[group id], ...</code>	Comma-separated list of one or more <code>[group id]</code> . Each element stands for an independent group passed on the result set.

Parameter	Description
<code>group.[group id].by=[criterion id], ...</code>	Comma-separated list of one or more field names or expressions used to group hits. Set to <code>[group id]</code> by default.
<code>group.[group id].by.[criterion id].expr=STRING</code>	Field name or expression used to group hits. Set to <code>[criterion id]</code> by default.
<code>group.[group id].by.[criterion id].lsb=INTEGER</code>	Position of the least significant within <code>[0, 63]</code> to be considered. For integer field only. 0 by default.
<code>group.[group id].by.[criterion id].msb=INTEGER</code>	Position of the most significant within <code>[lsb, 63]</code> to be considered. For integer field only. 63 by default.
<code>group.[group id].by.[criterion id].limit=INTEGER</code>	Maximum number of byte used to compare character string. 0 means that the strings are compared till the end. 0 by default.
<code>group.[group id].aggregation=[agg id], ...</code>	Comma-separated list of one or more aggregation names.
<code>group.[group id].aggregation.[aggr id].function=POLICY</code>	Score actions that combine the ranking value for a group. POLICY is one of SUM, MAX, MIN, AVG, STDDEV, CONCAT. MAX by default.
<code>group.[group id].aggregation.[aggr id].expr=STRING</code>	Field name or expression that contain the data to be aggregated. Set to <code>[aggr id]</code> by default, you can omit it when <code>expr</code> is equal to the <code>[aggr id]</code>
<code>group.[group id].aggregation.[aggr id].separator=STRING</code>	Sequence on characters placed in-between the concatenated values.
<code>group.[group id].aggregation.[aggr id].aggregationField</code>	Name of the field or expression where the aggregation result is written. It can then be used to be displayed (combined with <code>add_hit_meta</code>) or to perform sort (combined with <code>sort</code>) for example.

Parameter	Description
<code>group.[group id].topn=INTEGER</code>	Number of hits to keep for each group.
<code>group.[group id].sort=STRING</code>	Defines the order of hits within each group to keep the top n best. See <code>sort=</code> .
<code>group.[group id].s=STRING</code>	Same as <code>group.[group id].sort</code> . See <code>sort=</code> .
<code>group.[group id].sort.[sort criterion id].*</code>	Sort criterion parameters. See <code>sort=</code> .

By default, a group pass has a single group dimension defined with `expr` set to `[group identifier].[group id]` and `[criterion id]` can neither contain comma, nor point. If such characters are required in the expression, use the expanded syntax.

Example: Keep one document for each state. The best one is selected according to the ranking sort, the others are discarded.

```
group=document_region
```

Example: Keep the five best documents for each state.

```
group=document_region
group.document_region.topn=5
```

Example: Same as previous, but uses a label to identify the group rather than the field name.

```
group=byRegion
group.byRegion.by=document_state
group.byRegion.topn=5
```

Example: Keep the best noted hit per region, sort regions per hit count, and display the count. This example relies on the constant virtual expression 1.

```
group=document_region
group.document_region.aggregation=1
group.document_region.aggregation.1.function=SUM
group.document_region.sort=desc(document_note)
sort=desc(1)
add_hit_meta=1
```

is the same as:

```
add_virtual_field=count:1
group=document_region
group.document_region.aggregation=count
group.document_region.aggregation.count.function=SUM
group.document_region.sort=desc(document_note)
```

```
sort=desc(count)
add_hit_meta=count
```

Example: Keep no more than five hit per region, and no more that eight times the same product. This configuration creates two independent collapsing passes. The result set may contain fewer than 5 hits per region because `document_product_name` removes hits of the groups made by `document_region` pass.

```
group=document_region,document_product_name
group.document_region.topn=5
group.document_product_name.topn=8
```

Example: Keep one record for each firstname, lastname pair, select most recently updated hit, and get the average note in `document_note` and the total expense in `total_expense`. Finally results are sorted alphabetically.

```
group=byPerson
group.byPerson.by=document_firstname,document_lastname
group.byPerson.s=desc(document_lastupdatedate)
group.byPerson.aggregation=document_note,document_price
group.byPerson.aggregation.document_note.function=AVG
group.byPerson.aggregation.document_price.function=SUM
group.byPerson.aggregation.document_price.aggregationField=total_expense
sort=asc(document_lastname),asc(document_firstname)
add_hit_meta=document_note
add_hit_meta=total_expense
```

Collapsing (DEPRECATED)

collapsing parameters are supported for backward compatibility. Prefer `group` parameters.

Parameter	Description
<code>collapsing</code>	Comma-separated list of group identifiers. Each element stands for an independent collapsing passed on the result set. For each group, the following <code>collapsing.[group identifier].*</code> parameters are available
<code>collapsing.[group identifier].expr=STR</code> <code>identifier]</code> by default.	Field name or expression used to group hits. Set to <code>[group identifier]</code> by default.
<code>collapsing.[group identifier].lsb=INT</code>	Position of the least significant within <code>[0, 63]</code> to be considered. For integer field only. 0 by default.
<code>collapsing.[group identifier].msb=INT</code>	Position of the most significant within <code>[lsb, 63]</code> to be considered. For integer field only. 63 by default.

Parameter	Description
<code>collapsing.[group identifier].limit=INT</code>	Maximum number of byte used to compare character strings. 0 means that the strings are compared till the end. 0 by default.
<code>collapsing.[group identifier].nbhits=INT</code>	Number of hits to keep for each group.
<code>collapsing.[group identifier].action.[virtual expression]=POLICY</code>	Score actions that combine the ranking values for a group. POLICY can be add, max, min or best. For each grouping key, there can be one action for each ranking element. best by default.

Example: Keep one document for each state. The best one is selected according to the ranking sort, the others are discarded.

```
collapsing=document_region
```

Example: Keep the five best documents for each state.

```
collapsing=document_region
collapsing.document_region.nbhits=5
```

Example: Same as previous but uses a label to identify the group rather than the field name.

```
collapsing=byRegion
collapsing.byRegion.expr=document_state
collapsing.byRegion.nbhits=5
```

Example: Keep one hit per region, sort regions per hit count, and display the count. This example relies on the constant virtual expression 1.

```
collapsing=document_region
collapsing.document_region.action.1=add
sort=desc(1)
add_hit_meta=1
```

Example: Keep no more than five hits per region, and no more that eight times the same product. This configuration creates two independent collapsing passes. The result set may contain fewer than 5 hits per region because `document_product_name` remove hits of the groups made by `document_region` pass.

```
collapsing=document_region,document_product_name
collapsing.document_region.nbhits=5
collapsing.document_product_name.nbhits=8
```

User Query

q or Query

- string in UQL format
- Parameter name can be in the form q.X to define a named query.

Example:

```
q=olympic games
query=olympic games
```

eq or ellql_query

- string in ELLQL format
- Parameter name can be in the form eq.X to define a named query.

Example: search for olympic games in text field

```
eq=#alphanum(text, "olympic games")
```

qt or query_template

- String, in the form of an ELLQL tree, defining how to combine the named queries
- **Default value:** `#and(#query(_default_) #query(restriction) #query(security) #query(refine))`
- If a named query does not exist, it is replaced by a `!NoOp()` node and therefore ignored.

Security, enforce_security

- Adds a security token, multivalued.
- Used to increase the auto-built disjunction in the "security" named query.
- By default, security tokens generate category lookups on the "security" field.
- `enforce_security=true|false` globally enables or disables security.

Example: search for word "secured" in text field with security enabled and security token provided

```
q=secured&enforce_security=true&security=unix:user:10125
```

l or lang

- string, monovalued
- The global query ISO language.
- This parameter does not restrict searches to documents in this language. It only controls the language used for computing linguistic expansions, spell checking, and summarization.

Example: search for word "vacation" in text field in English language

```
q=vacation&l=en
```

r or refine

- Adds a refinement.
- The combination of refines which creates a named query called "refines".

Example: search for word "vacation" in text field and refine on source category "corpus"

```
q=vacation&r=f/source/corpus
```

cr or cancel_refine

Cancels a refinement completely.

Example: search for word "vacation" in text field, refine on source category "corpus" and cancel this refine

```
q=vacation&r=f/source/corpus&cr=f/source/corpus
```

zr or zap_refine

Removes a refinement and refines on the father, if applicable. For example, if we have refinement on "Top/MyPath/A/B/C", zapping this refinement replaces it with a refinement on "Top/MyPath/A/B"

Example: search for word "vacation" in text field, refine on source category "corpus" and zap this refine.

```
q=vacation&r=f/source/corpus&zr=f/source/corpus/english
```

handle_unknown_refine_as_false

If `true`, when a `r=f/unknownfacet/refine` is specified, returns 0 results rather than an error.

Default: `false`

UQL Interpretation

dp or default_prefix

Sets the default prefix handler. The default value is the one specified in the search logic. Can be restricted to a particular named query using `dp.NAMED_QUERY_NAME` or `default_prefix.NAMED_QUERY_NAME`.

Example: search for word "vacation" and set default prefix handler to "title", so "vacation" is searched in the title field.

```
q=vacation&dp=title
```

qec,query_expansion_config

Sets the QueryExpansionConfig for a given prefix.

The argument name can be specified as:

Parameter	Description
<code>qec=SPEC</code>	Sets the QEC of the default prefix
<code>qec.PREFIX=SPEC</code>	Sets the QEC on a given prefix
<code>qec.ALL=SPEC</code>	Sets the QEC on all semantic prefixes
<code>q.QUERY.qec=SPEC</code>	Sets the QEC on a named query
<code>q.QUERY.qec.PREFIX=SPEC</code>	Sets the QEC on a given prefix only on a named query

The argument value (`SPEC`) is specified as a QEC specification.

Note: Specifications are cumulative.

Examples:

- `name=spellslike spec=phonetic`
- `name=text spec=synonyms1`
- `name=spells2 spec=phonetic{m=3}`
- `text:spellslike:foo will do phonetic + synonyms1`
- `text:spellslike:spell2:foo will do phonetic{m=3} + synonyms1`

pal or patterns_all_languages

Sets whether pattern expansions are performed in all languages (Boolean parameter).

Limits Parameters

Limits

Defines search limits. This parameter is specified as a KV Map.

The basic keys are:

- `max_fetched_hits`
- `max_fetched_hits_per_slice`
- `max_query_time`
- `max_total_time`
- `hits_sampling`

Advanced keys are:

- `max_kept_hits`
- `main_heap_flush_interval`
- `slice_heap_flush_interval`
- `interrupt_grace_delay`
- `full_grace_delay`

Detailed documentation of these parameters can be found in SearchLogic / LimitsConfig in XML Configuration Reference

Example: Override timeout and limits

```
limits=max_total_time:0&limits=max_query_time:n&limits=max_fetched_hits_per_slice:0
```

Timeout

Shortcut for the limits parameter. When you want to set a timeout, unlike the `max_total_time` parameter, the `timeout=X` parameter also sets the value of `max_query_time` parameter.

- `timeout=INT` - Sets the global timeout. Query timeout is 75% of the global value.
- `timeout=INT, INT` - Sets the query and global timeouts.

nhits

Shortcut for `limits=max_kept_hits:INT`

hf or full_hits, nresults

Number of full hits. `hf=0` disables full hits fetch.

b or start

First full hit index. For example, to search on the:

- first page: `b=0&hf=10`
- second page: `b=10&hf=10`

Hit Meta Parameters

use_logic_hit metas

- `true/false` - (Optional) Removes all metas defined in the search logic. Default is `true`.

- `metaname1,metaname2,metaname3` - Keeps only this list of metas. Be careful they must exist in search logic unless `allow_unknown_hit_meta` is `true`.

`allow_unknown_hit_meta`

Allows you to set an unknown meta in the list provided in `use_logic_hit metas`, which is silently discarded. Default is `false`.

`add_hit_meta`

Add a new hit meta to the query result. Use multiple `add_hit_meta` clauses to add multiple metas. The value is a list of key:value separated by commas:

key:value	Description
<code>name:string</code>	The name of the meta
<code>index_field:string</code>	Name of the index field used to retrieve the meta value from, when different from the meta name. Default is <code>name</code> .
<code>multi_field:string</code>	Name of the index multiple fields (a.k.a. csv field) used to retrieve the values.
<code>dynamic_field:string</code>	Name of the index dynamic field used to retrieve the values.
<code>context_name:string</code>	For <code>dynamic_field</code> use only, specify the context to extract from the dynamic field. If empty, it is assumed that <code>context_name</code> has <code>name</code> value
<code>highlight:boolean</code>	When true, the meta value is highlighted in the summary. <code>false</code> by default.
<code>summarize:boolean</code>	<code>false</code> by default.
<code>select_values:boolean</code>	<p>This parameter only works for value and alphanumeric fields.</p> <p>It allows you to configure the number of values displayed for metas in hit content. This is typically useful to restrict the number of values retrieved from multivalued fields when you do not want to clutter hit content with too many values for a given meta.</p> <p>You can specify a min and a max values to this operation with the <code>min_values</code> and <code>max_values</code> attributes. For example, if a multivalued <code>foo</code> field has 10 values, 5 of which matching the query, you can display N of these values in a new bar meta by adding the following Search API HTTP parameters:</p> <pre>add_hit_meta=name:bar,index_field:foo,select_values:true,max_values:N</pre> <p>This operation uses the query to determine whether a given value is a match or not, you therefore have to add another parameter like</p>

key:value	Description
	<p><code><property_name>:<value_to_match></code> to your query to select only the values matching <code>value_to_match</code> for the <code>property_name</code> meta.</p> <p>Note: If the property has no value matching <code>value_to_match</code>, you lose a search result.</p>

Note: `index_field`, `multi_field`, `dynamic_field` are exclusive. By default the meta value is extracted from an `index_field` with the same name as the meta. Dynamic fields, and multi fields contain multiple keys and values, the values matching the meta name are returned.

There is a compact syntax for `add_hit_meta` that supports only `index_fields` and highlighting:

- `name` - Name of the index field
- `name,hl` - Name of the index field, highlight:Boolean

Example: Simplified syntax

```
add_hit_meta=document_field_name,hl
```

is the same as: `add_hit_meta=index_field:document_field_name,highlight:true`

Example: Extracts color and value from the dynamic property named `document_dynamic_field`.

```
add_hit_meta=dynamic_field:document_dynamic_field,name:color&
add_hit_meta=dynamic_field:document_dynamic_field,name:value
```

Example: You could also extract color from the dynamic property named `document_dynamic_field` and output its value in meta named property using a virtual expression:

```
add_hit_meta=dynamic_field:document_dynamic_field,name:property,context_name:color
```

To use a virtual expression in a hit meta:

- `hit_meta.name.expr=expr` - where `name` can contain any character, and `expr` is a virtual expression.

Add meta specific operation:

- First, you have to define the type of your operation, using the following syntax:

```
(add_)hit_meta.META_NAME.operation.OPERATION_ID.type
```

- Then, you can associate properties to a given operation:

```
(add_)hit_meta.META_NAME.operation.OPERATION_ID.property.OPERATION_PROPERTY
```

The following tables list available properties associated by type:

Type	Properties
custom	<ul style="list-style-type: none"> <code>class_id:string</code> - the fully-qualified name of the class performing the operation <code>YOUR_KEY:string</code> - a KV passed to your operation
date_format	<code>output_format:string</code> - default is <code>%m/%d/%Y</code>
highlight	<ul style="list-style-type: none"> <code>facet_ids:string</code> <code>extra_prefix_handlers:string</code>
printf	<code>output_format:string</code>
snippet	<p>Makes a single value of a meta shorter when at least one of the meta's values matches. The typical use case for <code>snippet</code> is long, mono-valued, text metas. For short, multi-valued string metas, use <code>select_values</code>.</p> <p>This operation uses the query to determine whether a given value is a match or not, you therefore have to add another parameter like <code><property_name>:<value_to_match></code> to your query to select only the values matching <code>value_to_match</code> for the <code>property_name</code> meta.</p> <p>Note: If the property has no value matching <code>value_to_match</code>, you lose a search result.</p> <ul style="list-style-type: none"> <code>min_length:int</code> - default is 1 <code>max_length:int</code> - default is 150 <code>max_sentence_segment_length:int</code> - default is 150 <code>max_sentence_segments:int</code> - default is 3 <code>split_on_sentence:boolean</code> - default is true <code>remove_duplicate_segments:boolean</code> - default is false <code>if_meta:string</code> - is a fallback meta if one of the <code>if_metas_match:string</code> meta matches the query. When both parameters are used, if a match is found, the summary is filled with the <code>if_Meta</code> value. <code>if_metas_match:string</code> - allows you to specify a comma-separated list of metas to test against the user query <code>highlight_facet_ids:string</code> <code>highlight_extra_prefix_handlers:string</code>

Type	Properties
standard_decoding	<ul style="list-style-type: none"> encoding:string - accepted values are url or idna
time_format	<ul style="list-style-type: none"> output_format:string - default is %m/%d/%Y %H:%M:%S
truncate	<ul style="list-style-type: none"> max_length:int strict:boolean - default is false

Example: Adds a meta hello from the field document_polite with a truncate processor named truncate_hello with max_length=10, strict=true

```
add_hit_meta=name:hello,index_field:document_polite&
hit_meta.hello.operation.truncate_hello.type=truncate&
hit_meta.hello.operation.truncate_hello.property.max_length=10&
hit_meta.hello.operation.truncate_hello.property.strict=true
```

add_hit metas

Extract all values from a multivalued field. It applies to multi fields (a.k.a. csv fields) and dynamic fields. The value is a key:value list separated by comma.

For dynamic fields only, you can also:

- Select a subset of contexts if you specify match rules (available rules: "exact", "prefix", "suffix", "substring"). You have to use them as key, the associated value being the pattern you are looking for.

Note: You can associate several rules. In this case, the hit meta is displayed if its name matches at least one rule.

- Specify the name of the meta that receives all the content of the dynamic field. If not set, for each pair key:value stored in the field, a meta named as the key is created.

Parameter	Description
multi_field:string	Name of the index field used to retrieve the meta names and values. Used by default.
dynamic_field:string[,ruleA:pattern,[,name:string]	Name of the index field used to retrieve the meta names and values, optional match rules, optional output meta name.

Simplified syntax when the type of field is not specified multi_field is assumed.

Example: simple syntax

```
add_hit_metas=document_csv_field
```

is the same as: `add_hit metas=multi_field:document_csv_field`

Example: dynamic field (displays all contexts)

```
add_hit metas=dynamic_field:document_dynamic_field
```

Example: dynamic field with exact match rules (displays `toto_first` and `toto_second` but not `toto_third`)

```
add_hit metas=dynamic_field:document_dynamic_field,exact:toto_first,exact:toto_second
```

Example: dynamic field with prefix rules (displays `toto_first` and `toto_second` but not `titi_first`)

```
add_hit metas=dynamic_field:document_dynamic_field,prefix:toto_
```

Example: dynamic field with name (if `dyn_[first:value1 first:value1]` and `dyn_[second:value2 second:value2]` are stored, displays `metagroup:value1,value2`)

```
add_hit metas=dynamic_field:document_dynamic_field,prefix:dyn_,name:metagroup
```

remove_hit_meta

In the search logic, metas are identified by their names in the `HitConfig` object. When you remove a meta, this removes all sources and all processors attached to this meta. Value = meta name

add_highlight

Highlights an existing meta. Value = meta name

add_summary

Summarizes an existing meta. Value = meta name

hit_meta_order

Comma-separated list specifying in which order to display metas. Applicable only to CSV output.

All metas not present in the list are displayed after the metas present in the list.

Note: The following metas are not affected and are always displayed first, in the following order:

`did, url, buildGroup, source, slice, score, mask, sort`

Faceting Parameters

Synthesis

- enabled/disabled

- Defaults to search logic config value.

synthesis_hits

- Number of hits used for category synthesis.
- Defaults to the value in the search logic.

use_logic_facets

Specifies the search logic facets to use and disables all other facets. For the specified facets, the configuration is read in the search logic.

- `true` - keeps all (default value)
- `false` - removes all
- `facet_id1, facet_id2, ..., facet_idN` - keeps only this list of facets

remove_facet

One argument: `facet_id`.

f.* or facet.*

Creates a facet. Facets are specified as multiple parameters, like `facet.FACET_ID.key=value` where `FACET_ID` is the facet id (`_a-zA-Z0-9`).

Keys for ALL facets

Key	Description
<code>type</code>	Specifies the facet type (default is <code>category</code>), which can be: <ul style="list-style-type: none">• <code>category</code> (aka "cat")• <code>value</code>• <code>date</code>• <code>dyndate</code>• <code>num_explicit</code> (Numerical explicit ranges)• <code>num_fixed</code> (Numerical fixed ranges)• <code>num_dynamic</code> (Numerical dynamic ranges)• <code>geo</code>• <code>autotile</code>• <code>multi</code>

Key	Description
	<ul style="list-style-type: none"> • 2D or matrix • enum
<code>min_docs:long</code>	Min number of docs that have the facet value (default is 1)
<code>max_elements:uint</code>	Max number of categories (default is 0=unlimited)
<code>max_per_slice</code>	Specifies the max number of different facets returned by a single slice.
<code>in_hits:boolean</code>	Default is <code>true</code>
<code>in_synthesis:boolean</code>	Default is <code>true</code>
<code>sort</code>	<p>Specifies the sorting function (default is <code>count</code>), which can be:</p> <ul style="list-style-type: none"> • <code>aggregation</code> - Sorts the categories using the aggregation function specified with the <code>sort_agg_fun: string</code> key (see below). Default sorting direction is descending. • <code>alphanum</code> - Sorts the categories lexicographically. The category path is used here, not the title. Default sorting direction is ascending. • <code>count</code> - Sorts the categories by decreasing order, with the number of documents matching the query and having this facet. • <code>date</code> - Sorts the categories by: <ul style="list-style-type: none"> ◦ Default with decreasing year, increasing month, increasing day ◦ Reverse (with <code>-date</code>) with increasing year, decreasing month, decreasing day • <code>explicit</code> - Sorts the categories using an explicit order. Use a comma-separated list of values. • <code>Lat</code> - Sorts the categories by latitude, using the average of points. • <code>Lng</code> - Sorts the categories by longitude, using the average of points. • <code>Num</code> - Tries to parse the category path as an integer, and sorts decreasingly. If the category is prefixed by a number it parses the prefix. In case of failure, it fallbacks to lexicographical sorting. Default sorting direction is ascending. • <code>range</code> - If the categories are ranges in the form <code>[a;b]</code>, it sorts the categories per increasing midrange value. Default sorting direction is ascending.

Key	Description
	<ul style="list-style-type: none"> <code>relevancy</code> - Sorts the categories by decreasing relevance. Relevance is defined by taking into account both the number of documents matching the query and having this facet, and the total number of documents having the facet. The idea is to use a method of ponderation similar to the classical TF-IDF. <p>To reverse the sorting order, prefix the sorting function with <code>-</code>, for example, <code>-aggregation</code>.</p>
<code>sort_agg_fun:</code> string	Sort aggregation function id. (requires <code>sort=aggregation</code>)
<code>explicit_sort_order:</code> string	Comma-separated list of values defining the explicit sort order. Note: You also need to configure properly <code>sort</code> parameter to use this feature.
<code>implem</code>	Implementation must be <code>cpu</code> , <code>mem</code> or <code>auto</code> . (default is <code>auto</code>)
<code>refinement_policy:</code> string	Refinement policy <code>exclusive</code> , <code>disjunctive</code> or <code>norefine</code> . (default is <code>exclusive</code>)

Keys for category facets

Key	Description
<code>form:</code> string	<code>exact</code> / <code>lowercase</code> / <code>normalized</code> (default is <code>normalized</code>)
<code>root:</code> string	Required
<code>field:</code> string	Category field (Required)
<code>max_depth:</code> uint	Max depth from refine (default is <code>0=unlimited</code>)
<code>max_depth_from_root:</code>	Default <code>0=unlimited</code>
<code>max_per_level:</code> uint	Default <code>100</code>

Example: Standard facet with aggregation - Use this for the default Exalead CloudView index schema, which includes a categories field.

```
f.common_facet.type=category
f.common_facet.root=Top/ClassProperties/facetmut3
f.common_facet.field=categories
f.common_facet.aggr.total=SUM(doc_int1)
```

Example: Dedicated facet with aggregation - Use this if you have added another category type field (in this case, `stdf3`) to your index schema.

```
f.std3.type=category
f.std3.root=Top
f.std3.field=std3
f.std3.aggr.total=SUM(doc_int1)
```

Keys for value facets

Key	Description
<code>vroot: string</code>	(Required)
<code>field</code>	(Required) the value field to use.

Example:

```
f.valf3.type=value
f.valf3.vroot=Top/
f.valf3.field=valf3
f.valf3.aggr.total=SUM(doc_int1)
f.cod_ente.type=value
f.cod_ente.vroot=Top/
f.cod_ente.field=cod_ente
f.cod_ente.aggr.total=SUM(anno_ruolo)
```

Keys for optimized enum facets

Key	Description
<code>vroot:string</code>	(Required)
<code>enum_facet_[if:string]</code>	(Required) enum facet identifier

Keys for date facets

Key	Description
<code>vroot:string</code>	(Required)
<code>expr:string</code>	(Required)
<code>start:string</code>	CONSTANT virtual expression that evaluates to the time when the synthesis starts
<code>end:string</code>	CONSTANT virtual expression that evaluates to the time when the synthesis ends
<code>before_start:bool</code>	Creates a virtual category for all dates before start (default is <code>false</code>)

Key	Description
after_end:bool	Creates a virtual category for all dates after end (default is <code>false</code>)
year:bool	Creates a virtual category for each (default is <code>true</code>)
year_desc:bool	Defines sort order for year (default is <code>true</code>)
month:bool	Creates a virtual category for each month (default is <code>true</code>)
month_desc:bool	Defines sort order for month (default is <code>false</code>)
week:bool	Creates a virtual category for each week (default is <code>false</code>)
week_desc:bool	Defines sort order for week (default is <code>false</code>)
day:bool	Creates a virtual category for each day (default is <code>true</code>)
day_desc:bool	Defines sort order for day (default is <code>false</code>)
hh:bool	Creates a virtual category for each hour (default is <code>false</code>)
hh_desc:bool	Defines sort order for hour (default is <code>false</code>)
mm:bool	Creates a virtual category for each minute (default is <code>false</code>)
mm_desc:bool	Defines sort order for minute (default is <code>false</code>)
ss:bool	Creates a virtual category for each second (default is <code>false</code>)
ss_desc:bool	Defines sort order for second (default is <code>false</code>)
max_depth:int	Default is 0
max_depth_from_root:	Default is 0

Keys for dynamic date facets

Key	Description
vroot:string	(Required)
expr:string	(Required)
missing_intervals:bool	Generates the missing intervals ensuring the dates are contiguous
year_fmt:string	Defines year-based generated category format
quarter_fmt:string	Defines quarter-based generated category format
month_fmt:string	Defines month-based generated category format
week_fmt:string	Defines week-based generated category format

Key	Description
<code>day_fmt:string</code>	Defines day-based generated category format
<code>hour_fmt:string</code>	Defines hour-based generated category format
<code>min_fmt:string</code>	Defines minute-based generated category format
<code>sec_fmt:string</code>	Defines second-based generated category format
<code>enable_year:boolean</code>	Enables the year level
<code>enable_quarter:boolean</code>	Enables the quarter level
<code>enable_month:boolean</code>	Enables the month level
<code>enable_week:boolean</code>	Enables the week level
<code>enable_day:boolean</code>	Enables the day level
<code>enable_hour:boolean</code>	Enables the hour level
<code>enable_min:boolean</code>	Enables the minute level
<code>enable_sec:boolean</code>	Enables the second level
<code>enable_iso8601:boolean</code>	Enables the iso8601 norm for date representation

Virtual numerical facets are used to organize search results in ranges.

Numerical range facets

Key	Description
<code>num_explicit</code>	Create ranges manually
<code>num_fixed</code>	Create ranges that are the same size
<code>num_dynamic</code>	Create ranges automatically

Generic keys for numerical range facets

Key	Description
<code>vroot:string</code>	(Required)
<code>expr:string</code>	(Required) see Virtual Expression Syntax reference page.
<code>range</code>	(0 to N times) <code>min,max,title</code> , for example <code>"-1,1,my title"</code>
<code>min:double</code>	Default is 0
<code>max:double</code>	Default is 2^{63}

Key	Description
lsb:int	Default is 0
msb:int	Default is 63
default_precision:int	Default is 2

num_fixed specific keys

Key	Description
rsizе:double	Range size (def 0=one range per distinct value).
below_min:bool	Creates a range for values below min.
above_max:bool	Creates a range for values above max.
fmt_range	Sets the range title format.
fmt_above	Sets the above title format.
fmt_below	Sets the below title format.
fmt_single	Sets the singleton title format.

num_dynamic specific keys

Key	Description
fmt_range	Sets the range title format.
nb_ranges:int	The maximum number of ranges to output (dynamic).
policy:enum	The policy to generate the ranges.
adjust_ranges:bool	Tries to adjust the ranges on multiples of 10 (deprecated).
more_accurate:bool	Better linear/geometrical ranges, but slower (deprecated).
mrsizе:bool	Size of each bin used to compute cardinality.
exclusiveRightBracket	In the case of dynamic ranges, this parameter determines if the facet is exclusive or inclusive from the right side (<code>true</code> means <code>[a,b[</code> , and <code>false</code> means <code>[a,b]</code>)

Example: Define custom price ranges and titles for each range

```
f.price.type=num_explicit
f.price.vroot=Top/Price
f.price.expr=document_dbl1
f.price.range=0,1000,0to1000euros
f.price.range=1000,2000,1000to2000euros
```

```
f.price.range=2000,4000,Tooexpensiveforyou
```

Keys for geographic facets

Key	Description
vroot:string	(Required)
field:string	(Required) the point field to use for synthesis
domain	(0 to N times) a domain specification: <ul style="list-style-type: none"> DiskDomain: <code>disk('x','y','radius'),'title'</code> - matches all points within the specified radius of point (x,y). PolygonDomain: <code>poly('x1','y1';x2,y2;...; 'xn','yn'),'title'</code> - matches all points in the polygon defined by the specified points.

Keys for Auto-Tile geographic facets

Key	Description
vroot: string	(Required)
field	(Required) the PointField to use
xmin, xmax, ymin, ymax	Parameters defining the facet with X and Y coordinates for the global geographic query. <code>xmax</code> and <code>ymax</code> define the Max envelope of the geographic facet.
xbin, ybin	Parameters defining the size of the tile (with X and Y absolute coordinates) within the global geographic query scope.

Keys for 2D or matrix facets

Key	Description
id1:string	(Required)
id2:string	(Required)
vroot: string	(Required)
ds: boolean	With dimension switch (default is <code>false</code>)
sec_sort: string	Secondary sort function. Same syntax as <code>sort</code> . (default is <code>count</code>)
sec_sort_agg_fun: string	Secondary sort aggregation function id

Key	Description
<code>sec_explicit_sort_order_string</code>	Comma-separated list of values defining the secondary explicit sort order
<code>f.mycompany_facet1.type=value</code> <code>f.mycompany_facet1.vroot=Top/</code> <code>f.mycompany_facet1.field=mycompany_facet1</code>	
<code>f.mycompany_facet2.type=value</code> <code>f.mycompany_facet2.vroot=Top/</code> <code>f.mycompany_facet2.field=mycompany_facet2</code>	
<code>f.mat.type=matrix</code> <code>f.mat.vroot=Top/</code> <code>f.mat.id2=mycompany_facet1</code> <code>f.mat.id1=mycompany_facet2</code> <code>f.mat.aggr.total=SUM(mycompany_int1)</code> <code>f.mycompany_facet1.max_elements=10</code> <code>f.mycompany_facet2.max_elements=10</code>	

Keys for multidimension facets

Key	Description
<code>vroot: string</code>	(Required)
<code>additional_tree_representation bool</code>	Generate the tree representation
<code>additional_tree_representation bool</code>	Generate the tree representation with the extra dimension separator
<code>f.id.dim.prop</code>	<p>Specifying MultiFacetDimensions for MultiDimensionFacets can be done with the <code>f.id.dim.prop</code> syntax where:</p> <ul style="list-style-type: none"> <code>id</code> - is the id of the MultiDimensionFacet <code>dim</code> - is the (0-based) dimension for which to add a MultiFacetDimension <code>prop</code> - is the property name to set, and can be: <code>id</code>, <code>sort</code>, <code>max_elements</code>, <code>sort_agg_fun</code>
<code>f.id.aggr.aggId=FCT(xpr)</code>	<p>Adds an aggregation function with id <code>aggId</code> and expression <code>xpr</code></p> <p><code>FCT = SUM, AVG, MIN, MAX, ...</code></p>
<code>f.heat_consump_apt.type=multi</code>	

```
f.heat_consump_apt.vroot=Top/heat_consump_apt
f.heat_consump_apt1.id=location_code
f.heat_consump_apt2.id=heating_type
f.heat_consump_apt.additional_tree_representation=true
f.heat_consump_apt.aggr.conso=SUM(consumption_statement)
f.heat_consump_apt.in_hits=false
```

Dynamic Search Target

st.* or target.*

Defines a search target. The search target is specified with multiple parameters, like `st.TARGET_NAME.key=value` or `target.TARGET_NAME.key=value` where `TARGET_NAME` is the search target name (`_a-zA-Z0-9`).

The following keys are supported by all search targets:

- `type:string` - "local", "simple"
- `watch_dog_connect_timeout_ms:int`
- `watch_dog_read_timeout_ms:int`

The local search target supports the following key: `build_groups:string`

The simple search target requires the definition of target slices. Target slices are specified with multiple parameters, like `st.TARGET_NAME.target_slices.TARGET_ID.key=value` or `target.TARGET_NAME.target_slices.TARGET_ID.key=value` where `TARGET_NAME` is the search target name (`_a-zA-Z0-9`) and `TARGET_ID` is a number (0-9). The following keys are supported:

- `build_groups:string`
- `slices:string`
- `instances:string`
- `power:int`

Examples:

```
st=my_dynamic_search_target
st.my_dynamic_search_target.type=local
st=my_dynamic_search_target
st.my_dynamic_search_target.type=local
st.my_dynamic_search_target.watch_dog_connect_timeout_ms=1000
st.my_dynamic_search_target.watch_dog_read_timeout_ms=5000
st.my_dynamic_search_target.build_groups=bg0
st=my_dynamic_search_target
st.my_dynamic_search_target.type=simple
st.my_dynamic_search_target.target_slices.0.build_groups=bg0
```

```

st=my_dynamic_search_target
st.my_dynamic_search_target.type=simple
st.my_dynamic_search_target.watch_dog_connect_timeout_ms=1000
st.my_dynamic_search_target.watch_dog_read_timeout_ms=5000
st.my_dynamic_search_target.target_slices.0.build_groups=bg0
st.my_dynamic_search_target.target_slices.0.slices=0,1
st.my_dynamic_search_target.target_slices.0.instances=my_instance
st.my_dynamic_search_target.target_slices.0.power=0
st.my_dynamic_search_target.target_slices.1.build_groups=bg1
st.my_dynamic_search_target.target_slices.1.slices=42
st.my_dynamic_search_target.target_slices.1.instances=my_instance
st.my_dynamic_search_target.target_slices.1.power=1

```

Textual Relevance Parameters

Relevance

Boolean parameter to enable all relevance computation.

Globally disabling relevance computation disables the following features:

- term scoring
- proximity scoring
- sorting
- node properties
- using ranking elements for faceting
- using ranking elements for facet aggregations
- retrieving ranking elements

Note: If your query contains grouping, either by a Search API parameter or in the product configuration set in the Administration Console > Search Logic, the relevance feature is forced to `true`.

Note: If your query contains a refinement on a disjunctive facet, the relevance feature is forced to `true`.

proximity_max_distance

Maximum distance for proximity matching.

ts (term score)

Sets the term score algorithm. Value must be one of the available algorithms:

- NO_RANKING
- RANK
- TFIDF
- IDF
- RANK_IDF
- RANK_TFIDF
- BM25
- BM25F
- CUSTOM - for this term score algorithm, you must define `ts.expr` to specify the custom expression.

Unranked search mode

Streaming

Boolean parameter. If `true`, all sorts and all limits are ignored.

The answer is NOT cacheable, and is streamed. It is efficient as the synthesis and full hits are computed on-the-fly. It is therefore especially adapted to fetch large results sets without much memory consumption.

In streaming mode, you must iterate over all returned hits, so do not use the `getHits()` method but `getHitsIterator()` instead, and read hits using the `hasNext` and `next` subcalls.

Note: In streaming mode, the `hf` parameter does not control the maximum total number of returned full hits, but the maximum number of returned hits per slice. Therefore, with 4 slices, `streaming=true`, `hf=2000`, up to 8000 hits can be returned.

allow_skip_nhits

Boolean parameter. If `true`, Exalead CloudView stops counting hits that match the query as soon as possible. The `nhits` value is allowed to be incorrect.

This advanced parameter must be used for high performance unranked exports without synthesis.

Search Logic Editing

Search API parameters are the preferred way to change the search logic at query time.

Yet, when modifications made to the search logic are not covered by standard Search API parameters, the **search logic editing** (a.k.a. sle) can be used as a generic way to set, add, or remove values in the search logic.

Unlike Search API parameters, the sle upward compatibility is not guaranteed. Any change made to the search logic during product upgrade may affect the sle.

Add an sle

It is possible to add one or more sle to the search URL with the following query parameters:

- `set:[path]=[new value]` - where `path` identifies the object field or list element to be replaced by the new value. The previous value is deleted.

Example: The following example set the `performMAX` Boolean of the query prefix handler named `profiles` to `true`

```
sle=set:uQLConfig.queryPrefixHandler[name="profiles"].performMAX=true
```

- `add:[path]=[new list element]` - where `path` identifies the list element before which the new value is inserted. The list element and any subsequent elements are shifted to the right.

Example: Add a full text prefix handler

```
sle=add:uQLConfig.queryPrefixHandler=FullTextPrefixHandler
(name="t",indexFields="title",dictionaryName="dict0",matchingMode="normaliz
```

- `remove:[path]` - where `path` identifies the list element to be removed. The element is deleted and any subsequent elements are shifted to the left.

Example: Remove the first operation of meta named "title"

```
sle=remove:hitConfig.meta[name="title"].metaSpecificOperation[0]
```

Value serialization

value serialization depends on the type.

Value type	Description
<code>null</code>	The <code>null</code> value
<code>boolean</code>	<code>true</code> and <code>false</code>
<code>integer</code>	Digits optionally preceded by a minus sign. For example, 314, -5
<code>double</code>	Floating point value. For example, 3.14, -.5, 1.23E9
<code>string</code>	Sequence of characters between double quotes. Double quotes have to be escaped with <code>\</code> . For example, "abc", "say \"hello\""

Value type	Description
list	Sequence of values separated by commas, between brackets. For example, <code>[1, 42, 5]</code> , <code>["a", "c"]</code>
object	Serialization of a new object. For example, <code>SortBy(name="myobject",expr="title",limit=100,order="asc")</code>

Note:

- values `"5"` and `5` are not equivalent. The first is a string whereas the second is an integer.
- values `""` and `null` are not equivalent. The first is an empty string whereas the second is a null value.

sle Path

The **path** is an element sequence that identifies either an object field or a list element.

The XML representation of the search logic is not representative. There may be missing nodes or case discrepancy.

The path must be built using the javadoc `sdk/java-customcode/docs/api/com/exalead/mercury/mami/search/v20/SearchLogic.html`.

Parameter	Description
<code>hitConfig.meta[n]</code>	Stands for the first element with name equal to <code>"title"</code> in the meta list of the <code>hitConfig</code> object in the Search Logic.
<code>virtualFieldDefi</code>	Stands for the first element of the <code>virtualFieldDefinition</code> list in the Search Logic.

Object serialization

The **object serialization** is made of two parts: `[class] (key=value, ...)`

- The java class identifier can either be the full qualified name of a java class using `.` as package separator and `'$'` for inner classes. For convenience, the simple class name is accepted, and the full name is deduced from the path.

For example, `com.exalead.search.v30.VirtualFieldDefinition` or `VirtualFieldDefinition`

- The initializers is a comma-separated sequence of assignment.

For example,

```
SnippetOperation(highlightFacetIds="Event, Person", highlightExtraPrefixHandling=
maxSentenceSegmentLength=210, maxLength=500)
```


Misc

use_logic_virtual_fields

- `true/false` - (Optional) Removes all virtual fields defined in the search logic. Default is `true`.
- `vfname1,vfname2,vfname3` - Keeps only this list of virtual fields. Be careful, they must exist in the search logic.

avf or add_virtual_field

name:url-escaped-expr

remove_virtual_field

One argument: name of the virtual field to remove. The virtual field can be used in a meta. It can also replace an existing virtual field.

tz or timezone

One argument: time difference to apply in virtual operator `#adjust_timezone(expr)`. Format: `[+-][00...12][00|15|30|45]`

Example: add 4 hours and 30 minutes

```
timezone=+04:30
```

The example below is printed:

```
original_time: 2016/10/02 14:15:16
adjusted_time: 2016/10/02 08:15:16
hit_meta.original_time.expr=document_lastmodifieddate&
hit_meta.original_time.operation.time_formatter1.type=time_format&
hit_meta.adjusted_time.expr=#adjust_timezone(document_lastmodifieddate)&
hit_meta.adjusted_time.operation.time_formatter2.type=time_format&
timezone=-06:00
```

of or output_format

Output format specifier. The following formats are supported:

- `flea` - internal binary representation, used only by the Java Search Client
- `xmlv10,xml` - XML representation
- `json` - JSON representation
- `csv` - CSV representation (do not include synthesis)

- `atom` - Atom representation

callback

JSONP support. If specified, and if the `output_format` parameter is set to `JSON`, wraps the response in a function whose name is given by the parameter.

cache

Key/Value	Description
<code>g=boolean</code>	Must we get in cache? (default is 1)
<code>s=boolean</code>	Must we search? (default is 1)
<code>p=boolean</code>	Must we put in cache? (default is 1)
<code>e=boolean</code>	Must we evict from cache before searching? (default is 0)
<code>f=boolean</code>	Must we force insertion, even if cache is not accepting queries? (default is 0)
Alias	Description
<code>cache=no</code>	Query is not cached and does not looked up in cache --> <code>g:0,p:0</code>
<code>cache=evict</code>	Runs the query from the index, evicts it from the cache if it was in cache --> <code>e:1,p:0</code>
<code>cache=only</code>	Only returns cached results. If there is no cached result, it returns an error --> <code>s:0</code>

Note: Warm-up queries are sent with `&cache=g:0,f:1`.

query_implicit_sequence_operator or qiso

The default query operator. If a user enters this query: `exalead cloudview` (without quotes), by default, it is parsed as "exalead AND cloudview".

- `qiso=AND` parses as "exalead AND cloudview"
- `qiso=OR` parses as "exalead OR cloudview"

The fetch, preview and thumbnail Commands

Theses commands are used to fetch, preview, or or generate a thumbnail for a document. You can use them from the Java `FetchClient`. These commands are suppted by the .NET `Fetcher` class using the built-in properties only.

About Thumbnails

Search Server actions to generate thumbnails

To generate thumbnails, the Search Server:

- Calls the connector fetcher to retrieve the original document from the data source.
- Calculates the thumbnails (it generally uses the convert to do that).
- Sends the document and the calculated thumbnails.

Behavior and tips

- The thumbnail always has the same proportion. If your document does not have the same proportion, a margin will be added to the picture to fill missing gaps.
- The thumbnail is scaled to its MINIMUM value. Let us say you have `width=90` and `height=1200`, the width value is the one guiding the downscale.
- Be careful, if your `BASEPORT+10` is accessible, anyone can generate thumbnails and if your cache is not properly configured, someone malicious could spam thumbnail generations with huge resolution values.
- The value of the source parameter does not necessarily have to be the same as the connector used to index the document. Remind that the source parameter is required to specify a fetcher to the thumbnails process. For example, you can use a JDBC connector to index your documents, and also create a "dummy" Files connector to generate thumbnails as the JDBC connector does not have a fetcher. This workaround is useful only if you have your own search front end, not if you are using a Mashup UI application.

Global Parameters

Parameter	Description
<code>source:string</code>	The source connector name.
<code>uri:string</code>	<p>The URI of the document to fetch. The root used in the configuration of the source connector (file system connector) is important.</p> <p>For example, if the root in the connector is <code>/data/user1/workspace/customers/cv</code> and you have a document named <code>myexample.doc</code>, you must encode the uri parameter as follows: <code>uri=%2F%252Fdata%252Fuser1%252Fworkspace%252Fcustomers%252Fcv%2Fmyexample.doc</code></p> <p>Otherwise you get a bad file url root exception.</p>

Parameter	Description
<code>security:string</code>	Security token for the fetch request
<code>enforce_security:boolean</code>	Specifies whether security tokens must be enforced
<code>all_parts:boolean</code>	Requests all parts of the document
<code>part:string</code>	Selects the part of the document to retrieve

Fetch Parameters

It is available at `http://SEARCH_API_HOST:SEARCH_API_PORT/fetch`

Parameter	Description
<code>override_filename:boolean</code>	Forces the file name in the Content-Disposition header
<code>filename:string</code>	New file name to use
<code>override_contenttype</code>	Forces the MIME Content-type in the Content-Disposition header
<code>contentType</code>	New MIME Content-type to use

Preview Parameters

It is available at: `http://SEARCH_API_HOST:SEARCH_API_PORT/preview?uri=uri_document&source=connector_name&q=%23all`

Parameter	Description
<code>q:string</code>	User query string
<code>l</code>	User query language
<code>start_page:integer</code>	Page of the document to preview
<code>pages:integer</code>	Number of pages of the document to render
<code>rewrite_base:boolean</code>	Base HTTP path for links rewriting. You should set this to the base URL under which you are proxying the preview request.

Thumbnail Parameters

It is available at `http://SEARCH_API_HOST:SEARCH_API_PORT/thumbnail`

Parameter	Description
<code>width:integer</code>	The width in pixels of the image generated. If not set, the default value in the <code>searchApi.xml</code> is used.
<code>height:integer</code>	The height in pixels of the image generated. If not set, the default value in the <code>searchApi.xml</code> is used.
<code>start_page:integer</code>	Use this parameter to specify the page of the document to thumbnail. For example, a value of 1 is for page 1.

The Search Results

This section describes the main results of the Search command.

Result	Description
<code>estimated="true"</code>	When the synthesis or the number of results is not exact, because of errors linked to timeout, limits or data sampling.
<code>status="limited"</code>	When the query is stopped because it has reached the "heap sort" limit. It occurs only when the limitation comes from the Search Server, not from the slice.
<code>status="timeout"</code>	When the query has been either partially or totally stopped because it reached the timeout limit.
If the query status is not error/timeout	<ul style="list-style-type: none"> if the query returned fewer hits / partial synthesis because of the use of <code>max_fetched_hits</code>, then: <ul style="list-style-type: none"> <code>status="limited"</code> in heap sort <code>status="ok"</code> in all other modes (local sort, unrankedsort, and unrankedstream) if the query returned fewer hits / partial synthesis because of the use of <code>max_fetched_hits_per_slice</code>, then: <ul style="list-style-type: none"> <code>status="ok"</code> in all modes because information of limited answer is not communicated from the index to the Search Server.

The spellcheck Command

The `spellcheck` or `sc` command provides spell checking. It is available at `http://SEARCH_API_HOST:SEARCH_API_PORT/spellcheck`.

You can use it from the java `SpellCheckClient`.

It supports all search command parameters. You can use:

- Boolean
- or the keys from the KV map:
 - `enabled:true`
 - `use_with_refinements:true`
 - `disable_after:10`
 - `max_suggestions:1`
 - `compute_nb_of_hits:true`
 - `remove_weak_suggestions:true`
 - `automatically_correct:false`

The suggest Command

This command provides search suggest. It is available at:

- `http://SEARCH_API_HOST:SEARCH_API_PORT/suggest/service/SUGGEST_NAME`
- or `http://SEARCH_API_HOST:SEARCH_API_PORT/suggest/dispatcher/DISPATCHER_NAME`

You can use it from the Java `SuggestClient` client. These commands are also supported by the .NET `Suggester` class using the built-in properties only.

Parameter	Description
<code>q:string</code>	The input query
<code>distance:integer</code>	The suggest dictionaries supports fuzzy matching at runtime. This sets the maximum Levenshtein distance between the input string and the suggestion. 0 means exact match.

Parameter	Description
<code>minLenForDist1:integer</code>	Only searches for distance 1 fuzzy matches if the original word in the query is at least N characters long. This avoids too much approximation on very short words. The best value is 3 characters.
<code>minLenForDist2:integer</code>	Only searches for distance 2 fuzzy matches if the original word in the query is at least N characters long. This avoids too much approximation on very short words. The best value is 6 characters.
<code>logic:string</code>	Specify a Search Logic name.
<code>exhaustive:boolean</code>	Displays exhaustive results.
<code>recurse:boolean</code>	Suggests new matches on query words recursively.
<code>autocomplete:boolean</code>	Suggests matches for the last word only.
<code>output:string</code>	<p>Output format:</p> <ul style="list-style-type: none"> <code>xml</code> - returns a complete output, with text suggestions, score, distance. <code>json</code> - returns text suggestions only. Other search output format such as csv, flea, and atom, are not supported. <p>Note: The Accept HTTP header is also taken into account if output is not specified.</p>
<code>security:string</code>	<p>Allows you to take users' security tokens into account in the search suggest. For example, <code>&security=TOKEN1&security=TOKEN2...</code></p> <p>Negative tokens must be preceded by <code>~</code>. You can declare positive and negative tokens one after the other, for example, <code>&security=TOKEN1&security=~TOKEN2</code></p>

The security Command

This Search API command provides security checks. It is available at `http://SEARCH_API_HOST:SEARCH_API_PORT/security/SOURCE_NAME`.

It can be used from the java `SecurityClient`. It also is supported by the .NET `AuthenticationClient` class using the built-in properties only.

Parameter	Description
<code>login:string</code>	User login

Parameter	Description
<code>password:string</code>	User password
<code>checkPassword:boolean</code>	Checks the user password

The expansion Command

This Search API command provides query expansions. It is available at `http://SEARCH_API_HOST:SEARCH_API_PORT/expansion`.

It supports all `search` command parameters.

The introspection Command

This Search API command provides search introspection. It is available at:

- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/introspection/logics` - search logic introspection
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/introspection/targets` - search target introspection
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/introspection/prefixes` - prefix handlers introspection
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/introspection/sorts` - sort introspection
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/introspection/facets` - facets introspection
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/introspection/metastats` - metastats introspection
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/introspection/expr_elements` - all potential expressions introspection
- `http://SEARCH_API_HOST:SEARCH_API_PORT/search-api/introspection/all` - all introspection

It supports all Search API command parameters.

Appendix - Virtual Field Expression Syntax

Virtual fields allow you to compute values from many elements of the CloudView index. The main purpose of a virtual field is to access stored index fields. For example, a virtual field called `revenue` with expression `price * quantity` accesses the two fields and calculate the total price.

You can then use virtual fields for multiple areas within your search application. For example, for a given hit, virtual fields can calculate:

- A meta to display in the search client.
- The value for the hit in a dynamic numerical facet.
- The value for the hit in a facet aggregation.
- Ranking elements.
- A value to use for filtering queries. For example, you can define a numerical prefix handler, `total_price`, allowing a user to make queries directly on the total price, such as `total_price: > 500`

What Is a Virtual Field Expression

A virtual field expression is either:

- a constant (for example `3` or `42.0`)
- a numerical index field (for example, `my_int_field`)
- a ranking key defined in the query, prefixed with `@` (for example, `@my_ranking_key`)
- a call to a built-in function, which can have arguments that are valid virtual field expressions
- an n-ary numerical operator (n=1,2,3) applied to n virtual field expressions

Expression Types

Virtual field expressions are entered, and have either `int` or `float` types.

- `int` values are represented as 64-bit signed integers
- `float` values are 64-bit IEEE doubles
- Boolean values are represented as integers with `true != 0`.

The type of an expression is given by the following rules:

- A constant with decimal separator has `float` type.
- A constant without decimal separator has `int` type.
- Typed functions have an explicit type. The type of each such function is given in the documentation below.
- Nontyped functions have a type that depends on the type of their arguments. If not otherwise stated, a function with only integer arguments has `int` type, and a function with mixed or float arguments has `float` type (for example, `4.2 + 4` is `8.2`).

Numerical Operators

Numerical operators compose virtual field expressions to produce another valid virtual field expression. They are, by order of decreasing precedence:

Operator	Type	Description
<code>-</code>	unary	Minus operator
<code>!</code>	unary	Logical not. Returns <code>1</code> if <code>expr</code> is zero, whatever its type.
<code>~</code>	unary	bitwise not. Perform float-to-int conversion.
<code>*</code> , <code>/</code> , <code>%</code>	binary	Multiplication, division, modulo operators
<code>+</code> , <code>-</code>	binary	Addition, subtraction operators
<code><<</code> , <code>>></code>	binary	Left/right shift operators. These operators always perform float-to-int conversion if given float arguments. Same behavior as <code>c++</code> signed shifts.
<code>==</code> , <code>!=</code> , <code>>=</code> , <code><=</code> , <code>></code> , <code><</code>	binary	Comparison operators (Warning: <code>=</code> is NOT supported).
<code> </code> (or), <code>&</code> (and), <code>!^</code> (xor), <code>~</code> (not)	binary	Bitwise operators. These operators always perform float-to-int conversion if given float arguments.
<code>&&</code> , <code>! </code>	binary	Logical and operators. Evaluation is lazy, meaning that the right side is not evaluated if the left size is <code>false</code> (resp. <code>true</code>).
<code>expr ? expr</code>	ternary	<code>expr</code> if/then/else operators
<code>?=</code>	binary	Fallback operator. If the expression on the left has a value, use that value. Else, use the value of the expression on the right. Evaluation is lazy.

Operator	Type	Description
		For example, this is useful to define default values for fields or ranking elements. Example: <code>@proximity?=3 + my_field?=2</code>

Built-ins

In the following, `expr` represents an assessable virtual field expression. `fd` denotes a field dependant type.

General Functions

Function	Type	Description
<code>#did()</code>	int	Returns the id of the current document
<code>#slice()</code>	int	Returns the current slice

Mathematic Functions

Function	Type	Description
<code>#random()</code>	float	Returns a uniform double between 0.0 and 1.0. NOTE: A new value is generated for every invocation of this function. For example, <code>2*random()</code> is uniform, but <code>#random()+#random()</code> is not, just as rolling two dice and taking the sum results in more sevens than twos or twelves.
<code>#hash(expr)</code>	float	Returns a hash of the expression in argument between 0.0 and 1.0
<code>#round(expr)</code>	int	Returns the value of the expression rounded to the closest integer
<code>#round(expr, precision)</code>	float	Returns the value of the expression rounded to precision digits after comma
<code>#floor(expr)</code>	int	Returns the value of the expression rounded to the closest lower integer
<code>#ceil(expr)</code>	int	Returns the value of the expression rounded to the closest upper integer

Function	Type	Description
<code>#exp(expr)</code>	float	Returns the base-e exponential function
<code>#log2(expr)</code>	float	Returns the base-2 logarithm function
<code>#ln(expr)</code>	float	Returns the base-e logarithm function
<code>#cos(expr)</code>	float	Returns the cosine function
<code>#sin(expr)</code>	float	Returns the sine function
<code>#tan(expr)</code>	float	Returns the tangent function
<code>#sqrt(expr)</code>	float	Returns the square root of <code>expr</code>
<code>#abs(expr)</code>	int	Returns the absolute value of <code>expr</code>
<code>#inrange(expr, minExpr, maxExpr)</code>	int	Returns <code>true</code> if <code>expr</code> is in the range <code>[minExpr;minExpr]</code>
<code>#stddev(expr, expr, expr, ...)</code>	float	(n-ary) Returns the population standard deviation of N expressions
<code>#avg(expr, expr, expr, ...)</code>	float	(n-ary) Returns the average value of N expressions
<code>#min(expr, expr, expr, ...)</code>	fd	(n-ary) Returns the minimum value of N expressions
<code>#max(expr, expr, expr, ...)</code>	fd	(n-ary) Returns the maximum value of N expressions
<code>#countif(operator, baseExpr, expr, expr, expr, ...)</code>	int	<p>(n-ary) Returns the number of <code>expr</code> expressions matching the relation <code>expr operator baseExpr</code>.</p> <p>Example: <code>#countif(==, 42, document_foo, document_bar, document_baz)</code> returns the number of fields between <code>foo</code>, <code>bar</code> and <code>baz</code> in the class <code>document</code> that equals 42</p>

Geographic Functions

Function	Type	Description
<code>#lat(expr)</code>	fd	Returns the first component of a point (the latitude, or <code>x</code> in cartesian mode)

Function	Type	Description
<code>#lng(expr)</code>	fd	Returns the second component of a point (the longitude, or <i>y</i> in cartesian mode)
<code>#dist(point_field, expr_lat, expr_lng)</code>	int	Returns the distance in meters between a point field and the point in argument
<code>#dist_latlong(lat_field, lng_field, expr_lat, expr_lng)</code>		Similar to <code>#dist</code> , for GPS coordinates, with the latitude in a numerical field and longitude in another numerical field
<code>#dist_eucl(x_field, y_field, expr_x, expr_y)</code>		Similar to <code>#dist</code> , for cartesian coordinates, with the x-coordinate in a numerical field and y-coordinate in another numerical field

Category Functions

Function	Type	Description
<code>#children_count("Top/path/to/root", categoriesField)</code>	int	Returns the number of categories in the document under the root <code>Top/path/to/root</code> in the <code>categoriesField</code> . For example, you can get the number of people referenced in a document with: <code>#children_count("Top/people", categories)</code>
<code>#cat_corpus_count("Top/path/to/cat", categoriesField)</code>	int	Returns the number of documents that have the given category in the current slice
<code>#has_category("Top/path/to/cat", categoriesField)</code>	int	Returns 1 if the document has the category, otherwise 0

Time Manipulation Functions

Time is represented using an internal index representation, which is not a timestamp and must not be directly manipulated. We provide the following functions to manipulate time in virtual field expressions.

Note: Since months and years do not always have the same durations, there are no `nmonths()` and `nyears()` functions.

We provide a syntactic sugar to manipulate time expressions. It is a basic set of mathematical operators for computation on date and time at query time through virtual field expressions. You can use the `y, m, w, d, H, M, S` suffixes to define expressions like: `#now() + 1d`, `#now() - 2y`, etc.

Function	Type	Description
<code>#now()</code> (or <code>#now</code>)	long	Returns the current time in internal index representation
<code>#datetime(year, month, day, hour, minute, second)</code>	long	Creates an index time from a human-readable time. <code>month</code> is between 1 and 12 and <code>day</code> between 1 and 31. If <code>hour</code> , <code>minute</code> or <code>second</code> are omitted, they default to 0. Any of <code>year</code> , <code>month</code> , etc. can be virtual field expressions.
<code>#datetime_from_date(dateExpr)</code>	long	Creates a datetime from a date virtual field expression <code>dateExpr</code> . For example, <code>DateFacets</code> .
<code>#fromunixts(ts)</code>	long	Creates an index time from the UNIX timestamp <code>ts</code>
<code>#tounixts(time)</code>	long	Returns the UNIX timestamp corresponding to the given index time
<code>#year(time)</code>	long	Returns the year for the given index time
<code>#month(time)</code>	long	Returns the month for the given index time (January is 1)
<code>#day(time)</code>	long	Returns the day for the given index time (1-based).
<code>#weekday(time)</code>	long	Returns the day of the week for the given index time (0=Sunday, 6=Saturday)
<code>#hour(time)</code>	long	Returns the hour for the given index time
<code>#minute(time)</code>	long	Returns the minute for the given index time
<code>#second(time)</code>	long	Returns the second for the given index time
<code>#nweeks(time1, time2)</code>	long	Returns the (signed) number of completely elapsed weeks between index times <code>time1</code> and <code>time2</code>
<code>#ndays(time1, time2)</code>	long	Returns the (signed) number of completely elapsed days between index times <code>time1</code> and <code>time2</code>
<code>#nhours(time1, time2)</code>	long	Returns the (signed) number of completely elapsed hours between index times <code>time1</code> and <code>time2</code>
<code>#nmins(time1, time2)</code>	long	Returns the (signed) number of completely elapsed hours between index times <code>time1</code> and <code>time2</code>

Function	Type	Description
#nsecs (time1, time2)	long	Returns the (signed) number of completely elapsed hours between index times <code>time1</code> and <code>time2</code>
#yesterday ()	long	Returns yesterday time in internal index representation
#years_ago (N)	long	Returns, in internal index representation, the time of N years ago
#months_ago (N)	long	Returns, in internal index representation, the time of N months ago
#weeks_ago (N)	long	Returns, in internal index representation, the time of N weeks ago
#days_ago (N)	long	Returns, in internal index representation, the time of N days ago
#hours_ago (N)	long	Returns, in internal index representation, the time of N hours ago
#minutes_ago (N)	long	Returns, in internal index representation, the time of N minutes ago
#seconds_ago (N)	long	Returns, in internal index representation, the time of N seconds ago
#addperiod (time, ndays, long		Returns a new index time differing from the original one. It is calculated using the specified units (ndays, nhours, ...) which can be positive or negative integers.
#adjust_timezone (time)	long	Returns a new index time adjusted to the timezone specified in the end user's query input. Note: By default, CloudView stores date time index fields in UTC format.
#parse_date (date_string, optional_format)	long	Creates an index time from given date string. If no <code>optional_format</code> is given, <code>%m/%d/%Y</code> is used.
#parse_time (datetime_string, optional_format)	long	Creates an index time from given datetime string. If no <code>optional_format</code> is given, <code>%m/%d/%Y-%H:%M:%S</code> is used.

String Functions

For alphanumeric fields

Function	Type	Description
#tf(field)	int	Number of terms in <code>field</code> . <code>storeTf</code> must be enabled

For both alphanumeric and value fields

Function	Type	Description
#regex_count(field, 'pattern')	int	Returns the number of occurrences of pattern in the content of <code>field</code>
#regex_match(field, 'pattern')	int	Returns 1 if pattern matches the content of <code>field</code>
#strlen(field)	int	Number of characters in <code>field</code>
#strhash(field)	int	Returns the hash64 of the content of <code>field</code>
#strcmp(field, field or string)	int	Compares the content of a field (<code>s1</code>) to the content of another field, or a string (<code>s2</code>). It returns an integer less than, equal to, or greater than zero if <code>s1</code> is found, respectively, to be less than, to match, or be greater than <code>s2</code> .
#strlower(field)	int	Returns the lowercase string content
#strncmp(field, field or string, long)	int	Compares the first <code>n</code> characters of two strings <code>s1</code> (field) and <code>s2</code> (field or string). It returns an integer less than, equal to, or greater than zero if <code>s1</code> is found, respectively, to be less than, to match, or be greater than <code>s2</code>
#strnormalize(field)	int	Returns the normalized string content
#strstr(field, field or string)	int	Looks for the first occurrence of a substring (field or string) in the content of another field

Multivalued fields Manipulation Functions

When a numerical/alphanumeric/value field is multivalued, all comparison operators can be used but the search policy must be explicit. For example, with `==`:

- (int boolean) #any(multivaluedField, ==, exp) - returns 1 if at least one value is equal to exp
- (int boolean) #all(multivaluedField, ==, exp) - returns 1 if all the values are equal to exp
- (int) #countif(multivaluedField, ==, exp) - returns the number of values that are equal to exp

Operators ==, !=, <, <=, >, >= can be used.

Function	Type	Description
#min(multivaluedField)	fd	Returns the minimal value of a multivalued field (numerical and alphanumeric fields are supported)
#max(multivaluedField)	fd	Returns the maximal value of a multivalued field (numerical and alphanumeric fields are supported)
#length(multivaluedField)	int	Returns the number of values in a multivalued field. It also works for nonmultivalued fields.
#sum(multivaluedField)	fd	Returns the sum of values in a multivalued field (numerical fields are supported)
#avg(multivaluedField)	float	Returns the average value of values in a multivalued field (numerical fields are supported)
#stddev(multivaluedField)	float	Returns the population standard deviation of values in a multivalued field (numerical fields are supported)

Dynamic Fields Manipulation Functions

Function	Description
#extract(dynamicField, "meta name")	Returns the value of "meta name" in the dynamic field
#extracthasvalue(dynamicField, "meta name")	Returns 1 if "meta name" is valued in the dynamic field, and 0 if not.

Type Casting

Function	Type	Description
#int(expr)	int	Returns the result of <code>expr</code> as a integer. A truncation is performed (not a round).

Function	Type	Description
<code>#float(expr)</code>	float	Returns a float-typed <code>expr</code> with the floating-point value of <code>expr</code>

Special Functions

Function	Type	Description
<code>#dscore(expr, s0, s1, x1, x2)</code>	int	<ul style="list-style-type: none"> If <code>expr = 0</code> - returns 0. If $0 < \text{expr} < x1$ - returns linear interpolation between <code>s0</code> and <code>s1</code>. If $x1 < \text{expr} < x2$ - returns linear interpolation between <code>s1</code> and 0. If $x2 < \text{expr}$ - returns 0. Can be used to implement geo distance ranking.

Ranking Elements

Ranking elements can be retrieved using the `@` notation. For example, `@term.score` retrieves the global score of the terms; `@a` returns the user key `a`.

When a node is named with the syntax `name="thename"`, and the node matched, some ranking values become accessible:

- `@thename.matched` returns 1
- `@thename.npos` returns the number of positions in this node
- `@thename.score` returns the local score of the node
- Alphanum nodes also have `@thename.rank` that returns the rank of the term and `@thename.tfidf` that returns the tfidf of this term.
- Distance nodes also have `@thenode.distance` that returns the distance of the document to the center of the distance.