
# Federated Content Search for Lexical Resources (LexFCS)

## *Specification*

Erik Körner, Thomas Eckart, Uwe Kretschmer, Axel Herold, Frank Wiegand, Frank Michaelis, Claus Zinn, Matthias Bremm, Louis Cotgrove, Thorsten Trippel, Felix Rau, Anne Klee, Daniel Werning, Dominik Blöse, Jan Oliver Rüdiger, Gaetano Manzone, Hagen Beelitz

Version 1.0, 2026-09-16

This publication was created in the context of the work of the association German National Research Data Infrastructure (NFDI e.V.). NFDI is financed by the Federal Republic of Germany and the 16 federal states, and the consortium Text+ is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 460033370. The authors would like to thank for the funding and support. The authors would also like to thank CLARIN ERIC for funding and support.

# Table of Contents

# Chapter 1. Introduction

The *Federated Content Search for Lexical Resources (LexFCS)* specification is an extension of the *CLARIN Federated Content Search (CLARIN-FCS) - Core 2* specification that allows search and retrieval of *lexical resources* including dictionaries, encyclopedias, normative data, terminological databases, ontologies etc.

## 1.1. Terminology

The key words `MUST`, `MUST NOT`, `REQUIRED`, `SHALL`, `SHALL NOT`, `SHOULD`, `SHOULD NOT`, `RECOMMENDED`, `MAY`, and `OPTIONAL` in this document are to be interpreted as in RFC2119.

## 1.2. Glossary

**NOTE** Based on *Glossary* in FCS Core 2 specification.

**CLARIN-FCS, FCS**

CLARIN federated content search, an interface specification to allow searching within resource content of repositories.

**Client**

A software component, which implements the interface specification to query endpoints, i.e. an aggregator or a user interface.

**CQL**

Contextual Query Language, previously known as Common Query Language, is a domain-specific language for representing queries to information retrieval systems, such as search engines, bibliographic catalogs and museum collection databases.

**Data View**

A Data View is a mechanism to support different representations of search results, e.g. a "hits with highlights" view, an image, or a geolocation.

**Endpoint**

A software component, which implements the CLARIN-FCS interface specification, and translates between CLARIN-FCS and a search engine.

**Hit**

Data or a subdivision of data returned by a Search Engine that matches the search criterion. What is considered a Hit highly depends on the Search Engine.

**Interface Specification**

Common harmonised interface and suite of protocols that repositories are required to implement.

**Resource**

A searchable and addressable entity offered by an endpoint, such as a text corpus or a dictionary.

**Result Set**

An (ordered) set of Hits that match a search criterion produced by a search engine as the result of processing a query.

**Search Engine**

A software component within a repository that allows for searching within the repository contents.

**SRU**

Search and Retrieve via URL is a protocol for Internet search queries. Originally introduced by Library of Congress LOC-SRU12, later standardisation process moved to OASIS OASIS-SRU12, OASIS-SRU20.

## 1.3. Normative References

**NOTE** Based on *Normative References* in FCS Core 2 specification.

**RFC2119**

Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, March 1997, https://www.ietf.org/rfc/rfc2119.html

**XML-Namespaces**

Namespaces in XML 1.0 (Third Edition), W3C, 8 December 2009, http://www.w3.org/TR/2009/REC-xml-names-20091208/

**OASIS-SRU-Overview**

searchRetrieve: Part 0. Overview Version 1.0, OASIS, January 2013, http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/os/part0-overview/searchRetrieve-v1.0-os-part0-overview.html (DOC), (PDF)

**OASIS-SRU12**

searchRetrieve: Part 2. SRU searchRetrieve Operation: APD Binding for SRU 1.2 Version 1.0, OASIS, January 2013, http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/os/part2-sru1.2/searchRetrieve-v1.0-os-part2-sru1.2.html (DOC), (PDF)

**OASIS-SRU20**

searchRetrieve: Part 3. SRU searchRetrieve Operation: APD Binding for SRU 2.0 Version 1.0, OASIS, January 2013, http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/os/part3-sru2.0/searchRetrieve-v1.0-os-part3-sru2.0.html (DOC), (PDF)

**OASIS-CQL**

searchRetrieve: Part 5. CQL: The Contextual Query Language version 1.0, OASIS, January 2013,

http://docs.oasis-open.org/search-ws/searchRetrieve/v1.0/os/part5-cql/searchRetrieve-v1.0-os-part5-cql.html (DOC), (PDF)

**LOC-SRU12**

SRU Version 1.2: SRU Search/Retrieve Operation, Library of Congress, http://www.loc.gov/standards/sru/sru-1-2.html

**LOC-CQL**

The *Contextual Query Language*, Library of Congress, https://www.loc.gov/standards/sru/cql/, see also OASIS-CQL

**LOC-CQLCS**

The *CQL Context Set*, Library of Congress, https://www.loc.gov/standards/sru/cql/contextSets/theCqlContextSet.html

**CLARIN-FCSCore2**

CLARIN Federated Content Search (CLARIN-FCS) - Core 2 specification, SCCTC FCS Task-Force, https://www.clarin.eu/content/federated-content-search-core-specification, (Sources), (HTML Build), (PDF Build),

## 1.4. Non-Normative References

**UD**

Universal Dependencies project, that provides terminology for linguistic tags, features and relations, https://universaldependencies.org/

**EDTF**

Extended Date/Time Format (EDTF) Specification, February 4, 2019, The Library of Congress, https://www.loc.gov/standards/datetime/

## 1.5. Typographic and XML Namespace conventions

The following typographic conventions will be used throughout this specification:

- `string`

  A LexFCS field type, a LexCQL query or an XML fragment.

The following typographic conventions for XML fragments will be used throughout this specification:

- `<prefix:Element>`

  An XML element with the Generic Identifier *Element* that is bound to an XML namespace denoted by the prefix *prefix*.

- `@attr`

An XML attribute with the name *attr*.

- `string`

  The literal *string* as the content of an element or value of an attribute.

Endpoints and clients `MUST` adhere to the XML-Namespaces specification. The CLARIN-FCS interface specification generally does not dictate whether XML elements should be serialised in their prefixed or non-prefixed syntax, but endpoints `MUST` ensure that the correct XML namespace is used for elements and that XML namespaces are declared correctly. Clients `MUST` be agnostic regarding syntax for serializing the XML elements, i.e. if the prefixed or un-prefixed variant was used, and `SHOULD` operate solely on *expanded names*, i.e. pairs of *namespace name* and *local name*.

For a list of common XML namespace names and prefixes see the table "XML Namespaces and prefixes" in Section 1.5 of the FCS Core 2 Specification.

# Chapter 2. Summary and Interface Specification

## 2.1. Summary of Changes

This specification extends the *CLARIN Federated Content Search (CLARIN-FCS) - Core 2* specification in the following ways:

- introducing a new query language, LexCQL, based on the Contextual Query Language (CQL) to allow querying lexical resources,
- extending the basic Hits Data View for inline markup and adding a new Lexical Data View,
- extending the CLARIN-FCS *Endpoint Description* with the *Lexical Search* Capability.

## 2.2. Discovery

The CLARIN-FCS SRU *explain* response is extended by adding the *Lexical Search* capability and *Lexical* Data View to the *Endpoint Description* to support client auto-configuration.

### 2.2.1. Capabilities

The *Lexical Search* capability indicates to clients that the FCS endpoint supports searches through lexical resources using LexCQL and serializes results in the LexFCS Data Views.

*Table 1. New Lexical Search Capability*

| Name | Capability Identifier | Summary |
|---|---|---|
| *Lexical Search* | `http://clarin.eu/fcs/capability/lexical-search/1.0` | Structured search in lexical resources |

### 2.2.2. Endpoint Description

The *Endpoint Description* is extended by a mandatory `<ed:SupportedLexFields>` element describing supported lexical fields for querying and results. The `<ed:SupportedDataView>` element supports the additional Lexical Data View which has its own MIME type and `@id` value.

The `<ed:EndpointDescription>` element is extended by

- one `<ed:SupportedLexFields>` element (`REQUIRED` if endpoint supports *Lexical Search*)

  A list of field types that are generally supported by this endpoint. This list is composed of one or more `<ed:SupportedLexField>` elements. The content of a `<ed:SupportedLexField>` `MUST` be the identifier of a field type (see Section 3.2, “Field”), e.g. `lemma`. Each `<ed:SupportedLexField>` element `MUST` carry an `@id` attribute. The value of the `@id` attribute is used in the `<ed:Resource>` element to indicate, which field types are supported by each resource (see below).

This list MUST NOT include duplicate entries regarding the content of the <ed:SupportedLexField> elements and their @id identifier.

The value of the @id attribute MUST NOT contain the characters , (comma) or ; (semicolon).

The <ed:Resource> element is extended by

- one <ed:AvailableLexFields> element (REQUIRED if endpoint supports *Lexical Search* capability)

  The <ed:AvailableLexFields> element MUST carry a @ref attribute that contains a whitespace-separated list of id values that correspond to the value of the appropriate @id attribute for the <ed:SupportedLexFields> elements that are referenced.

  In case of sub-resources, each Resource SHOULD support all field types that are supported by the parent resource. However, every resource MUST declare all supported field types independently, i.e. there is no implicit inheritance semantic.

*Example of an Endpoint Description with Lexical Search support*

```
<ed:EndpointDescription xmlns:ed="http://clarin.eu/fcs/endpoint-description"
version="2">
  <ed:Capabilities>
    <ed:Capability>http://clarin.eu/fcs/capability/basic-search</ed:Capability>
    <ed:Capability>http://clarin.eu/fcs/capability/lexical-search/1.0</ed:Capability>
  </ed:Capabilities>
  <ed:SupportedDataViews>
    <ed:SupportedDataView id="hits" delivery-policy="send-by-default">application/x-
clarin-fcs-hits+xml</ed:SupportedDataView>
    <ed:SupportedDataView id="lex" delivery-policy="send-by-default">application/x-
clarin-fcs-lex+xml</ed:SupportedDataView>
  </ed:SupportedDataViews>
  <ed:SupportedLexFields>
    <ed:SupportedLexField id="any">any</ed:SupportedLexField>
    <ed:SupportedLexField id="lang">lang</ed:SupportedLexField>
    <ed:SupportedLexField id="entryId">entryId</ed:SupportedLexField>
    <ed:SupportedLexField id="lemma">lemma</ed:SupportedLexField>
    <ed:SupportedLexField id="pos">pos</ed:SupportedLexField>
    <ed:SupportedLexField id="baseform">baseform</ed:SupportedLexField>
    <ed:SupportedLexField id="sentiment">sentiment</ed:SupportedLexField>
  </ed:SupportedLexFields>
  <ed:Resources>
    <ed:Resource pid="hdl:11022/0000-0007-FE0F-C">
      <ed:Title xml:lang="de">SentiWS</ed:Title>
      <ed:Title xml:lang="en">SentiWS</ed:Title>
      <ed:Description xml:lang="de">Der SentimentWortschatz, oder kurz SentiWS, ist
eine öffentlich verfügbare deutschsprachige Ressource für die Sentiment Analyse,
Opinion Mining und ähnliche Zwecke. ...</ed:Description>
      <ed:Description xml:lang="en">SentimentWortschatz, or SentiWS for short, is a
publicly available German-language resource for sentiment analysis, opinion mining
etc. ...</ed:Description>
```

```
      <ed:LandingPageURI>https://hdl.handle.net/11022/0000-0007-FE0F-
C</ed:LandingPageURI>
      <ed:Languages>
        <ed:Language>deu</ed:Language>
      </ed:Languages>
      <ed:AvailableDataViews ref="hits lex" />
      <ed:AvailableLexFields ref="any lang entryId lemma pos baseform sentiment" />
    </ed:Resource>
  </ed:Resources>
</ed:EndpointDescription>
```

The Example of an Endpoint Description with Lexical Search support shows a simple Endpoint Description for an endpoint that supports the *Lexical Search* Capability and provides the Lex Data View, which is indicated by a `<ed:SupportedDataView>` element. It only provides one resource identified by the persistent identifier `hdl:11022/0000-0007-FE0F-C`. For this resource basic metadata such as multilingual titles, descriptions and a landing page is provided. The predominant language of the resource's content is German.

The Generic Hits Data View and the Lex Data View are supported for this resource, because the `<ed:AvailableDataViews>` element references the `<ed:SupportedDataView>` elements via `@id` for both `hits` and `lex`. The element `<ed:AvailableLexFields>` references seven lex field types from the list of all field types the endpoints supports in `<ed:SupportedLexFields>`. Each supported field type is referenced in the space separated list of the `@ref` attribute of the `<ed:AvailableLexFields>` element pointing to the `@id` identifiers of `<ed:SupportedLexField>` elements.

## 2.3. Searching and Result presentation

Queries `MUST` be formulated using the LexCQL query language. Results `MUST` be serialised using both the *Generic Hits* Data View as defined in the CLARIN FCS Core Specification and the Lexical Data View. For the *Generic Hits* Data View, the Extension of the Hits Data View for LexFCS `MAY` be used, which supports limited inline markup to provide extra context to Hits.

### 2.3.1. Request parameter x-fcs-lex-fields

This specification defines an optional request parameter `x-fcs-lex-fields` for the operation *searchRetrieve*. This parameter allows clients to specify the set of returned field types by providing a comma separated list of *Field Type Identifiers* as described in Table 3, "Lexical Search Field types".

It overrides the server's standard behavior and `MUST` only be used in conjunction with the Lexical Data View or `MAY` be ignored silently. The parameter only modifies the result presentation, not the search process itself: requested field types that are not part of a matching entry `MUST` be ignored; no "null" or empty elements `MAY` be created.

# Chapter 3. LexFCS Data Model

The *LexFCS* data model represents information about a single lexical record in a `<lex:Entry>` element and makes no general assumptions about the granularity, type or structure of the underlying lexical resource or its elements.

## 3.1. Entry

Each `<lex:Entry>` element consists of at least one `<lex:Field>` element, each representing a specific kind of information.

### Entry attributes

The `<lex:Entry>` element `MAY` provide the default content language in IETF BCP 47 of the whole entry via the `@xml:lang` attribute. If more information is required, the `OPTIONAL` `@langUri` attribute `MAY` additionally be provided. Language specified on individual `<lex:Value>` elements overrides the `<lex:Entry>` language.

*Table 2. Allowed attributes of element `<lex:Entry>`*

| Value Attribute | Description | Data Type | Examples (without quotes) |
|---|---|---|---|
| `xml:lang` | Content language code in IETF BCP 47 | String | "de", "egy-Egyp", "cop-x-cops" |
| `langUri` | URI referencing information about the content language, if not expressable using BCP 47 | URI | "https://en.wikipedia.org/wiki/Early_New_High_German" |

## 3.2. Field

Each `<lex:Field>` element has a type that `MUST` be specified using a `@type` attribute. A `<lex:Field>` element of a specific type `MAY` only occur once. The `<lex:Field>` element of type `lemma` is mandatory. For an overview of all allowed field types and their restrictions, refer to Table 3, “Lexical Search Field types”.

The `@type` attribute value of a `<lex:Field>` element governs the semantics, data type, and mandatory / optional attributes of all its `<lex:Value>` elements. A `<lex:Field>` element `MUST` contain at least one `<lex:Value>` element, but `MAY` contain an unlimited number of `<lex:Value>` elements.

*Table 3. Lexical Search Field types*

| Field Type Identifier | Short Description | Value Data Type | Value Examples (without quotes) |
|---|---|---|---|
| `lemma` | Lemma form, **mandatory** | String | "Dog", "cat", "walking", "better" |

| Field Type Identifier | Short Description | Value Data Type | Value Examples (without quotes) |
|---|---|---|---|
| entryId | Identifier of the current entry | String | "https://purl.org/ili/i54106", "Zwahr-2049-pawk", "104730" |
| phonetic | Phonetic form | String | "təˈdā" |
| transcription | Transcription or transliteration | String | - |
| translation | Translation | String | "pechare" (cup) |
| *Prosaic Descriptions* | | | |
| definition | Definition or description | String | "A car is a vehicle with an engine [...]" |
| etymology | Etymology information | String | "From Middle English crane, [...]" |
| *Grammar and Morphology* | | | |
| baseform | Baseform (or stem, root, ...) of a lemma | String | - |
| case | Morphological Case | String | "Abl", "ablative", "Nom" |
| degree | Degree of comparison | String | "Sup", "superlative", "Dim" |
| frequency | Frequency information like occurrences, relative frequency, word rank or frequency class | String | "17234", "0.567", "rare" |
| gender | Morphological Gender | String | "Fem", "feminin", "Masc", "Com" |
| grammar | General grammatical information, only to be used if more specific fields can not be supported | String | "part. praet.", "adv., superl." |
| mood | Grammatical feature of verbs, used for signaling modality | String | "Ind", "indicative", "imperative" |
| number | Morphological Number | String | "Dual", "Sing", "Plur" |
| pos | Part-of-Speech | String | "NOUN", "NN", "N", "substantive" |
| segmentation | Composita segmentation, hyphenation; into phonemes, syllables, subwords | String | "Schach+Brett", "to·day", "ins\|ta\|gram\|men" |
| sentiment | Sentiment information | String | "Pos", "positive", "-0.823" |
| tense | Property of verbs indicating chronology | String | "Fut", "future", "past tense" |
| *Relation to other Lexical Entries* | | | |

| Field Type Identifier | Short Description | Value Data Type | Value Examples (without quotes) |
|---|---|---|---|
| antonym, holonym, hypernym, hyponym, meronym, synonym | Semantic relations | String | - |
| related | Unspecified relation | String | - |
| *External References* | | | |
| ref | A URI referencing a related resource | URI | "http://example.org/somewhere/", "https://coptic-dictionary.org/entry.cgi?tla=C2535" |
| senseRef | ID of a sense definition | String | "https://purl.org/ili/i54106", "pwn:02961779-n", "8.10" |
| *Citations / Quotations* | | | |
| citation | A citation, quotation or usage example of this entry's lemma | String | "I got into my car." |

# 3.3. Value

The <lex:Value> element contains the actual information content of a specific information type. Its semantics, data type, and mandatory / optional attributes are governed by the type of the <lex:Field> element in which it is contained.

It is strongly encouraged to use terms of established vocabularies, where feasible. This includes in particular the use of linguistic tags and features of the Universal Dependencies annotation guidelines for the corresponding <lex:Field> types.

## 3.3.1. Generic Value attributes

A <lex:Value> element can be modified with additional attributes, which are, in most cases, optional.

Table 4, "Lexical Search Value attributes" lists all allowed attributes for <lex:Value> elements. Additional information about their usage and cases in which they are mandatory are stated below.

*Table 4. Lexical Search Value attributes*

| Value Attribute | Description | Data Type | Examples (without quotes) |
|---|---|---|---|
| xml:id | XML ID as target for @idRefs attribute | XML ID | "id_sense1" |
| xml:lang | Content language code in IETF BCP 47 | String | "de", "egy-Egyp", "cop-x-cops" |

| Value Attribute | Description | Data Type | Examples (without quotes) |
|---|---|---|---|
| `langUri` | URI referencing information about the content language, if not expressable using BCP 47 | URI | "https://en.wikipedia.org/wiki/Early_New_High_German" |
| `preferred` | `<lex:Value>` is preferred among other `<lex:Value>` elements in the same `<lex:Field>` (`MAY` be used for UI hints) | Boolean | "true" |
| `ref` | An unspecified reference to external information about the content of the `<lex:Value>` element. This can be an audio file for `phonetic` field value or an external page with more information. | URI | "https://thesaurus-linguae-aegyptiae.de/sentence/ICEDBefDgQaukEMWmajL4HChr4I" |
| `idRefs` | Whitespace separated list of XML IDs to reference other field/values | String | "id_sense1 id_sense2" |
| `vocabRef` | Reference to the used vocabulary of the content of the `<lex:Value>` element | URI | "https://universaldependencies.org/u/pos" |
| `vocabValueRef` | Reference to a specific value of the used vocabulary | URI | "https://universaldependencies.org/u/pos/NOUN", "https://universaldependencies.org/u/feat/Gender#Fem" |
| `type` | A classification for a field value. It may function to group values and `MAY` be used for UI hints. | String | "hyphenation" (field type `segmentation`), "sample" (field type `ref`) |

The `<lex:Value>` element `MAY` provide the language of its content in IETF BCP 47 via the `@xml:lang` attribute. If more information is required, the `OPTIONAL` `@langUri` attribute `MAY` additionally be provided. This overrides the `<lex:Entry>` language information (if any).

The `@xml:id` and `@idRefs` attributes are used to link `<lex:Value>` elements inside the same `<lex:Entry>` element, which can be used to highlight these relations in the user interface (like a `citation` `<lex:Value>` element referencing its corresponding `definition` `<lex:Value>` element). The semantics of this reference is unspecified.

The `@type` attribute `SHOULD` be used to further specify generic field `<lex:Value>` by using values from limited vocabulary to classify their intended usage. Certain values are `RECOMMENDED` for different `<lex:Field>` types, but with the exception of the field `citation`, it is an open list. Known `@type` values `MAY` be used by clients to interpret and process `<lex:Value>` contents in specific ways.

The `@vocabRef` attribute refers to the general vocabulary from which the content of the `<lex:Field>` element originates, e.g. `https://universaldependencies.org/u/pos/` for referencing the part-of-speech (POS) tags of the Universal Dependencies project, which provides context for the

interpretation of a string like "NOUN". `@vocabValueRef` refers to a specific value of a vocabulary, like `https://universaldependencies.org/u/pos/NOUN` for a noun according to the POS tags of the Universal Dependencies project. If both are provided, they `MUST` refer to the same vocabulary.

### 3.3.2. Citation Value attributes

Values in `<lex:Field>` elements of type `citation` are allowed more optional attributes, which are listed in Table 5, “Lexical Search Value attributes for type 'citation'”

*Table 5. Lexical Search Value attributes for type 'citation'*

| Value Attribute | Description | Data Type | Examples (without quotes) |
|---|---|---|---|
| `source` | Name of the source from which the citation value was taken | String | "FCS Daily - Latest news and gossip stories" |
| `sourceRef` | Reference of the source from which the citation value was taken | URI | "https://wwww.fcs-daily.de/article/123" |
| `date` | Date information for the citation value | EDTF date/time (Level 0) | "1992-02-10", "2024", "2000-01-01T01:02:03Z" |

### 3.3.3. Value attributes by Field type

Table 6, “Attributes for Lexical Search Field types” lists which attributes `MUST` or `SHOULD` be attached to `<lex:Value>` elements of certain *Field Type Identifiers* to provide clients context for interpretation of values.

*Table 6. Attributes for Lexical Search Field types*

| Field Type Identifier | Attribute Usage Recommendations |
|---|---|
| `entryId` | The attributes `@vocabRef` or `@vocabValueRef` `MAY` be used to provide context. |
| `definition`, `etymology` | Hierarchical relations between `<lex:Value>` elements across `<lex:Field>` elements can be indicated by using the `@xml:id` and `@idRefs` attributes. Subordinate `<lex:Value>` elements contain an `@idRefs` attribute that refers to the `@xml:id` attribute of another `<lex:Value>` element. Clients `MAY` decide to show this hierarchy. |
| `pos`, `case`, `degree`, `gender`, `mood`, `number`, `sentiment`, `tense` | The `@vocabRef` or `@vocabValueRef` attributes `SHOULD` be used to allow disambiguation of the provided values. Clients `MAY` use them to help users, e.g. by providing translations or pointing to definitions. |

| Field Type Identifier | Attribute Usage Recommendations |
|---|---|
| `segmentation` | Values `SHOULD` use the pipe (`\|`) character as separator to allow for post-processing by clients (e.g., exchange of separators, splitting of parts). Other separator characters (e.g., `+`, `-`, `·`) `MAY` be used but uniform processing by clients can not be guaranteed.<br><br>The `@type` attribute `SHOULD` be used for `segmentation` `<lex:Value>` elements to indicate what type of segmentation is being performed or described. |
| `ref` | The `@type` attribute `MUST` be used to indicate what kind of reference is being provided. |
| `senseRef` | The attributes `@vocabRef` or `@vocabValueRef` `MUST` be used to provide context. |
| `citation` | The *special* attributes `@source`, `@sourceRef` and `@date` `SHOULD` be used to provide additional context for `citation` `<lex:Value>` elements.<br><br>The `@type` attribute `SHOULD` be used to indicate what type of citation, quotation or example is being given. If the `@type` attribute is used, it `MUST` contain one of the following values:<br><br>• `example`: content is a usage example, either created for this purpose or adapted from real-world text,<br>• `quotation`: content is a verbatim excerpt cited to illustrate usage or meaning,<br>• `generated`: content is a machine-generated, synthetic example for the use of the lemma. |

## 3.4. Serialisation

The serialisation of a LexFCS entry is specified by the corresponding Data Views.

# Chapter 4. LexCQL

LexCQL uses the Contextual Query Language (CQL) to query lexical resources available in the FCS. This has the benefit of using an existing, well-known and standardised query language with an established ecosystem, including libraries, parsers and extensive documentation. The proposed Context Set can be found in Section A.1, “CQL ContextSet specification”.

LexCQL queries are defined for flexibility, i.e. endpoints SHOULD be able to retrieve lexical records that include alternative spellings and normalisation variants (e.g., case variations, diacritics, and umlauts). Consequently, it SHOULD be straightforward to formulate meaningful queries, reduce frustration caused by missing or incomplete results, as well as perform fuzzy search. While endpoints SHOULD implement this flexible, user-oriented handling, they remain free to rank exact or highly relevant results higher in the response. To balance this flexibility, users SHOULD be provided with optional operators, relations and modifiers, enabling them to *refine* search queries for higher precision.

Endpoints MUST support term-only queries. Furthermore, endpoints and clients MUST support *Level 2* of the CQL Server Conformance. This ensures that endpoints can *parse* and clients can *serialize* all valid CQL expressions. To report errors and informational messages, LexCQL adopts the diagnostics codes defined in CLARIN-FCS Core 2 and OASIS SRU 2.0. Endpoints MUST utilize these standard diagnostics; a comprehensive list of valid error messages can be found in CLARIN-FCS Core 2, Section 3.1.

As in the Contextual Query Language (CQL), all parts of LexCQL are case insensitive apart from user supplied search terms and values for relation modifiers.

| | |
|---|---|
| **NOTE** | This does not *imply* that endpoints are REQUIRED to support all of CQL (and LexCQL), but rather that they are able to *parse* all of CQL and generate the appropriate error message, e.g. if a query includes a feature they do not support. |

## 4.1. Search Term

A search term MAY be enclosed in double quotation marks ("), though it need not be. However, it MUST be enclosed if it contains any of the following characters:

- <, > (left or right angle bracket)
- (, ) (left or right parenthesis)
- = (equals sign)
- / (forward slash)
- " (double quotation mark)
- whitespace characters

A backslash (\) MUST be used to escape literal double quotation marks (") and backslashes (\) within a quoted term.

### Examples

- `car`
- `"car"`
- `"car wash"`
- `car's`
- `"27\""` to search for the literal term 27"
- `"\\"` to search for the literal term \

## 4.2. Queryable Fields (Indexes)

LexCQL supports querying all lexical fields (in CQL called "indexes") that are supported by the LexFCS Data Model using their respective field types. Endpoints `MUST` support queries on the `lemma` field, which serves as the default field if no explicit field is specified in the query. Endpoints `SHOULD` support as many of the remaining field types as possible.

To support queries on all available fields, there is an additional virtual field `any`. Queries on this field execute with the same semantics as an `OR` conjunction across all supported fields within a lexical entry.

To support queries related to the language of a lexical record, an additional virtual field `lang` is defined. It refers to the primary language of the entire lexical entry. It shares the same semantics as the `@xml:lang` attribute of a `<lex:Field>` in a lexical `<lex:Entry>`, where it is queried with the `lang` relation modifier.

See Section 4.3, “Relations” for more information on allowed relations and their behaviour.

| NOTE | The query examples in this chapter are intended to illustrate the structural feature described in each case. Query examples with a full semantic description can be found in the appendix (Section B.2.1.2, “Query Examples”). |
|---|---|

### Examples

- `car`

  Implicit use of the default `lemma` field.

- `lemma = "car"`

  Explicit use of the `lemma` field.

- `pos = "NOUN" AND synonym = "house"`

  Combined search on different fields.

- `lang = "de" AND translation =/lang=en "parliament"`

Virtual field `lang` of the lexical entry and relation modifier `lang` for values of field `translation`.

- `any = "bike"`

  Virtual field `any` for searching on all fields within a lexical entry.

# 4.3. Relations

LexCQL supports the following relations between a query field and a search term, some of which `MUST` be supported by every endpoint while others are optional. All relations check whether a given search term matches the corresponding field value, based on one of the following assumptions regarding the structure of the field value:

- Tokenised: search term and field value are both split into sets of word tokens.
- Untokenised: search term and field value are both sequences of characters.
- Entity reference: the search term will be evaluated as an Uniform Resource Identifier (URI) of an externally defined entity.

| NOTE | The specific method of tokenisation is not specified and depends on the implementation used at the endpoint. |
|---|---|

## Mandatory

- `=` (Equality)

  The search term is tokenised by the endpoint and it is evaluated whether all tokens are present in the tokenised field value while ignoring the given token order. For instance, a query for `Wege` matches "Alle Wege führen nach Rom", as the token "Wege" occurs within it. Likewise, a multi-word query such as `"Rom führen"` matches the same phrase because both tokens are present. Endpoints `MAY` take measures to enhance recall, e.g., lemmatisation, case folding, or diacritic normalisation. For example, if an endpoint implements lemmatisation, a query for `"mouse green"` `MAY` match inflected forms like "green mice".

  The `regexp` relational modifier and the default special characters are evaluated only with respect to individual tokens.

  Endpoints `MAY` fall back to an untokenised matching if tokenisation is not supported.

  | NOTE | The `=` relation is the default relation for term-only queries, so the query `car` is identical to `lemma = car`. |
  |---|---|

- `==` (Exact Equality)

  This relation is used for exact matching against the complete untokenised value of a field. A query for `lemma == Wege` `MUST NOT` match the string "Alle Wege führen nach Rom", but `MUST` only match records where the entire untokenised field value is identical to "Wege".

By default, endpoints MUST NOT perform any recall enhancement, e.g., lemmatisation, case folding, or diacritic normalisation, to ensure exact matching. However, relation modifiers (ignoreCase, ignoreAccents) MAY be used to override this behaviour.

The regexp relation modifier will match against any substring of the untokenised field value. Regular expression anchoring (^, $) can be used to change this behaviour.

## Optional

- <> (Exact Inequality)

  This relation evaluates exact inequality. It matches all records where the untokenised field value is not identical to the query string. For example, lemma <> "car" matches any record whose untokenised value is not exactly "car". It is the only inequality relation supported in LexCQL.

- is (Identity)

  This relation is used to search for results based on entities or concepts that are defined in external vocabularies using Uniform Resource Identifiers (URIs). It thus *resolves* term values to entities rather than literal strings.

  Most grammar fields use Universal Dependencies (UD) vocabularies as default. In these cases, the corresponding namespace can be omitted. As an example, the query case is Acc uses the default namespace https://universaldependencies.org/u/feat/Case and is equivalent to the query case is "https://universaldependencies.org/u/feat/Case#Acc". All field types with a default vocabulary are given in Table 7, “Field Types and Universal Dependencies Vocabularies”.

  Endpoints MUST be able to support full URIs or resolve namespace prefixes to their canonical form for matching. Endpoints MAY support other namespace prefixes.

  Entity references are considered to be using the default namespace if they do NOT contain a colon (:) character. Otherwise, they are either full URIs or using namespace prefixes that MAY need resolving for matching.

*Table 7. Field Types and Universal Dependencies Vocabularies*

| Field Type | Universal Dependencies Vocabulary | Example Query |
|---|---|---|
| case | https://universaldependencies.org/u/feat/Case | case is Acc |
| degree | https://universaldependencies.org/u/feat/Degree | degree is Sup |
| gender | https://universaldependencies.org/u/feat/Gender | gender is Fem |
| mood | https://universaldependencies.org/u/feat/Mood | mood is Imp |
| number | https://universaldependencies.org/u/feat/Number | number is Sing |
| pos | https://universaldependencies.org/u/pos/ | pos is NOUN |
| sentiment | https://universaldependencies.org/u/feat/Polarity | sentiment is Neg |

| Field Type | Universal Dependencies Vocabulary | Example Query |
|---|---|---|
| tense | https://universaldependencies.org/u/feat/Tense | `tense is Fut` |

> **NOTE** The `.html` extension for Universal Dependencies URIs should be considered `OPTIONAL` and both forms are equivalent for the purpose of matching records.

### Examples

- `lemma = "car"`

  Matches lexical records where the lemma contains the token "car". Endpoints `MAY` determine their own balance between precision and recall (e.g., applying case-insensitivity or unicode normalisation).

- `definition = "person"`

  Matches lexical records where the definition contains the token "person" (e.g., matching a definition like "describes a person who...").

- `lemma == "car"`

  Matches lexical records whose lemma exactly matches the string "car", including canonically equivalent characters.

- `lemma <> "car"`

  Matches lexical records whose lemma is not exactly the string "car".

- `pos is "https://universaldependencies.org/u/pos/NOUN"`

  Matches lexical records whose part-of-speech corresponds to a noun within the Universal Dependencies part-of-speech tagset.

- `pos is NOUN`

  Matches lexical records whose part-of-speech corresponds to a noun within the Universal Dependencies part-of-speech tagset. Search term is shortend by using the default namespace.

- `pos is wd:Q1166153`

  Matches lexical records whose part-of-speech corresponds to an intransitive verb according to Wikidata using the (non-default) namespace "http://www.wikidata.org/entity/" with the namespace prefix "wd".

## 4.4. Relation Modifiers

Relations `MAY` be modified using relation modifiers, each separated by the slash character (`/`). Each individual endpoint decides whether and to what extent relation modifiers are supported. The

following relation modifiers are valid:

- (default)

  By default, special characters within search terms are interpreted as wildcards.

  - A single asterisk (*) masks zero or more characters.
  - A single question mark (?) masks exactly one character, thus N consecutive question marks mask N characters.
  - A backslash (\) acts as an escape character for *, ?, the double quote ("), and the backslash character itself. A backslash MUST be followed immediately by one of these characters; otherwise the query is invalid.

- `lang`

  Specifies the language of the search term. A match SHOULD only occur with lexical field values that contain an explict language specification matching this language, or if the default language of `<lex:Entry>` matches. It is RECOMMENDED to use and support an IETF BCP 47 compliant language code.

- `ignoreCase`, `respectCase`

  Instructs the endpoint to either ignore or respect string casing during matching, overriding the endpoint's default casing behavior.

- `ignoreAccents`, `respectAccents`

  Instructs the endpoint to either ignore or respect diacritics and character variants, overriding the endpoint's default behavior (where `respectAccents` is RECOMMENDED as the default). This modifier controls the handling of Unicode combining characters, historical letter forms (e.g., ſ U+017F), ligatures (e.g., ﬁ U+FB01), or multi-character equivalents (e.g., ß U+00DF). When `ignoreAccents` is active, endpoints SHOULD apply Unicode compatibility normalisation (NFKD) and strip combining marks (e.g., ﬁ → fi), fold historical letter forms to their modern equivalents (e.g., ſ → s), and apply case-folding rules for characters such as ß → ss. When `respectAccents` is active, endpoints MUST preserve the full distinction of characters as encoded, including diacritics, combining marks, and compatibility-equivalent or historical letter forms. No accent stripping, compatibility folding, or character-level normalisation (beyond standard Unicode normalisation for comparison) are applied for matching purposes.

- `regexp`

  Interprets the search term as a POSIX regular expression. Support for extended regular expression features is OPTIONAL and endpoint-specific. By default, regular expressions are unanchored, meaning they match against any substring of the index value unless the expression explicitly uses anchoring constructs such as `^` (start of string) or `$` (end of string). This overrides the interpretation of the default special characters mentioned above.

| | |
|---|---|
| **WARNING** | The OASIS CQL Context Set defines the modifiers `word` and `string` for controlling tokenisation during matching (`string` meaning no tokenisation is |

> applied). LexCQL does not implement these modifiers; they are ignored if present.

### Examples

- `lemma = "car s*"`

  Matches records where the lemma contains the token "car" and any token starting with "s" (e.g., "safety car", "car service").

- `lemma = "car s\*"`

  Matches records where the lemma contains the literal tokens "car" and "s*".

- `lemma =/lang=fr "chat"`

  Given search term in French.

- `lemma =/ignoreCase "Handy"`

  Given search term to be evaluated regardless of case (incl. "Handy", "handy", "haNDY").

- `lemma =/ignoreAccents "café"`

  Given search term to be evaluated as "café", "cafe" or other accent-normalised variants.

- `definition =/regexp "^cat"`

  Given search term is a regular expression describing any value beginning with "cat" using an anchored regular expression.

- `synonym =/lang=en/ignoreCase "handy"`

  Given search term in English regardless of case.

## 4.5. Operators

LexCQL supports the following Boolean operators to construct complex queries. By default, Boolean operators share the same precedence and are evaluated from *left to right*. Parentheses `MAY` be used to change the evaluation order and group expressions explicitly.

- `AND`

  Represents the intersection of two result sets. A record `MUST` match both connected search clauses to be included in the final results.

- `OR`

  Represents the union of two result sets. A record `MUST` be included in the final results if it matches either of the connected search clauses.

## Examples

- `lemma = "play" AND pos = "VERB"`

  Combination of two search clauses with the operator "AND".

- `lemma = "light" AND (pos = "ADJ" OR pos = "NOUN")`

  Using parentheses to change evaluation order of a complex query.

- `lemma = "play" AND pos <> "NOUN"`

  Combination of search clauses with two different relations.

# Chapter 5. LexFCS Data Views

Data formats for the representation of results.

## 5.1. Lexical Data View

The *Lexical Data View* (Lex Data View) is the mandatory serialisation of search results for *Lexical Search* queries. It structures information into key and multiple values pairs. More details in Chapter 3, *LexFCS Data Model*.

| | |
|---|---|
| **Description** | The representation of a lexical resource |
| **MIME type** | `application/x-clarin-fcs-lex+xml` |
| **Payload Disposition** | *inline* |
| **Payload Delivery** | *send-by-default* (`REQUIRED`) |
| **Recommended Short Identifier** | `lex` (`RECOMMENDED`) |
| **XML Schema** | DataView-Lex.xsd |

The Lexical Data View is serialised as XML in the `<fcs:DataView>` element, specified in the FCS Core 2 specification (section "Result Format", §2.2.3). The elements `<lex:Entry>`, `<lex:Field>` and `<lex:Value>` with their attributes are the direct serialisation of the Data Model.

A minimal example can be seen at Example of basic Lexical Data View. A few more targeted examples can be found in Section B.1.1, “Serialisation with the Lexical Data View” demonstrating edge cases and special features when using this Data View.

*Example of basic Lexical Data View*

```
<!-- potential @pid and @ref attributes omitted -->
<fcs:DataView type="application/x-clarin-fcs-lex+xml"
xmlns:fcs="http://clarin.eu/fcs/resource">
  <lex:Entry xmlns:lex="http://clarin.eu/fcs/dataview/lex">
    <lex:Field type="lemma">
      <lex:Value xml:lang="de">Becher</lex:Value>
    </lex:Field>
    <!-- ⋯ -->
  </lex:Entry>
</fcs:DataView>
```

Complete serialisation examples in the *Lexical Data View* are provided along with this specification.

## 5.2. Extension of the Hits Data View for LexFCS

The *Generic Hits (HITS)* Data View is mandatory in FCS Core 2 specification (section "Basic Search",

§2.2.3.2). This specification extends the `<hits:Hit>` element with an optional `@kind` attribute (see Section 5.2.1, "Using the @kind Attribute"), which provides information on the content of the Hit result.

| | |
|---|---|
| **Description** | The representation of the hit |
| **MIME type** | `application/x-clarin-fcs-hits+xml` |
| **Payload Disposition** | *inline* |
| **Payload Delivery** | *send-by-default* (`REQUIRED`) |
| **Recommended Short Identifier** | `hits` (`RECOMMENDED`) |
| **XML Schema** | DataView-LexHits.xsd, based on "DataView-Hits.xsd" |

*Example of basic* **Hits** *Data View*

```
<!-- potential @pid and @ref attributes omitted -->
<fcs:DataView type="application/x-clarin-fcs-hits+xml"
xmlns:fcs="http://clarin.eu/fcs/resource">
  <hits:Result xmlns:hits="http://clarin.eu/fcs/dataview/hits">The quick brown
<hits:Hit>fox</hits:Hit> jumps over the lazy<hits:Hit>dog</hits:Hit>.</hits:Result>
</fcs:DataView>
```

### 5.2.1. Using the @kind Attribute

To extend the Hits Data View, the `<hits:Hit>` element is reused as per the FCS Core 2 Specification, but with the additional optional `@kind` attribute, which provides content hinting. The values of the `@kind` attribute follow the scheme `lex-<FIELD-TYPE>`, e.g. `lex-lemma` for lemma, `lex-pos` for part of speech (POS) tags etc. Textual content outside of `<hits:Hit>` is displayed unchanged.

*Example of extended Hits Data View with additional* `@kind` *attributes*

```
<fcs:DataView type="application/x-textplus-fcs-hits+xml"
xmlns:fcs="http://clarin.eu/fcs/resource">
  <hits:Result xmlns:hits="http://clarin.eu/fcs/dataview/hits"><hits:Hit kind="lex-
lemma">Apple</hits:Hit>: <hits:Hit kind="lex-pos">NOUN</hits:Hit>. <hits:Hit
kind="lex-definition">An apple is an edible fruit produced by an apple
tree.</hits:Hit></hits:Result>
</fcs:DataView>
```

Endpoints `MUST` generate responses that are valid according to the XML schema "DataView-LexHits.xsd".

# A. Normative Appendix

## A.1. CQL ContextSet specification

Used identifier: `http://text-plus.org/cql/lex/1.0/` (*draft*)

Recommended prefix: `lex`

> **TIP** For more examples of CQL ContextSets, see the list of Context Sets at the Library of Congress (LoC).

### A.1.1. Indexes

> **TIP** For more information about CQL indexes, see The CQL Context Set, section "INDEXES".

| Index Title | Description |
|---|---|
| `antonym`, `holonym`, `hypernym`, `hyponym`, `meronym`, `synonym` | Semantic relations |
| `any` | All available indexes (virtual index) |
| `baseform` | Baseform (stem, root, ...) of a lemma |
| `case`, `degree`, `gender`, `mood`, `number`, `tense` | Morphological information, e.g. in full-form dictionaries |
| `citation` | A citation, quotation or usage example of this entry's lemma. |
| `definition` | Definition or description |
| `entryId` | Identifier of the current entry |
| `etymology` | Etymology information |
| `grammar` | General grammatical information, only to be used if more specific fields can not be supported |
| `frequency` | Frequency information like occurrences, relative frequency, word rank or frequency class |
| `lang` | Language of the lexical entry's content |
| `lemma` | Lemma form |
| `phonetic` | Phonetic form |
| `pos` | Part of Speech |
| `ref` | A URI referencing a related resource |
| `related` | Unspecified (semantic) relation |
| `segmentation` | Composita segmentation, hyphenation; into phonemes, syllables, subwords |

| Index Title | Description |
|---|---|
| `senseRef` | ID of a sense definition |
| `sentiment` | Sentiment information |
| `transcription` | Transcription or transliteration |
| `translation` | Translation |

### A.1.2. Relations

**TIP** More information about CQL relations can be found The CQL Context Set, section "RELATIONS".

#### A.1.2.1. Implicit Relations

- `=`

  The search term is tokenised by the endpoint and it is evaluated whether all tokens are present in the tokenised field value while ignoring the given token order. Endpoints `MAY` fall back to an untokenised matching if tokenisation is not supported.

  Endpoints `MAY` take measures to enhance recall, e.g., lemmatisation, case folding, or diacritic normalisation.

- `==`

  This relation is used for exact matching against the complete untokenised value of a field.

- `<>`

  This relation evaluates exact inequality. It matches all records where the untokenised field value is not identical to the query string.

#### A.1.2.2. Defined Relations

- `is`

  This relation is used to search for results based on entities or concepts that are defined in external vocabularies using Uniform Resource Identifiers (URIs).

### A.1.3. Booleans

**TIP** For more information about Booleans in CQL, see The CQL Context Set, section "BOOLEANS".

- `AND`
- `OR`

## A.1.4. Relation Modifiers / Relation Qualifiers

**TIP** For more information about relation modifiers/qualifiers, see The CQL Context Set, section "RELATION MODIFIERS".

### A.1.4.1. Functional Modifiers

- `lang=value`

  Specifies the language of the search term. It is `RECOMMENDED` to use and support an IETF BCP 47 compliant language code.

- `ignoreCase`, `respectCase`

  Instructs the endpoint to either ignore or respect string casing during matching, overriding the endpoint's default casing behavior (which is undefined).

- `ignoreAccents`, `respectAccents`

  Instructs the endpoint to either ignore or respect diacritics and character variants, overriding the endpoint's default behavior (where `respectAccents` is `RECOMMENDED` as the default). This modifier controls the handling of Unicode combining characters, historical letter forms (e.g., ſ U+017F), ligatures (e.g., ﬁ U+FB01), or multi-character equivalents (e.g., ß U+00DF). When `ignoreAccents` is active, endpoints `SHOULD` apply Unicode compatibility normalisation (NFKD) and strip combining marks (e.g., ﬁ → fi), fold historical letter forms to their modern equivalents (e.g., ſ → s) and apply case-folding rules for characters such as ß → ss. When `respectAccents` is active, endpoints `MUST` preserve the full distinction of characters as encoded, including diacritics, combining marks, and compatibility-equivalent or historical letter forms. No accent stripping, compatibility folding, or character-level normalisation (beyond standard Unicode normalisation for comparison) are applied for matching purposes.

### A.1.4.2. Matching

- (default)

  By default, special characters within search terms are interpreted as wildcards.

  - A single asterisk (`*`) masks zero or more characters.
  - A single question mark (`?`) masks exactly one character, thus N consecutive question marks mask N characters.
  - A backslash (`\`) acts as an escape character for `*`, `?`, the double quote (`"`), and the backslash character itself. A backslash `MUST` be followed immediately by one of these characters; otherwise the query is invalid.

- `regexp`

  Interprets the search term as a POSIX regular expression. Support for extended regular expression features is `OPTIONAL` and endpoint-specific. By default, regular expressions are unanchored, meaning they match against any substring of the index value unless the

expression explicitly uses anchoring constructs such as `^` (start of string) or `$` (end of string). This overrides the interpretation of the default special characters mentioned above.

### A.1.5. Boolean Modifiers

**none**

### A.1.6. Examples

1. `cat`
   `"cat"`
   `"United Nations"`
   Different variants to search in the default index (i.e. `lemma`).
2. `lemma == mouse`
   Search for exact string value `mouse` in `lemma` index.
3. `pos = ADJ`
   Search for token "ADJ" in index `pos`.
4. `pos is "https://universaldependencies.org/u/pos/NOUN" AND definition = person`
   Search for nouns with token "person" in definition, using a tagset from the Universal Dependency Project for index `pos`.
5. `pos is NOUN AND definition = person`
   Search for nouns with token "person" in definition, using (implictly) a tagset from the Universal Dependency Project for index `pos`.
6. `definition = "cat"`
   Search for records whose definition contains the token "cat".
7. `pos = "NOUN" AND lemma <> "lion" AND definition = carnivore`
   Search for nouns with token "carnivore" in index `definition`, excluding records having the exact lemma "lion".
8. `pos = NOUN AND ( lemma = Apfel OR lemma = "Birne" )`
   Search for nouns with lemma containing either token "Apfel" or token "Birne" (or both).
9. `translation =/lang=en car`
   Search for records with an English translation that contains the token "car".
10. `lemma =/lang=fr "chat"`
    Search for records with a lemma in French that contains the token "chat".
11. `lemma =/ignoreCase "drive"`
    Search for records containing token "drive" regardless of case.
12. `lemma =/ignoreAccents "für"`
    Search for records with a lemma containing token "für" while ignoring diacritics.
13. `definition =/regexp "^cat.*"`
    Search for records whose definition contains a token beginning with "cat" using a regular expression.
14. `lemma == "\^ca\?r*"`

Search for records with the literal term "^ca?r*" as lemma.

# B. Non-Normative Appendix

## B.1. Best Practices

### B.1.1. Serialisation with the Lexical Data View

#### B.1.1.1. Specifying a default language – @xml:lang and @langUri

Each `<lex:Value>` element can specify language information using the `@xml:lang` and `@langUri` attributes. To avoid redundancy, a default language `MAY` be specified at the `<lex:Entry>` element. If `<lex:Value>` elements do not specify their language explicitely, they inherit the entry's language information.

`<lex:Value>` elements with the same `@xml:lang` attribute value but different `@langUri` attribute values `MUST` be considered to describe different languages. If in doubt, specify the language attributes redundantly.

*Default and explicit language specification for Values*

```
<!-- abbreviated example from examples/dwee-Becher.lex.xml -->
<lex:Entry xmlns:lex="http://clarin.eu/fcs/dataview/lex" xml:lang="de"> ①
  <lex:Field type="lemma">
    <lex:Value xml:lang="de">Becher</lex:Value>
  </lex:Field>
  <lex:Field type="segmentation">
    <lex:Value type="hyphenation">Be|cher</lex:Value> ②
  </lex:Field>
  <lex:Field type="hypernym">
    <lex:Value xml:lang="de">Gefäß</lex:Value> ③
    <lex:Value>Gegenstand</lex:Value>
    <lex:Value xml:lang="goh">pehhari</lex:Value>
  </lex:Field>
  <lex:Field type="synonym">
    <lex:Value>Eimer</lex:Value>
    <lex:Value xml:lang="de"
  langUri="https://en.wikipedia.org/wiki/Early_New_High_German">Kraus</lex:Value>  ④
    <lex:Value xml:lang="goh">kelich</lex:Value>
  </lex:Field>
</lex:Entry>
```

① Specifying German as default language of `<lex:Entry>` via `@xml:lang`.

② Any `<lex:Value>` element without its own language attributes inherits language information from its parent `<lex:Entry>`, here `@xml:lang="de"`.

③ `@xml:lang="de"` is redundant.

④ Due to `@langUri`, the content language of this `<lex:Value>` element `MUST` be considered to be a different language, even though the `@xml:lang` attribute value is the same as the one at

`<lex:Entry>` level. As no language information is inherited here, both attributes `@xml:lang` and `@langUri` MUST be specified.

#### B.1.1.2. Contextualise Value contents for improved interpretability

The example Contextualisation of senseRef Values shows how plain text content can be semantically enriched by explicitly stating the used vocabulary or by referencing the value's external definition via attributes `@vocabRef` or `@vocabValueRef`.

*Contextualisation of senseRef Values*

```
<!-- abbreviated example from examples/wortschatz-Auto.lex.xml -->
<lex:Entry xmlns:lex="http://clarin.eu/fcs/dataview/lex" xml:lang="de">
  <lex:Field type="lemma">
    <lex:Value xml:lang="de">Auto</lex:Value>
  </lex:Field>
  <lex:Field type="senseRef">
    <!-- Dornseiff -->
    <lex:Value vocabRef="https://doi.org/10.1515/9783110457742">8.10</lex:Value> ①
    <!-- GermaNet -->
    <lex:Value vocabRef="http://textplus.sfs.uni-tuebingen.de/api/germanet/synset"
>s123456789</lex:Value>
    <!-- Gemeinsame Normdatei (GND) -->
    <lex:Value vocabValueRef="http://d-nb.info/gnd/4129315-0">4129315-0</lex:Value>
 ②
    <!-- Princeton WordNet -->
    <lex:Value vocabRef="http://wordnet-rdf.princeton.edu/ontology#Synset"
>02961779-n</lex:Value> ③
  </lex:Field>
</lex:Entry>
```

① Using the `@vocabRef` attribute referencing `https://doi.org/10.1515/9783110457742`, the value `8.10` can be interpreted as category *"8.10 Auto, Fahrt"* of the Dornseiff dictionary, volume 9.

② The `@vocabValueRef` attribute value contains the authoritative reference to the authority file GND for the value "4129315-0".

③ Using the `@vocabRef` attribute, the value "02961779-n" can be interpreted in the context of Princeton WordNet synsets.

#### B.1.1.3. Connecting Values using @xml:id and @idRefs

Any `<lex:Value>` element can refer to other `<lex:Value>` elements in the same `<lex:Entry>` element by specifying their IDs – stated in their `@xml:id` attribute – in a whitespace separated list in the `@idRefs` attribute. The type of this relation is unspecified.

*Value relations using @xml:id and @idRefs*

```
<!-- abbreviated example from examples/GermaNet_Ei.lex.xml -->
<lex:Entry xmlns:lex="http://clarin.eu/fcs/dataview/lex" xml:lang="de">
```

```
  <lex:Field type="lemma">
    <lex:Value xml:lang="de">Ei</lex:Value>
  </lex:Field>
  <lex:Field type="senseRef">
    <lex:Value xml:id="sense_1" vocabRef="http://textplus.sfs.uni-
  tuebingen.de/api/germanet/synset">s39427</lex:Value> ①
    <lex:Value xml:id="sense_2" vocabRef="http://textplus.sfs.uni-
  tuebingen.de/api/germanet/synset">s25806</lex:Value>
    <lex:Value xml:id="sense_3" vocabRef="http://textplus.sfs.uni-
  tuebingen.de/api/germanet/synset">s25813</lex:Value>
  </lex:Field>
  <lex:Field type="definition">
    <lex:Value idRefs="sense_1">unzählbar, ohne Plural: […]</lex:Value> ②
    <lex:Value idRefs="sense_2">ein Schalengebilde, in dem […]</lex:Value>
    <lex:Value idRefs="sense_3">eine Keimzelle</lex:Value>
  </lex:Field>
  <lex:Field type="hypernym">
    <lex:Value idRefs="sense_1">festes Nahrungsmittel</lex:Value> ②
    <lex:Value idRefs="sense_2">Keim</lex:Value>
    <lex:Value idRefs="sense_2">Keimling</lex:Value>
    <lex:Value idRefs="sense_3">Gamet</lex:Value>
  </lex:Field>
  <lex:Field type="hyponym">
    <lex:Value idRefs="sense_2">Brutei</lex:Value>
    <lex:Value idRefs="sense_2 sense_3">Windei</lex:Value> ③
  </lex:Field>
</lex:Entry>
```

① `<lex:Value>` element with XML ID `sense_1`.

② `<lex:Value>` elements referring to the `<lex:Value>` element with `@xml:id="sense_1"` attribute.

③ A `<lex:Value>` elements refering to multiple XML IDs.

#### B.1.1.4. Connecting Values within Fields using @xml:id and @idRefs to build Value hierarchies

`<lex:Value>` elements within `<lex:Field>` elements of type `definition`, `etymology` and `senseRef` can be organised hierarchically by using the `@xml:id` and `@idRefs` attributes. `<lex:Value>` elements can refer to their head `<lex:Value>` element via their `@idRefs` attribute.

If `@idRefs` contains multiple IDs then only the first one is assumed to be the direct parent `<lex:Value>` element, so additional IDs `MAY` still connect the `<lex:Value>` element to other `<lex:Value>` elements as described in Section B.1.1.3, “Connecting Values using @xml:id and @idRefs”.

*Value hierarchy using @xml:id and @idRefs*

```
<!-- abbreviated example from examples/dwdswb-herausgehen.lex.xml -->
<lex:Entry xmlns="http://clarin.eu/fcs/dataview/lex" xml:lang="de">
  <lex:Field type="lemma">
    <lex:Value xml:lang="de">herausgehen</lex:Value>
  </lex:Field>
```

```
  <lex:Field type="definition">
    <lex:Value xml:id="def-1">(zu Fuß) (von irgendwo, drinnen) heraus bzw. nach draußen, […]</lex:Value> ①
    <lex:Value xml:id="def-1-1" idRefs="def-1">etw. (eine Situation, einen Zustand) hinter […]</lex:Value> ②
    <lex:Value xml:id="def-2">sich (aus einem Objekt) herauslösen oder entfernen lassen</lex:Value>
    <lex:Value xml:id="def-3">sich ableiten, herausfiltern, schlussfolgern lassen</lex:Value>
    <lex:Value xml:id="def-5">besonders von postalischen oder elektronischen Sendungen; […]</lex:Value>
    <lex:Value xml:id="def-5-1" idRefs="def-5">von Informationen; gestreut , verbreitet […]</lex:Value>
  </lex:Field>
  <lex:Field type="citation">
    <lex:Value idRefs="def-1">ich sah sie aus dem Garten herausgehen</lex:Value>
    <lex:Value idRefs="def-1" source="Die Welt, 29.02.2020">Nach etwa 350 Metern […]</lex:Value> ③
    <lex:Value idRefs="def-1-1" source="Neue Westfälische, 08.05.2023">Die Kliniken […]</lex:Value>
  </lex:Field>
</lex:Entry>
```

① A `<lex:Value>` element with an `@xml:id` attribute of value "def-1".

② A `<lex:Value>` element stating in its `@idRefs` attribute to be a child of the `<lex:Value>` element with `@xml:id` attribute of value "def-1".

③ `<lex:Value>` elements in `<lex:Field>` elements of type other than `definition` or `etymology` can refer to the same `<lex:Value>` element but will not be seen as children of this hierarchy, only as related in an unspecifed way.

#### B.1.1.5. Referencing external resources using the @ref attribute

The `@ref` attribute is used to reference an external resource that provides additional information about the `<lex:Value>` element's content. Depending on the `<lex:Field>` type, the referenced resource `MAY` be interpreted differently.

For example in `<lex:Value>` elements in `<lex:Field>` element with `@type="phonetic"`, the external resource might provide a link to an audio file with pronunciation.

*Audio integration for phonetic Values*

```
<!-- abbreviated example from examples/dwdswb-herausgehen.lex.xml -->
<lex:Entry xmlns="http://clarin.eu/fcs/dataview/lex" xml:lang="de">
  <lex:Field type="lemma">
    <lex:Value xml:lang="de">herausgehen</lex:Value>
  </lex:Field>
  <lex:Field type="phonetic">
    <lex:Value ref="https://www.dwds.de/audio/032/herausgehen.mp3">hɛˈʀaʊsgeːə
n</lex:Value>
```

```
  </lex:Field>
</lex:Entry>
```

# B.2. Further Explanations

## B.2.1. LexCQL

### B.2.1.1. Relations and their Modifiers

This section provides further explanations and example-oriented illustrations of the relations `=` (Section B.2.1.1.1) and `==` (Section B.2.1.1.2) defined in Section 4.3, “Relations”, as well as their changing behavior with different modifiers (e.g. Section B.2.1.1.6) applied.

Although there are term-only queries, the illustrations in this chapter always show only the terms (red) to demonstrate matching against the field values (green), and not complete queries, as this would only be valid for the `=` relation, but not for the `==` relation.

#### Untokenised and Tokenised Form

To understand the function of the relations and their modifiers, it is important to understand the difference between untokenised and tokenised forms, which apply to the content of a search term or field value. Figure 1 illustrates the difference between untokenised and tokenised form of terms and fields. The example terms and field values are then used in the following sections to illustrate the concrete matching mechanisms of search terms and field values according to the used relations and relation modifiers.

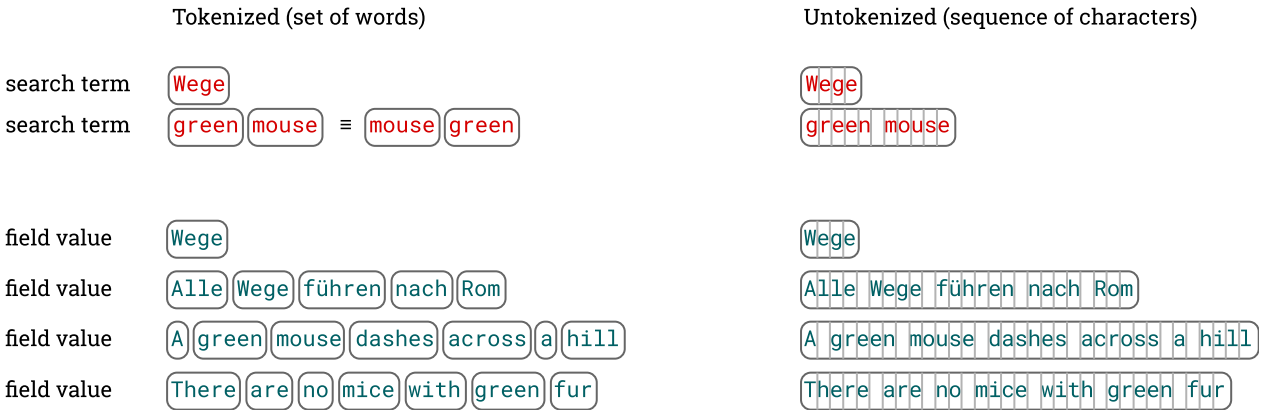


*Figure 1. Terms and fields in untokenised and tokenised form*

The tokenised form of a term or field value is the result of a tokenisation process, which splits the content into smaller units (tokens) according to certain rules. The untokenised form is the original content without any splitting. As result, the tokenised form may contain multiple tokens, either in terms or in field values (e.g., "green mouse" is tokenised into "green" and "mouse") and results in a set of tokens, while the untokenised form is a fixed sequence of characters. The matching of search terms and field values can be performed on either the untokenised or tokenised form, depending on the specified relation and its modifiers.

**= (equality)**

Using the = relation, the search term, split into tokens, is matched against the field value in its tokenised form. The matching is performed on a token-by-token basis, and the search term matches the field value if all tokens of the search term are found in the field value, regardless of their order or position.

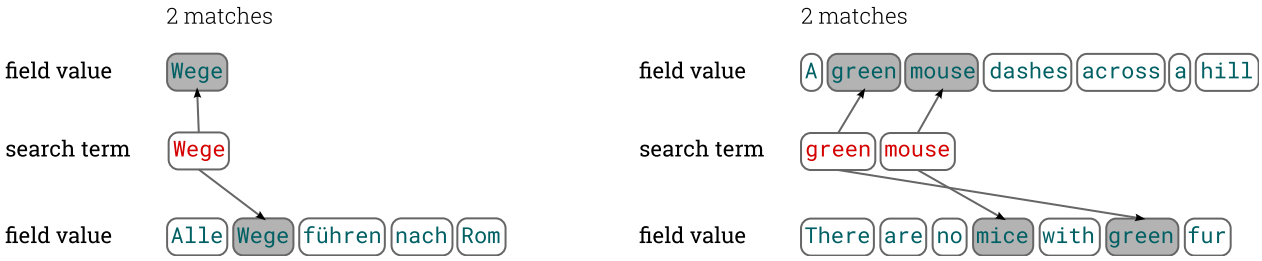


*Figure 2. The = relation uses tokenised form for matching.*

Another noticeable aspect of the = relation is that it allows for normalisation of the search term and field value, such as case-insensitivity or base form matching, depending on the server's implementation. This means that the search term "mouse" may match a field value "Mouse" or "mice".

The = relation is the default relation for term-only queries (i.e., queries without an explicit relation, e.g., `green mouse`), which are translated to a tokenised lookup in the field *lemma* (i.e., `lemma = "green mouse"`). It provides a flexible matching mechanism that can accommodate variations in spelling, capitalisation, and other factors and therefore is recall-oriented.

**== (exact equality)**

Using the == relation, the search term is matched against the field value in its untokenised form. The matching is performed on the entire string, and the search term matches the field value only if they are exactly equal, including order and position of characters. That's why in Figure 3 the only match is on field value "Wege". All other matches would be partial matches, which are not supported.

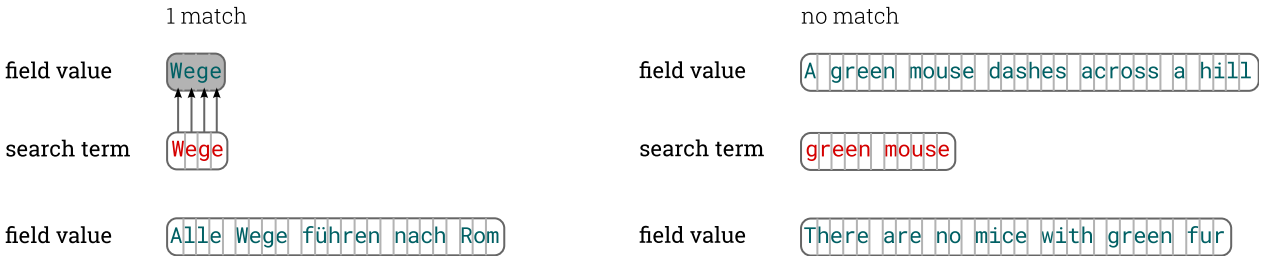


*Figure 3. The == relation uses untokenised form for matching.*

**==/regexp (exact equality with regular expression modifier)**

Using the == relation with the regular expression modifier (regexp), the search term is matched against the field value in its untokenised form, but the search term is interpreted as a regular expression. This allows for more flexible matching patterns, such as prefixes, suffixes, or complex character sequences.

It is important to understand that simply using this relation modifier without any Regex-specific

control characters changes the matching behaviour in such a way that the == relation no longer has to match the entire field value, but now allows partial matches. This is because Regex itself specifies mechanisms for explicitly querying *at the start* (^) or *at the end* ($), which are intended to be supported in order to realise the full potential of exact full matches and partial matches.

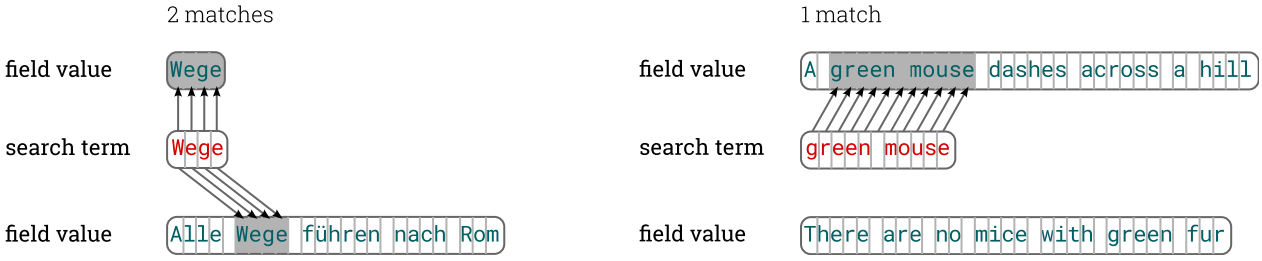


*Figure 4. The == relation with regular regexp modifier allows for partial matches.*

**= (equality with special characters)**

Using special characters (Section 4.4, "Relation Modifiers") in the search term with the = relation, the search term is interpreted as a pattern that can match multiple field values, depending on the specific special characters used. The matching is performed on the tokenised form of the field value, and the search term matches if it satisfies the pattern on at least one token of the field value.

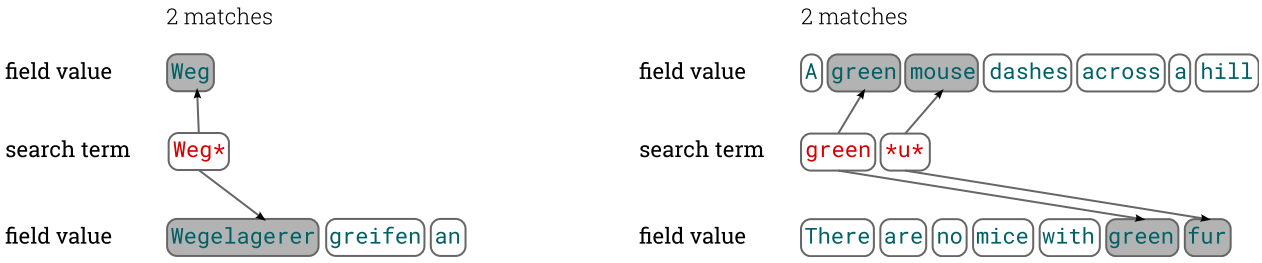


*Figure 5. The = relation with special character * allows for flexible matching patterns.*

As seen in Figure 5 above, special characters can significantly alter the matching behaviour of the = relation. To demonstrate that better, we introduce another German example field value "Wegelagerer greifen an" (English: "Highwaymen attack") and show how the search term Weg* matches it, because it looks up tokens starting with "Weg" followed by none or any characters, which applies to the token "Wegelagerer" in the field value.

**NOTE**

As the use of the = relation allows for normalisation – such as searching for the base form of a word – this can lead to results that are technically correct but not self-explanatory to users. In the example above, the search term *u* is used. The illustration shows a hit for the token "fur". Strictly speaking, however, "mice" should also be marked as a hit here, as the base form "mouse" contains a "u" and therefore fulfils the conditions of the pattern. Nevertheless, it is immediately apparent that the result is not entirely self-explanatory when searching for *generic* patterns – such as, in this case, all tokens containing "u" anywhere – than when searching for mouse, in which case it is easier to understand why "mice" appears in the results.

**== (exact equality with special characters)**

Using the same examples as in Section B.2.1.1.4, the == relation with special characters is illustrated in Figure 6. The first example searches the term Weg* and matches the field values "Weg" and

"Wegelagerer greifen an", as it searches the entire field value in its untokenised form; the search term is considered a match because it begins with "Weg" and is followed by either no characters or any characters in the entire string. The second example doesn't match at all, as both field values do not start with a "g".

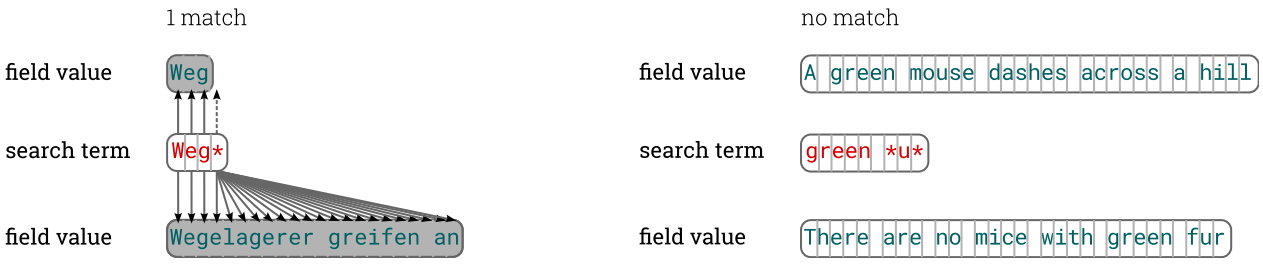


*Figure 6. The == relation with special character * allows for flexible matching patterns on entire field values.*

**==/regexp (exact equality with regular expression modifier and special characters)**

In direct comparison with the previous Section B.2.1.1.5, regular expression special characters are now used in combination with the regexp modifier; this allows more complex search patterns to be formulated, whilst also removing the requirement for the == relation that a match must be found across the entire field value.

This is illustrated in the second example in Figure 7, which uses the same search term as before: green.*u. However, as partial matches are now possible, the search for green followed by any number of arbitrary characters, followed by a u, matches "green mou" in one of the example sentences and "green fu" in the other.

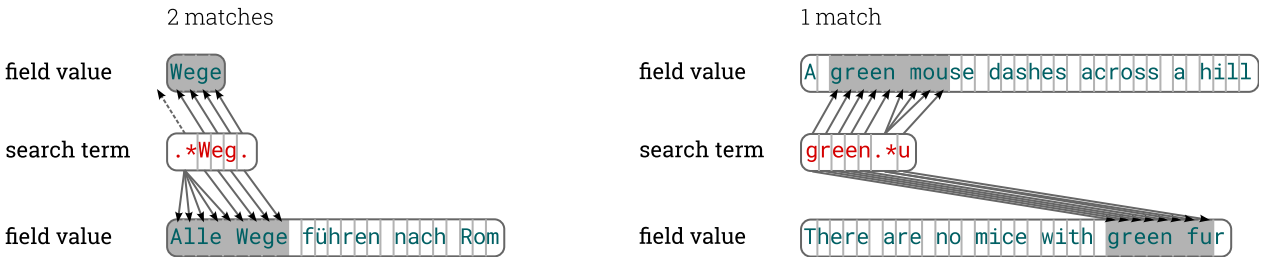


*Figure 7. The == relation with regexp modifier and special characters . and * allows for precise pattern based partial matches.*

Nevertheless, it is still possible to use the regular expression special characters ^ and $ to formulate search patterns that are evaluated at the start, at the end, or from start to end of the field value. See the following examples for illustration

1. Pattern from start: ^A matches "A" at the beginning of field values "Alle Wege führen nach Rom" and "A green mouse dashes across a hill"
2. Pattern from end: .ill$ matches "hill" at the end of field value "A green mouse dashes across a hill"
3. Pattern from start to end: ^A.*ill$ matches "A green mouse dashes across a hill" from start to end of the field value.

#### B.2.1.2. Query Examples

This section contains sample queries with detailed explanations of the semantics they express and the specific LexCQL features used.

- `lemma == "lead" AND definition = "river"`

  Entries whose lemma is exactly "lead" and whose definition contains the token "river".

- `lemma == "drive*" AND pos is VERB`

  Entries whose lemma begins with "drive" and are verbs according to the definition of the Universal Dependencies project. Query omits default namespace `https://universaldependencies.org/u/pos/` for field `pos`.

- `lemma = "seal" AND (pos is NOUN OR pos is "https://universaldependencies.org/u/pos/PROPN")`

  Entries whose lemma contains the token "seal" which are either noun or a proper noun (or both) according to the definition of the Universal Dependencies project. Query uses namespaces both implicitly and explictly and changes evaluation order with parentheses.

- `lemma = "seal" AND pos is NOUN OR pos is "https://universaldependencies.org/u/pos/PROPN"`

  Entries whose lemma contains the token "seal" and which are nouns, or entries that are proper nouns, or entries that are both. Query is evaluated left to right.

- `lemma ==/lang=de/ignoreCase/ignoreAccents "Grüße" OR definition =/regexp "^[Gg]r(ue|ü|u)[sßz]+e$"`

  Entries whose lemma is in German language and matches "Grüße" (including all variations in terms of capitalisation and accents), or whose definition contains a token matching a regular expression that captures various spelling variants, or both.

- `synonym ==/regexp "^car"`

  Entries that have a synonym with the prefix "car".

- `lang == de AND translation ==/lang=en car`

  Entries in German language that translate to "car" in English.

# Changelog

## 2026-09-16 — Publication of LexFCS (v1.0)

- Specify field select mechanism (`x-fcs-lex-fields`)
- LexCQL default field `lemma`, virtual index `any` to query all fields, more grammar-related fields
- Update LexCQL relation matching, remove `NOT` operator, add `<>` relation
- Specify `is` relation matching with default entity values
- Add extra appendix for LexCQL query matching, reorganize query examples
- Rephrasing, fix formatting

## 2025-06-20 — Publication of LexFCS (v0.3)

- Specify *Lex Fields* in the *Endpoint Description* for client self-configuration, describing what lex fields an endpoint supports for querying and that can appear in results (Lex Data View)
- Change Lex Data View namespace to `http://clarin.eu/fcs/dataview/lex`
- Update Lex Field Types
- Various fixes

## 2024-12-16 — Publication of LexFCS (v0.2)

- Specify *Lex Data Model* with *Lex Data View* serialization
- LexCQL for new *Lex Data Model*

## 2024-04-10 — Publication on CLARIN-ERIC GitHub

## 2023-05-09 — Publication of first draft (v0.1) on Zenodo

- Propose *LexCQL* query language and extension of *Hits Data View* for LexFCS
- LexFCS CQL Context Set for *LexCQL*