Literal Substring Search

Contents of this Page

- Literal Substring Implementation Details
  - Algorithm:
  - Example of use:
  - Associated JUnits:

Literal Substring Implementation Details

The same as the substring search but with special characters enabled.

Algorithm:

The Literal Substring search has the following characteristics:

- This search is case in-sensitive.
- It only searches on the literal property value and the literal reverse property value.
- A leading and trailing wild card is added to the token in the search text.
- The literal property part (without the wild cards) of the query is boosted by 50. This gives a literal match priority.
- Parsing is done with the following analyzers:
  - literal_propertyValue - Uses our custom literal analyzer. This literal analyzer uses Lucene's WhitespaceTokenizer with Lucene's LowerCaseFilter.
  - propertyValue - Uses our custom standard analyzer that has no stop words.

Example of use:

The following examples are based on the Automobiles coding scheme.

Example 1:

Search string: a^s

Lucene query: +literal_propertyValue:*a^s* literal_propertyValue:a^s^50.0

Result: 1 result

- entity code: SpecialCharactersConcept
- entity description: Concept containing special characters

Example 2:
Search string: a^s sp*cial

Lucene query: +spanNear([mask(spanWildcardQuery(literal_reverse_propertyValue:s^a*)) as propertyValue, mask(spanWildcardQuery (literal_propertyValue:sp*cial*)) as propertyValue], 0, true) ((+literal_propertyValue:a^s +literal_propertyValue:sp*cial)^50.0)

Result: 1 result

- entity code: SpecialCharactersConcept
- entity description: Concept containing special characters

Associated JUnits:

Junits can be found here: https://github.com/lexevs/lexevs/blob/master/lbTest/src/test/java/org/LexGrid/LexBIG/Impl/function/query/lucene/searchAlgorithms/TestLiteralSubString.java