CloudBackend

QUERY user guide

version 1.3.5.2

2021-04-28

Document overview

The following QUERY user guide document will give an overview on how to use the query calls in the SDK of CloudBackend (CBE). This is done by listing the code and some added usage examples.

Conventions

Describing text is printed in font Calibri. Descriptions inside listing tables uses font family serif.

Advanced query filters

Simple queries are performed by retrieving the entire contents of one or multiple containers. More complex queries are performed on the same set of containers using a filter. Please note that this documentation is a work in progress as from time to time refers to internal states, hidden and abstracted by the SDK.

Information how to apply Filters with code using the SDK can be found in the .h header files.

A query is performed by calling the query with filter. The payload can include the following fields:

Payload field	Description
container id	The ID of the container that is to be listed/seached (a decimal number). More than one **<container>** element may appear, provided that all IDs belong to the same container root.
query	The query. See below for more information about queries. *(optional)*
filter	A regular expression that is used to filter each result entry. See below for more information. *(optional)*
offset	The offset of the first object entry to return. The default, if unspecified, is 0. *(optional)*
count	The number of entries to return. The default is 100. *(optional)*
order	The sort order. Can be one of published, updated, title, length, s1, s2, s3 and s4. The default if not present is to sort by relevance according to the search criteria. *(optional)*
locale	A locale (like en_US or sv_SE) to be used when sorting. If unspecified or illegal, the C locale will be used. *(optional)*
numeric	Use numeric sorting, so that Q9 comes before Q10. *(optional)*
ascending	Can be true or false. The default is true.
include_deleted	Can be true or false. The default is false. *Entry with* **<dc:deleted></dc:deleted>** *tag is a deleted file.*
checksum	Can be true or false. The default is false. *Entry is returned with the* **<dc:md5>**123**</dc:md5>** *tag, which contains the checksum of the main stream.* **NOTE:** If set to 'true', it will make most other parameters like query, filter and order ineffective.
datasrc	Can be default or db. Default is key/value for the SDK, but if checksum is set to true the it will be set to db. The SDK can change the datasrc to db or key/value by setting it to default. Other clients can of course set it to db if required.

The query language

The CloudBackend Core object indexing service is based on a key/value database. Being an indexing service, it indexes words in an object and allows fast matching based on both boolean and probabilistic methods.

An index database excels at finding objects (in this case, object metadata entries) that include a specific word (called a *term*). Think of it as an index at the end of a book, where you can quickly locate all the pages that refer to a specific word or concept. Multiple result sets can then be merged based on common boolean operations like AND, OR, NOT etc or simply weighted based on the relevance (which is more useful in an off-line indexing scenario than metadata queries.)

The query syntax

The query language is very simple, yet rather powerful. A query is a white-space separated list of search terms, where each term has one of the following forms:

```
(+/-)? (prefix)? (term with no blanks)
(+/-)? (prefix)? (term with ' and blanks)
(+/-)? (prefix)? (term with " and blanks)
```

A *term* is the word or phrase you wish to search for. Phrases are always enclosed by single or double quotes. The *prefix*, which is optional, is the key name. Finally, you may include a + or - sign in front of the prefix (or term, if there is no prefix) to indicate that the term *must* or *must not* exist.

Some fields will also be indexed *stemmed* (see next section). Searching for terms in these fields will actually search for the stemmed term. By using the phrase syntax (single or double quotes) or by beginning the term with a capital letter will force the query parser to search for the word as-is instead.

Building the query tree

All search terms are combined into a query tree before being sent to the key/value database, using the following rules:

- All "plus" terms are combined with an AND relation, which simply means that all of them must exist.
- All "minus" terms are combined with an OR relation, and the group is added to the query with an AND_NOT relation, meaning that none of the terms must exist.
- All un-prefixed terms with neither a plus or minus sign in front of them are combined with an OR relation, and then added to the query with an AND_MAYBE relation. This means that they do not have to be present, but entries that do include them will get higher ranking than those which don't.
- Prefixed terms with neither a plus or minus sign in front of them are grouped by their prefix. All terms with the same prefix are grouped with an OR relation, and all groups are then combined with an AND relation. The group of groups is then added to the query with an FILTER relation, which is equivalent with an AND relation, except that FILTER terms do not affect the ranking at all.

See below for example queries that might make it easier to understand how the query tree is created from a query string.

Fields and prefixes

There are a few pre-defined fields in the metadata that are bound to a so-called query prefix, and they are listed below. A query prefix is just a way of saying that the specific word has a special meaning and should be searchable as such. For example the **title** prefix, means that the words in the title for an object will not only be indexed as-is, but also with a prefix. **title:spring** thus finds entries where the word spring is found in the title. Note, title has nothing to do with the name of the author or creator of an object.

The list of pre-defined prefixes are shown below, as well at the **entry** field they are bound to. Using the metadata capabilities in the SDK, the developer can add an arbitrary number of key/value pairs to an objects metadata. They can be in the form of non-indexed key/value pairs or indexed key/value pairs that are searchable using filters (but take up more space).

All defined prefixes are boolean prefixes, which means that if specified in a query, boolean semantics apply and not probabilistic.

Some of the fields (currently author, content, rights, keyword, recipient, s1, s2, s3, s3, summary, tag and title) will also be indexed stemmed, that is, in addition to being indexed as-is (but with a prefix), the root of the word will also be indexed (with the prefix). A query with the term **keyword:phones** will also match an entry with the field **<keyword>phone</keyword>**.

The field keyword, s1, s2, s3, s4, summary and title and content will additionally be indexed without the prefix, since they are assumed to include textual content.

Additionally, the fields **published**, **updated**, **link rel="alternate" length="..."** and **title** are treated special to make it possible to sort the query result based on these fields.

The inner working of metadata has been modelled after the W3C Atom standard with the ability for user custom extensions of key/value data and tags.

Query prefix	Key/Value field	Description
author	atom:entry	The author's name, email or URI.
country	dc:country	A two-letter ISO country code (same as used in domain names).
(no prefix)	atom:content	Full-text index
object	dc:object	The object ID as a decimal number.
	dc:enddate	This key is not indexed as-is. See below.
flag	dc:flag	A general-purpose flag that can be used by applications.
container	dc:container	The ID of the container the object is located in (a decimal number).
	dc:atom:id	This key is not indexed.
keyword	dc:keyword	Keywords that describe the object.
lang	dc:lang	The language of the object as a two-character ISO code.
month		This prefix is calculated from the startdate and enddate keys.
namespace	dc:namespace	The XML namespace if the object is a XML document.
	atom:published	This key is not indexed as-is. See below.
pubmonth		This prefix is calculated from the atom:published key.
pubweek		This prefix is calculated from the atom:published key.
pubyear		This prefix is calculated from the atom:published key.
recipient	dc:recipient	The recipient's name, email or URI.
rights	atom:rights	A copyright string.
s1	dc:s1, ni:s1	A general purpose prefix that is also sortable.
s2	dc:s2, ni:s2	A general purpose prefix that is also sortable.
s3	dc:s3, ni:s3	A general purpose prefix that is also sortable.
s4	dc:s4, ni:s4	A general purpose prefix that is also sortable.
schema	atom:category type="schema" scheme="..."	The W3C schema of objects of type XML document if they have a schema defined.
	dc:startdate	This key is not indexed as-is. See below.
summary	atom:summary	A summary of the object's content.
tag	dc:tag	A user-specified tag (like "family", "London", "My wedding" etc).
title	atom:title	A objects title or rather simply the title of the object.
type	atom:link rel="alternate" type="..."	The objects MIME type.
week		This prefix is calculated from the startdate and enddate keys.
year		This prefix is calculated from the startdate and enddate keys.

Flag values in use

Flags are application specific and defined by the developer. These flags are used for some of the built in capabilities of CloudBackend Core.

User/group entries

- hidden
- avatar

IMAP mail

- answered
- deleted
- flagged
- seen
- draft
- recent

Special handling of published, startdate and enddate

The fields **published**, **startdate** and **enddate** will not be indexed as-is. Instead, the dates specified will be recalculated as pubyear/year (the year with 4 digits), pubmonth/month (the month as two digits, 01 to 12) and pubweek/week (the ISO week number, 01 to 53). In case of startdate and enddate, the full range will be indexed.

For example, give the following fields in the Atom entry,

```
<atom:entry>
 <atom:published>2020-08-30T11:12:57Z</atom:published>
 <dc:startdate date="2020-12-10" time="01:54:00" zone="Z"/>
 <dc:enddate date="2021-01-09" time="00:00:00" zone="Z"/>
 <!-- more ... -->
</atom:entry>
```

The following terms will be indexed:

- pubyear:2020
- pubmonth:08
- pubweek:35
- year:2020
- year:2021
- month:12
- month:01
- week:50, week:51, week:52, week:01, week:02

The idea is that any object, be it a text document, picture, calendar entry, email or to-do note, can be marked with a date or date range. Relevant objects can then be found quickly and presented in a calendar or reminder view in an application.

(Note that since the year, month and week is indexed individually, the application will have to post-filter the search result, since the query "({year:2020 AND month:12})" will yield a false match.)

Special handling of content

The **content** element will always be stripped from the entry. However, it is still indexed (as-is and stemmed), which means that this field is perfect for full-text indexing. The client can thus easily create a full-text index of any object simply by transforming the object to plain text, put the text inside a **content** element in the object's entry and call **saveMetadata**.

Unknown prefixes

When an unknown prefix is found, it is simply indexed as-is, with prefix. It is not added without prefix and it is not stemmed. This applies to all user defined key/values and tags.

Please note that key/value pairs and tags that in the SDK are defined as non-indexed will not be possible to query. They are retrieved by querying the object on other parameters and then retrieving the metadata to view them. It is recommended to use non-indexed key/value pairs as the default for all meta-data that is not required to be searchable as this is much more economical with less used storage space for the tenant. It will also increase the query performance for the keys that should be searchable.

Example queries

Normal, plus and minus terms

```
dogs cats +black -cool -"hot dogs"
```

The query above searches for items that include the words "dog" or "cat" in one of several "text" fields (currently keyword, s1, s2, s3, s4, summary and title or content). Note the singular form! Un-prefixed words are stemmed. Additionally, the word "black" must be present in the object for a match. The word "cool" and the phrase "hot dogs", on the other hand, must not be present at all.

The query above will be translated into the following query tree:

```
(("black") AND_MAYBE ("dog" OR "cat")) AND_NOT ("cool" OR "hot dogs")
```

If the query instead had been written using an initial capital letter on the words "dogs" or "cats", or had these words been quoted, like this:

```
"dogs" cats +black -cool -"hot dogs"
```

The resulting query three would have been:

```
(("black") AND_MAYBE ("dogs" OR "cats")) AND_NOT ("cool" OR "hot dogs")
```

(Note the plural form on "dogs" and "cats".)

Prefixed terms

Consider the following query:

```
title:money
```

It will simply find all entries that include the word "money" in the title. The next query will find all entries that describe JPEG or GIF images (based on the objects mime-type, which requires that the object have a stream 0 from where the mime-type is deducted, objects can not be queried on mime-type for stream 1..n), and has the word "money" in the title.

```
title:money type:image/png type:image/gif
```

Note that since the type field is searched for twice, the terms image/png and image/gif are combined with an OR relation, like this:

```
"title:money" AND (type:image/png OR type:image/gif)
```

This is probably the expected behaviour for most fields, but perhaps not all. Suppose one would like to find all entries that include the words "money" and "dollars" in the title. The following query would *not* work:

```
title:money title:dollar
```

This is since it would return all entries with either the word "money" or the word "dollar" in it. Here the prefixed "plus" terms should be used:

```
+title:money +title:dollar
```

The query above will correctly translate into the correct query tree:

```
title:money AND title:dollar
```

Date queries

The following query example will return a set of candidate entries to be displayed in a calendar view of January 2021:

```
year:2021 month:01
```