1. Cxense Search 2
1.1 Cxense Searchについて 2
1.1.1 Cxense Search実装例（インデックス） 2
1.1.2 Cxense Search実装例（クエリ） 15
1.2 Cxense Searchについて(旧バージョン) 33
1.2.1 Cxense Search API ガイド 33
1.2.1.1 Cxense Search - スタートガイド 34
1.2.1.2 Cxense Search - API 名前空間 37
1.2.1.3 Cxense Search - /api/index - 索引 37
1.2.1.4 Cxense Search - /api/content - フィードと削除 41
1.2.1.5 Cxense Search - /api/search - HTTP API 43
1.2.1.5.1 Cxense Search - p_aq - 高度なクエリ言語 46
1.2.1.5.2 Cxense Search - p_aq - 日付のフィルター 54
1.2.1.5.3 Cxense Search - p_uq - ユーザーが検索したキーワードを認識 56
1.2.1.5.4 Cxense Search - p_rs - 結果セットの設定 57
1.2.1.5.5 Cxense Search - p_f - ファセット 58
1.2.1.5.6 Cxense Search - p_sm - 並び替えとランキング 61
1.2.1.5.7 Cxense Search - p_sm - 緯度経度での検索 61
1.2.1.5.8 Cxense Search - p_dr - 重複排除 64
1.2.1.5.9 Cxense Search - p_c / p_s - ページ操作 65
1.2.2 Cxense Search よくあるお問い合わせ 66
1.2.2.1 Cxense Search 利用フロー 66
1.2.2.2 frontpageに分類されたページの検索 67
1.2.2.3 Webページのタイトル 68
1.2.2.4 Webページの索引付け対象 68
1.2.2.5 同義語辞書の注意事項 68
1.2.2.6 同義語辞書の管理方法 71
1.2.2.7 文字の正規化 73
1.2.2.8 索引からのドキュメントの削除 73
1.2.3 Cxense Search 追加情報 74
1.2.3.1 Cxense Search - 辞書管理 74
1.2.3.2 Cxense Search - 関連性のチューニング 76
1.3 現行のSearchと旧バージョンのSearchの対比 78

Cxense Search

- 新しいSearchと従来のSearchのソリューションの対比
- 以前のCxense Searchについて
- 新しいCxense Searchについて

Cxense Searchについて

Cxense Searchではドキュメントの検索と更新に2つのAPIを提供しています。これらはCxense InsightのAPIドキュメントの一部として直接アクセスできます。

- /document/search
- /document/update

辞書データの作成、更新、検索の利用については Cxense Insight API の以下の機能を利用します。

- /processing/dictionary/create
- /processing/dictionary/update
- /processing/dictionary/search

Searchの統計は以下のAPIで見られます。

- /report/search
- /report/search/usage

これらのAPIを利用したチュートリアルを以下に紹介します。

- Cxense Search実装例（クエリ）
- Cxense Search実装例（インデックス）

Cxense Search実装例（インデックス）

English | 日本語

- 概要
- 前提条件
    - 自動クロールを無効化します
- Webページのアップロードを行います
- ドキュメントをフィールドにセットしてアップロードします。
- ドキュメントをバッチでアップロードします。
- ドキュメントを消去します。

概要

Cxense Search実装例（クエリ）では既にインデックスにあるドキュメントを検索する新しいSearch APIの手法を紹介しました。
このページでは自動的にクロールされてインデックスにpushされたドキュメントを、手動で置き換え又は補完する方法を解説します。

APIの詳細なドキュメントについてはリンク先：/document/update , /document/search , /document/delete をご参照ください。

以下全てのコードはPythonのバージョン3.Xを利用します。実際のサイトで利用可能です。

前提条件

何の準備もしないまま、コードサンプルを実行しようとすると、大抵は"Permission Denied"や "Error while processing request"のエラーが返ってくるでしょう。
下記の項目はこのページのスクリプトを実行するに当たって、行っておくべき前提条件です。

- まだ利用できるサイトグループをお持ちでは無い場合
  Cxenseの担当者に御依頼下さい。
- サイトグループにてサイトを追加します。(まだ作成していない場合。)
- 対象サイトグループに対して更新権限を付与したユーザーで作業します。
- 自動クロールを停止します。(スクリプトについては下記で説明します。)
- サイトの言語設定を行います。ISO 639-1 two-letter language code 形式で

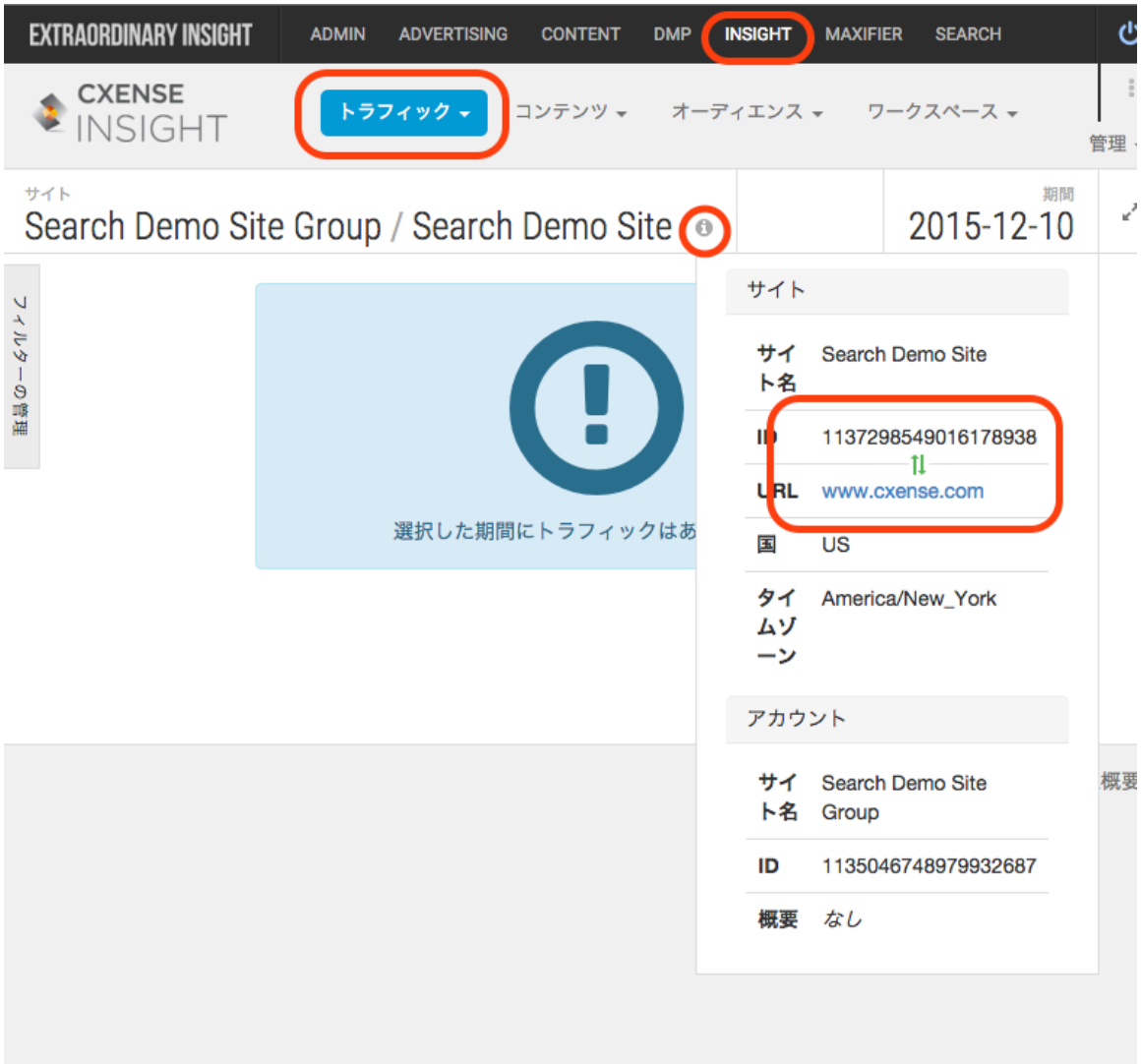
指定してください。(shown in script below)

- 利用するサイトは有効かつプライベートドメインを利用している必要があります。
  インデックスにアップロードするドキュメントのURLはサイトのURLと一致させて下さい。
- 最低1ページ以上のクロール済みのサイトを必要とします。何もページがない場合、ダミーページを用意して3PV以上発生させてクローラーを巡回させて下さい。(最大1時間程度かかることがあります。)
  ダミーページはSearch稼働後に削除出来ます。

下記のスクリプトを実行するために自動クロールを無効化してください。
右の図ではURLとサイトIDをCxense Insightの画面から取得する方法を示しています。(このケースではサイトID1137298549016178938で利用するURL(ドメイン)はwww.cxense.comになります。)

For running the scripts in this tutorial you will need the following:

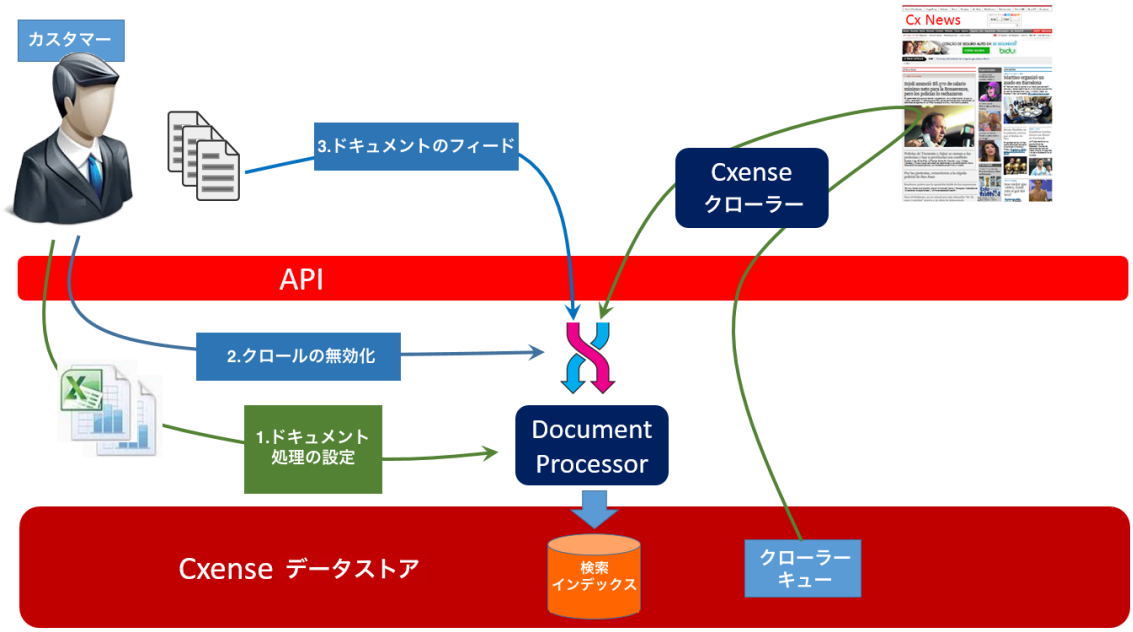
- The site id (see figure to the right)
- The site's URL/domain (see figure to the right)
- Your Cxense user id (the one you use to log into Cxense Insight)
- Your secret Cxense API Key
- Your customer prefix (contact your Cxense representative to get one - only required in the case of documents with fields)



自動クロールを無効化します

自動クロール中にAPIでドキュメントをアップロードする事が出来ません。

以下の青の2の切り替えで示されているように、どちらか一方でなければなりません。



下記でドキュメントをフィードする前に、自動クロールを無効化する手順を紹介します。

```
_username     = "<YOUR CXENSE USER ID>"
_secret       = "<YOUR CXENSE API KEY>"
_siteId       = "<YOUR SITE ID>"
_siteLanguage = "<2 LETTER ISO 639-1 LANGUAGE CODE>"
import json, datetime, hmac, hashlib, http.client

def cxApi(path, obj):
    date = datetime.datetime.utcnow().isoformat() + "Z"
    signature = hmac.new(_secret.encode('utf-8'), date.encode('utf-8'), digestmod=hashlib.sha256).hexdigest()
    headers = {"X-cXense-Authentication": "username=%s date=%s hmac-sha256-hex=%s" % (_username, date,
signature)}
    connection = http.client.HTTPSConnection("api.cxense.com", 443)
    connection.request("POST", path, json.dumps(obj), headers)
    response = connection.getresponse()
    status = response.status
    responseObj = json.loads(response.read().decode('utf-8'))
    connection.close()
    return status, responseObj

def errorHandling(status, response, message):
    if status != 200:
        raise Exception("%s (http status = %s, error details: '%s')" % (message, status, response['error']))

def disableCrawling(siteId):
    status, response = cxApi('/site/update', {"siteId":siteId, 'automaticCrawling':False})
    errorHandling(status, response, "Failed to disable crawling")

def setSiteLanguage(siteId, language):
    status, response = cxApi('/site/update', {"siteId":siteId, 'language':language})
    errorHandling(status, response, "Failed to set the site's language")

def runOnce(siteId, language):
 setSiteLanguage(siteId, language)
    disableCrawling(siteId)

runOnce(_siteId, _siteLanguage)
```

理論上、自動クロールは任意の回数無効化できますがスワップ後に再確認することをお勧めします。

カスタムアップロードのスクリプトを作りましょう（Hello World!）

基本となるカスタムインデックスのスクリプトを作成します。

```
_username     = "<YOUR CXENSE USER ID>"
_secret       = "<YOUR CXENSE API KEY>"
_siteId       = "<YOUR SITE ID>"
_siteUrl      = "<THE URL OF THE SITE>"

import json, datetime, hmac, hashlib, http.client

def cxApi(path, obj):
    date = datetime.datetime.utcnow().isoformat() + "Z"
    signature = hmac.new(_secret.encode('utf-8'), date.encode('utf-8'), digestmod=hashlib.sha256).hexdigest()
    headers = {"X-cXense-Authentication": "username=%s date=%s hmac-sha256-hex=%s" % (_username, date,
signature)}
    connection = http.client.HTTPSConnection("api.cxense.com", 443)
    connection.request("POST", path, json.dumps(obj), headers)
    response = connection.getresponse()
    status = response.status
    responseObj = json.loads(response.read().decode('utf-8'))
    connection.close()
    return status, responseObj

def errorHandling(status, response, message):
    if status != 200:
        raise Exception("%s (http status = %s, error details: '%s')" % (message, status, response['error']))

def uploadDocument(siteId, url, title):
    status, response = cxApi('/document/update', {"siteId":siteId, 'url':url, "title":title})
    errorHandling(status, response, "Unable to upload document")
    return response

def search(siteId):
    status, response = cxApi('/document/search', {"siteIds":[siteId]})
    errorHandling(status, response, "Unable to retrieve search result")
    return response

url = _siteUrl + '/demo'
title = 'Hello World!'
uploadDocument(_siteId, url, title)
print(search(_siteId))
```

2つの関数「cxApi()」と「errorHandling()」は、このチュートリアル全てのスクリプトの一部になります。
何度も同じコードが出てくるので、これらは以下のコード例では省略されます。

上記のスクリプトを実行した結果は以下のように出力されます。なお、Cxenseのフレームワークがアップロードしたフィールド以外にも、複数のフィールドを自動的に追加しています。

```
{
 'totalCount': 1,
 'matches': [
  {
   'id': '2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583',
   'siteId': '1137298549016178938',
   'score': 1.0,
   'fields': [
    {'value': 'Hello World!', 'field': 'title'},
    {'value': 'fc22e236724ac00585f0ce43e91c15de387d4531b0b2cc0137684fd6319c5a47', 'field':
'contenthash'},
    {'value': '2015-08-09T18:03:44.000Z', 'field': 'createtime'},
    {'value': '2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583', 'field': 'id'},
    {'value': '2015-08-09T18:03:44.000Z', 'field': 'recs-publishtime'},
    {'value': '1137298549016178938', 'field': 'siteid'},
    {'value': '2015-08-09T18:05:48.000Z', 'field': 'timestamp'},
    {'value': 'http://cxense.com/demo', 'field': 'url'},
    {'value': '2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583', 'field': 'urlhash'}
   ],
  }
 ]
}
```


Webページのアップロードを行います

前の項目ではシンプルにURLとタイトルフィールド（これはidフィールドのベースとなります。）をアップロードしました。（自動的に追加されたフィールドは除きます。）
ここでは、WEBページをアップロードをドキュメントとしてアップロードする方法を説明します。インデックスするページを参照してください。

```
<!DOCTYPE html>
<html>
 <head>
  <title>An UFO has just landed on the White House Lawn</title>
  <meta property="articleId" content="24757772"/>
  <meta property="og:article:published_time" content="2015-04-01T13:52:00Z"/>
  <meta property="og:description" content="Breaking News"/>
 </head>
 <body>
  <h1>An UFO has just landed on the White House Lawn</h1>

  <p>We have just learned that an UFO has landed on the White House lawn.</p>
  <p>This article will be updated as we learn more.</p>

  <h2>Eyewitnesses Confirm</h2>
  <p>Several eyewitnesses have confirmed the White House UFO sighting.</p>
 </body>
</html>
```

以下はアップロード用のコードと、検索結果として返されています。

```
def uploadDocument(siteId, url, title, sections, publishTime, articleId, description):
   reqObj = {'siteId':siteId, 'url':url, 'title':title, 'sections':sections, 'publishTime':publishTime, 'articleId':articleId,
'description':description}
   status, response = cxApi('/document/update', reqObj)
   errorHandling(status, response, "Unable to upload document")
   return response

def search(siteId):
   status, response = cxApi('/document/search', {"siteIds":[siteId]})
   errorHandling(status, response, "Unable to retrieve search result")
   return response

url         = 'http://cxense.com/demo'
title       = 'An UFO has just landed on the White House Lawn'
articleId   = '24757772'
publishTime = '2015-04-01T13:52:00Z'
description = 'Breaking News'
sections  = [
   {
      'body':[
         'We have just learned that an UFO has landed on the White House lawn.',
         'This article will be updated as we learn more.'
      ]
   },
   {
      'heading':'Eyewitnesses Confirm',
      'body':[ 'Several eyewitnesses have confirmed the White House UFO sighting.' ]
   },
]
uploadDocument(_siteId, url, title, sections, publishTime, articleId, description)
print(json.dumps(search(_siteId), indent=4, sort_keys=True))
```

以下、検索結果となります。


```
{
   "matches": [
      {
         "fields": [
            {
               "field": "body",
               "value": [
                  "We have just learned that an UFO has landed on the White House lawn.",
                  "This article will be updated as we learn more.",
                  "Several eyewitnesses have confirmed the White House UFO sighting."
               ]
            },
            {
               "field": "contenthash",
               "value": "f4fdc0ec54df677050fa12b2c87b549c63c27f2843b80301761bfcdd9b30879b"
            },
            {
               "field": "createtime",
               "value": "2015-08-09T18:03:44.000Z"
            },
            {
               "field": "description",
```

```
    "value": "Breaking News"
  },
  {
    "field": "heading",
    "value": "Eyewitnesses Confirm"
  },
  {
    "field": "id",
    "value": "2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583"
  },
  {
    "field": "kw-concept",
    "value": "eyewitnesses confirm"
  },
  {
    "field": "kw-entity",
    "value": [
      "confirm",
      "news",
      "white house",
      "white house lawn",
      "white house ufo"
    ]
  },
  {
    "field": "publishtime",
    "value": "2015-04-01T13:52:00.000Z"
  },
  {
    "field": "recs-articleid",
    "value": "24757772"
  },
  {
    "field": "recs-publishtime",
    "value": "2015-04-01T13:52:00.000Z"
  },
  {
    "field": "siteid",
    "value": "1137298549016178938"
  },
  {
    "field": "teaser",
    "value": "Breaking News"
  },
  {
    "field": "timestamp",
    "value": "2015-09-05T19:14:50.000Z"
  },
  {
    "field": "title",
    "value": "An UFO has just landed on the White House Lawn"
  },
  {
    "field": "url",
    "value": "http://cxense.com/demo"
  },
  {
    "field": "urlhash",
    "value": "2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583"
```

```
        }
      ],
      "id": "2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583",
      "score": 1.0,
      "siteId": "1137298549016178938"
    }
```

```
    ],
    "totalCount": 1
}
```

ドキュメントをフィールドにセットしてアップロードします。

下記ではCxenseのオスロ本社に関してのドキュメントをアップロードしています。
これには都市（string）、社員数（number）、設立年月日（time）、所在地の緯度経度（geopoint）を格納しています。
前までのコードサンプルにはなかったカスタマープリフィックスが必要になっているので注意してください。

```
_username     = "<YOUR CXENSE USER ID>"
_secret       = "<YOUR CXENSE API KEY>"
_siteId       = "<YOUR SITE ID>"
_prefix       = "<YOUR CUSTOMER PREFIX>"

def uploadDocument(siteId, url, fields):
    status, response = cxApi('/document/update', {'siteId':siteId, 'url':url, 'fields':fields})
    errorHandling(status, response, "Unable to upload document")
    return response

def search(siteId):
    status, response = cxApi('/document/search', {"siteIds":[siteId]})
    errorHandling(status, response, "Unable to retrieve search result")
    return response

url    = 'http://cxense.com/demo'
fields = [
    {'field':'city',      'type':'string',   'value':'Oslo'},
    {'field':'employees', 'type':'number',   'value':'45'},
    {'field':'started',   'type':'time',     'value':'2010-10-30T00:00:00.000Z'},
    {'field':'location',  'type':'geopoint', 'value': {'latitude':59.95, 'longitude':10.75}}
]
for field in fields:
    field['field'] = '%s-%s' % (_prefix, field['field'])

uploadDocument(_siteId, url, fields)
print(json.dumps(search(_siteId), indent=4, sort_keys=True))
```

One thing to keep in mind is that the field keys cannot be longer than 26 characters (or 30 if you include the length of the customer prefix that we are adding within the loop above), and the value can be a maximum of 40 characters. The set of allowed key characters is limited to lower case a-z, digits and the hyphen ('-') whereas the value can hold any readable alphanumeric character regardless of script (Latin, Hebrew, Cyrillic, Kanji, you name it) as well as quite a few special characters. Here are some regular expressions defining a valid field key and a valid field value respectively:

- ^[a-z0-9¥-]{1,26}$
- ^[¥w0-9@&*()¥{¥}¥- ¥¥¥.,/º¥?'%%+=#$^~<>|!; ´ ¥[¥]:]{0,40}$

検索結果に同じドキュメントが返ってくるのを確認できます。

```
{
    "matches": [
        {
            "fields": [
                {
                    "field": "sds-location",
```

```
            "type": "geopoint",
            "value": {
               "latitude": 59.95,
               "longitude": 10.75
            }
         },
         {
            "field": "sds-employees",
            "type": "number",
            "value": 45
         },
         {
            "field": "sds-started",
            "type": "time",
            "value": "2010-10-30T00:00:00.000Z"
         },
         {
            "field": "contenthash",
            "value": "4f53cda18c2baa0c0354bb5f9a3ecbe5ed12ab4d8e11ba873c2f11161202b945"
         },
         {
            "field": "createtime",
            "value": "2015-08-09T18:03:44.000Z"
         },
         {
            "field": "id",
            "value": "2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583"
         },
         {
            "field": "recs-articleid",
            "value": "24757772"
         },
         {
            "field": "recs-publishtime",
            "value": "2015-08-09T18:03:44.000Z"
         },
         {
            "field": "sds-city",
            "value": "Oslo"
         },
         {
            "field": "siteid",
            "value": "1137298549016178938"
         },
         {
            "field": "timestamp",
            "value": "2015-09-05T20:42:51.000Z"
         },
         {
            "field": "url",
            "value": "http://cxense.com/demo"
         },
         {
            "field": "urlhash",
            "value": "2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583"
         }
      ],
      "id": "2d926c0d05ef0a383d1c3d716dacc7d2fcdd1583",
      "score": 1.0,
```

```
    "siteId": "1137298549016178938"
}
```

```
    ],
    "totalCount": 1
}
```

ドキュメントをバッチでアップロードします。

上記の例のように、ドキュメントを一回一回アップロードするのは非常に効率が悪いため、バッチを使い効率よくアップを行えます。
一回のバッチで同時に100個のドキュメントをアップロードできます。

```
_username     = "<YOUR CXENSE USER ID>"
_secret       = "<YOUR CXENSE API KEY>"
_siteId       = "<YOUR SITE ID>"
_prefix       = "<YOUR CUSTOMER PREFIX>"
_batchSize    = <BATCH SIZE>

batch = []

def flushDocumentBuffer():
    global batch
    status, response = cxApi('/document/update', batch)
    errorHandling(status, response, "Unable to upload document batch")
    batch = []
    return response

def addToDocumentBatch(siteId, url, fields):
    global batch
    batch.append({'siteId':siteId, 'url':url, 'fields':fields})
    if len(batch) >= _batchSize:
        flushDocumentBuffer()

documents = [
    {
        "url" : 'http://cxense.com/oslo-office',
        "fields" = [
            {'field':'city',      'type':'string',   'value':'Oslo'},
            {'field':'employees', 'type':'number',   'value':'45'}
        ]
    },
    {
        "url" : 'http://cxense.com/melbourne-office',
        "fields" = [
            {'field':'city',      'type':'string',   'value':'Melbourne'},
            {'field':'employees', 'type':'number',   'value':'25'}
        ]
    }
]

for document in documents:
 for field in document['fields']:
     field['field'] = '%s-%s' % (_prefix, field['field'])
    addToDocumentBatch(_siteId, document['url'], document['fields'])
flushDocumentBuffer()
```

直接ドキュメントをアップロードする代わりに、バッチでアップロードが可能です。
バッチは一定のサイズに達する度にバッチ全体のアップアロードが行われます。(変数_batchSizeで定義されます。)
残った文書が何もない場合でも、同様です。バッチ内にある文書は全てアップロードされます。

ドキュメントを消去します。

検索インデックスからドキュメントを消去します。

```
_username     = "<YOUR CXENSE USER ID>"
_secret       = "<YOUR CXENSE API KEY>"
_siteId       = "<YOUR SITE ID>"

def deleteDocument(url):
    status, response = cxApi('/document/delete', {"siteId":_siteId , 'url':url})
    errorHandling(status, response, "Unable to delete document")

url = _siteUrl + '/demo' #これは最初の方に出た例"Hello World!"のドキュメントです。
deleteDocument(url)
```

Cxense Search実装例（クエリ）

English | 日本語

- 概要
- 最も簡単に実行できるSearchプログラム
- 検索結果の構造
- 検索結果の絞り込み
- Searchクエリ
    - クエリ構文
    - 複数のクエリの組み合わせ方
- ランキングとソート
- ハイライト表示
- ファセット
- フィルター
- クエリ使用情報の取得
- 辞書をベースとした機能
    - 辞書管理方法
    - 辞書の使い方
    - 辞書テンプレート
        - 入力文字チェック - もしかして?検索
        - クエリコンプリション
        - 同義語登録

概要

本ページのチュートリアルでは Cxense Insight API の以下の機能を使います。

- /document/search
- /processing/dictionary/search
- /reports/search
- /reports/search/usage

このチュートリアルでは検索についてのみ取り扱います。
データのアップロードとインデックスについては別項（Cxense Search実装例（インデックス））で解説します。

⚠ このチュートリアルの前提条件は、既にサイトがホワイトリスト化され、Cxense Insightタグが展開されている事を想定しています。
また、3PV以上のトラフィック（クローラーがサイトを巡回する為の最小PV数）以上あるページがサイト内にあることを想定します。

以下全てのコードはPythonのバージョン3.Xを利用します。
実際のサイトで利用するには以下が必要となります。

- サイトID（サイトIDの確認方法）
- CxenseユーザーID（Cxense Insightにログインするために使うID）
- CxenseユーザーID に紐づく API Key

最も簡単に実行できるSearchプログラム

まずはシンプルなコードから開始します（1行目は UNIX/Macバージョン、2行目はWindowsバージョン）。

```
$ python cx.py /document/search '{"siteIds":["サイトIDを指定"]}'
> python cx.py /document/search "{¥"siteIds¥":[¥"サイトIDを指定¥"]}"
```

cx.py のインストールと使用方法についてはこちらでご紹介しています。
同じコマンドとして使用する Pythonの Searchプログラムは以下をご覧ください。
上のコマンドでのアウトプットとしたのスクリプトは同じ出力結果になります。

```
_username     = "CxenseのユーザーIDを指定"
_secret       = " CxenseのAPIキーを指定"
_siteId       = "サイトIDを指定"

import json, datetime, hmac, hashlib, http.client

def cxApi(path, obj):
    date = datetime.datetime.utcnow().isoformat() + "Z"
    signature = hmac.new(_secret.encode('utf-8'), date.encode('utf-8'), digestmod=hashlib.sha256).hexdigest()
    headers = {"X-cXense-Authentication": "username=%s date=%s hmac-sha256-hex=%s" % (_username, date,
signature)}
    connection = http.client.HTTPSConnection("api.cxense.com", 443)
    connection.request("POST", path, json.dumps(obj), headers)
    response = connection.getresponse()
    status = response.status
    responseObj = json.loads(response.read().decode('utf-8'))
    connection.close()
    return status, responseObj

def errorHandling(status, response, message):
    if status != 200:
        raise Exception("%s (http status = %s, error details: '%s')" % (message, status, response['error']))

def search(siteId):
    status, response = cxApi('/document/search', {"siteIds":[siteId]})
    errorHandling(status, response, "Unable to retrieve search result")
    return response

result = search(_siteId)
print(result)
```

2つの関数「cxApi()」と「errorHandling()」は、このチュートリアル全てのスクリプトの一部になります。
何度も同じコードが出てくるので、これらは以下のコード例では省略されます。

内部動作で全てのスクリプトは以下のAPIを実行します。

/document/search { "siteIds":['ご利用中のsite id'] }

実行後、ターミナルウィンドウへ結果をダンプします。
返ってくる結果は、辞書の形状を取っています。
ターミナルウィンドウへは10個分の配列が返されます。構造は以下の通りです。

{ 'totalCount': 6707, 'matches': [{}, {}, {}, {}, {}, {}, {}, {}, {}, {}] }

上記の例でクエリリクエストに使っているオブジェクトはサイトIDだけになります。
他にもクエリに利用できる書式は幾つかあります。
Cxense searchのクエリを指定しない場合はすべてのドキュメントがヒットします。
したがって、この例では 'totalCount': 6707 ドキュメントの総数がインデックスの合計サイズになります。

検索結果の構造

print(result['matches'][0])

下記の例では、返却される結果を分かりやすく一部のみ切り取り、インデントを挿入して見易くしています。
このデータは返却結果で一番最初にマッチングしたものです。

```
{
 'id': '74251d7b4d8ee736359a0f62c097f9eb2f5f38ef',
 'score': 1.0,
 'fields': [
  {'field': 'body',                    'value': ['Desejo de praticamente toda mulher, perder aquela gordurinha .....]},
  {'field': 'category0',                'value': 'sua-pele'},
  {'field': 'category1',                'value': 'sua-pele|novidades'},
  {'field': 'contenthash',              'value':
'ab464f79c708b60616a7c91e62f76850552c35c8ff9171c7d6c1a49febd4b281'},
  {'field': 'createtime',               'value': '2015-08-05T20:34:52.000Z'},
  {'field': 'description',              'value': 'Desejo de praticamente toda mulher, perder aquela gordurinha .....'},
  {'field': 'dominantimage',              'value': 'http://p2.trrsf.com/image/fget/cf/600/600/images.terra.com/.....'},
  {'field': 'dominantimagedimensions',    'value': '600x600', },
  {'field': 'dominantthumbnail',          'value':
'http://content-thumbnail.cxpublic.com/content/dominantthumbnail.....'},
  {'field': 'dominantthumbnaildimensions', 'value': '300x300', },
  {'field': 'heading',                  'value': 'Drenagem Lipossônica Ativa reduz gordura da barriga em uma s'},
  {'field': 'id',                     'value': 'd0e85cb9001e3f929fedcc8c2f2d64454f089239'},
  {'field': 'kw-category',              'value': ['novidades', 'sua-pele']},
  {'field': 'kw-concept',               'value': ['barriga', 'drenagem', 'gordura', 'gordura localizada', 'sessão']},
  {'field': 'kw-entity',              'value': ['drenagem lipossônica ativa', 'stesis']},
  {'field': 'kw-location'.              'value': 'são paulo'},
  {'field': 'kw-pageclass',             'value': 'article'},
  {'field': 'kw-person',                'value': 'dino volpa', },
  {'field': 'kw-taxonomy',              'value': ['sua-pele', 'sua-pele/novidades']},
  {'field': 'link-canonical',           'value':
'http://beleza.terra.com.br/sua-pele/novidades/novo-tipo-de--horas,.....'},
  {'field': 'og-image',                 'value': 'http://p2.trrsf.com/image/fget/cf/600/600/images.terra.com/2014.....'},

  {'field': 'og-site-name',             'value': 'Terra'},
  {'field': 'og-title',               'value': 'Novo tipo de drenagem diminui 8 cm de barriga em duas horas'},
  {'field': 'og-type',                'value': 'article'},
  {'field': 'og-url',                 'value': 'http://beleza.terra.com.br/sua-pele/novidades/novo-tipo-de-hora.....'},
  {'field': 'publishtime'               'value': '2014-06-02T00:00:00.000Z'},
  {'field': 'recs-category-type'          'value': 'taxonomy'},
  {'field': 'recs-category0',           'value': 'taxonomy|sua-pele'},
  {'field': 'recs-category1',           'value': 'taxonomy|sua-pele|novidades'},
  {'field': 'recs-publishtime',          'value': '2014-06-02T00:00:00.000Z'},
  {'field': 'recs-rawtitle',            'value': 'Sua Pele: Drenagem Lipossônica Ativa reduz gordura da barriga em uma
s'},
  {'field': 'siteid',                 'value': '1133920500627333062'},
  {'field': 'teaser',                  'value': 'Desejo de praticamente toda mulher, perder aquela gordurinha .....},
  {'field': 'thumbnail',                'value': 'http://content-thumbnail.cxpublic.com/content/thumbnail200x1.....'},
  {'field': 'timestamp',                'value': '2015-08-05T20:34:52.000Z'},
  {'field': 'title',                  'value': 'Novo tipo de drenagem diminui 8 cm de barriga em duas horas'},
  {'field': 'url',                    'value': 'http://beleza.terra.com.br/sua-pele/novidades/novo-tipo-de.....'},
  {'field': 'urlhash',                  'value': 'd0e85cb9001e3f929fedcc8c2f2d64454f089239'}
 ]
}
```


検索結果の絞り込み

上記のように一つのidに含まれる検索結果は非常に膨大です。実際の結果はさらに多くのドキュメントに含まれる値を返してきます。
取得するフィールドを指定することで、結果として返却されるデータ量を減らすことができます。
これには resultFields パラメーターを使用します。

エンドユーザーに検索結果を返す時、ユーザーはデフォルトで最初の10行を見たいかもしれません。
start と count パラメータを使うことでページングが可能です。
以下はどのようにog-title と og-url の検索結果の11行目から20行目までを確認する方法です。

```
$ python cx.py /document/search '{"siteIds":["<サイトIDを指定>"], "start":11, "count":10,
"resultFields":["og-title","og-url"]}'
> python cx.py /document/search "{¥"siteIds¥":[¥"<サイトIDを指定>¥"], ¥"start¥":11, ¥"count¥":10,
¥"resultFields¥":[¥"og-title¥",¥"og-url¥"]}"
```

以下は Pythonスクリプトで同じコマンドを使っています。

```
def search(siteId, start=0, count=10, resultFields=None):
    requestObj = {"siteIds":[siteId], "start":start, "count":count, "resultFields":resultFields}
    status, response = cxApi('/document/search', requestObj)
    errorHandling(status, response, "Unable to retrieve search result")
    return response

result = search(_siteId, start=6, count=2, resultFields=['og-title', 'og-url'])
print(result)
```

上では (count=2) を利用して2つのドキュメントを取得しています。
そして、それぞれのドキュメントで取得するフィールドをog-title と og-url に絞ります (resultFields=['og-title', 'og-url'])。
さらに、最初の6ドキュメント(#0番目から#5番目)をスキップし、1行目の出力を6番目(`start=6`)からとしています。
結果、以下のように出力されます。

```
{
 'totalCount': 6317,
 'matches': [
  {
   'id': '38492b274883f91f0be6d0ac27d4e9208fc06b9e',
   'siteId': '1133920500627333062',
   'fields': [
    {'field': 'og-title', 'value': 'Veja seis dicas para afastar o tédio do relacionamento'},
    {'field': 'og-url', 'value': 'http://noticias.terra.com.br/.....'}
   ],
   'score': 1.0
  },
  {
   'id': '9f7a76cfc98d1550c43c6f31ad1219fa32d91059',
   'siteId': '1133920500627333062',
   'fields': [
    {'field': 'og-title', 'value': 'De R$ 11 a R$ 6 mil: confira os salários mínimos pelo mundo'},
    {'field': 'og-url', 'value': 'http://noticias.terra.com.br/.....'}
   ],
   'score': 1.0
  }
 ]
}
```

Searchクエリ

上記の例までは特にクエリを利用していませんでした。
クエリはフィルタの一種です。
したがってクエリがないと、全てのインデックスされたドキュメントがヒットしてしまいます。

下記の例ではクエリパラメータに"Rio de Janeiro"を設定してみます。

```
def search(siteId, query=None, start=0, count=10, resultFields=None):
    requestObj = {"siteIds":[siteId], 'query':query, "start":start, "count":count, "resultFields":resultFields}
    status, response = cxApi('/document/search', requestObj)
    errorHandling(status, response, "Unable to retrieve search result")
    return response

query = 'query("Rio de Janeiro")'
result = search(_siteId, query=query, count=1, resultFields=['og-title'])
print(json.dumps(result, indent=4, sort_keys=True))
```

結果は以下のように出力されます。
読みやすくする為結果を一つだけにしています(start=0, count=1)。
また、表示にJSONを使っています。

クエリについては、値を返す上で特有の構文があります。
今の時点で大切なことは、以下の結果でタイトルに 'Rio de Janeiro' という語句をもったものが確認できることです。

```
{
   "matches": [
      {
         "fields": [
            {
               "field": "og-title",
               "value": "A um ano dos Jogos, veja como está obras no Rio de Janeiro"
            }
         ],
         "id": "119768d074c5984f6f6489f9d9ce5505e34d31c5",
         "score": 0.836741,
         "siteId": "1133920500627333062"
      }
   ],
   "totalCount": 572
}
```

クエリ構文

旧来のCxense Searchのクエリは URLの一部を示して使われてきましたが、
ここで説明されるクエリ構文のような特有の構文は新しいSearch APIでも変わりません。

```
query(<field>^<boost>,....,<field>^<boost>:"<query content>", token-op=<token operator>
```

クエリ構文のそれぞれの項目について解説します。

- field - 上で示されていたように title、url、body などのフィールドを参照します。
  デフォルトは全てのフィールドを示す 特別なフィールド名の _all です。
- boost - 検索結果としてより重要であるフィールドを定義するために、それぞれのフィールドに重み付けの数値を定義することが出来ます。
  例えば、body よりも title の方に重みを高くしたい、というのは一般的です。boost 値は順位方式の一部で、
  特定ドキュメントに付与するランクに影響します。複数のフィールドに重み付けをした場合には、
  それぞれのフィールドで独立してブーストが行われます。デフォルトのブースト値は 1 です。
- token operator - クエリの処理方法を設定します。and 、or 、phrase の3つのオプションがあります。
  この例では 'Rio de Janeiro', という、複数のフレーズからのクエリでリクエストを行います。この時に以下の値を指定します。

  -- and (デフォルトです): すべての単語がドキュメントに存在しなければならない。
  -- or: 1つでも単語がドキュメントに存在していれば良い。

-- phrase: 全ての単語がドキュメントに存在し、なおかつ同じ順番で現れなければらない。
（※日本語では、Tokenizeを行う関係でphraseを使用しないと意図通りに検索できない場合があります）

次のスクリプトではクエリを分けて組み上げる方法を紹介しています。

```
def search(siteId, query=None, start=0, count=10, resultFields=None):
    requestObj = {"siteIds":[siteId], 'query':query, "start":start, "count":count, "resultFields":resultFields}
    status, response = cxApi('/document/search', requestObj)
    errorHandling(status, response, "Unable to retrieve search result")
    return response

def genQuery(queryTerms, tokenOp='and', fieldBoosts={'_all':1}):
    fieldBoosting = (','.join(["%s^%s" % (k, str(v)) for k, v in fieldBoosts.items()]))
    return 'query(%s:"%s", token-op=%s)' % (fieldBoosting, queryTerms, tokenOp)

query = genQuery("Rio de Janeiro", fieldBoosts={'title':5, 'body':1})
print(query)

result = search(_siteId, query=query, count=1, resultFields=['og-title'])
print(json.dumps(result, indent=4, sort_keys=True))
```

下のセクションではクエリと検索結果の両方を表記します。
ここでは、<title^5>でtitleフィールドに<body>フィールドの5倍の重み付けを行い、他のフィールドを無視して2つのフィールドを取得しています。

```
query(title^5,body^1:"Rio de Janeiro", token-op=or)
{
   "matches": [
      {
         "fields": [
            {
               "field": "og-title",
               "value": "Miss Rio de Janeiro - Rayanne Morais"
            }
         ],
         "id": "af85c72a80ea97f6bbd7835ce919ce392332f257",
         "score": 3.5711548,
         "siteId": "1133920500627333062"
      }
   ],
   "totalCount": 6604
}
```

複数のクエリの組み合わせ方

ブール演算子を利用して、複数のクエリを結合することができます。
例えば下記の例では、ユーザーは完全一致で 'Rio de Janeiro' か、もしくは 'São Paulo'を検索したいと仮定します。
これを行う方法は二つのクエリにそれぞれ(token-op=phrase)をセットして、以下のようにクエリを組み合わせてください。

```
def genQuery(queryTerms, tokenOp='or', fieldBoosts={'_all':1}):
    fieldBoosting = (','.join(["%s^%s" % (k, str(v)) for k, v in fieldBoosts.items()]))
    return 'query(%s:"%s", token-op=%s)' % (fieldBoosting, queryTerms, tokenOp)

query = genQuery("Rio de Janeiro", tokenOp='phrase') + ' or ' + genQuery("São Paulo", tokenOp='phrase')
print(query)

OUTPUT:
query(_all^1:"Rio de Janeiro", token-op=phrase) or query(_all^1:"São Paulo", token-op=phrase)
```

ランキングとソート

デフォルトでは関連性（スコア順）の並びとなっています。
これをアルファベット順や、数値、日付、緯度経度などを基準に並べ替えを行えます。

```
def search(siteId, query=None, sort=None, start=0, count=10, resultFields=None):
    requestObj = {"siteIds":[siteId], 'query':query, 'sort':sort, "start":start, "count":count, "resultFields":resultFields}
    status, response = cxApi('/document/search', requestObj)
    errorHandling(status, response, "Unable to retrieve search result")
    return response

def genQuery(queryTerms, tokenOp='or', fieldBoosts={'_all':1}):
    fieldBoosting = (','.join(["%s^%s" % (k, str(v)) for k, v in fieldBoosts.items()]))
    return 'query(%s:"%s", token-op=%s)' % (fieldBoosting, queryTerms, tokenOp)

def genSortConfig(type, order, field=None, longitude=None, latitude=None):
    sortConfig = {'type':type, 'order':order, 'field':field, 'longitude':longitude, 'latitude':latitude}
    return [{k:v for k,v in sortConfig.items() if v != None}]

query = genQuery('Rio de Janeiro', tokenOp='phrase')
sort = genSortConfig('time', 'descending', 'publishtime')
print(sort)

result = search(_siteId, query=query, sort=sort, count=1, resultFields=['og-title'])
print(json.dumps(result, indent=4, sort_keys=True))
```

"publishtime"を降順で並べ替えた結果が以下になります。

```
[{'type': 'time', 'order': 'descending', 'field': 'publishtime'}]
{
   "matches": [
      {
         "fields": [
            {
               "field": "og-title",
               "value": "Reis do vexame? Vasco e Palmeiras têm 26 vergonhas no no século"
            }
         ],
         "id": "2589db29e9b3603db75b7f5b5e97552eb7186898",
         "siteId": "1133920500627333062"
      }
   ],
   "totalCount": 2344
}
```

ハイライト表示

検索に利用した検索文字列を結果内の任意のフィールドでハイライトできます。

```
def search(siteId, query=None, sort=None, highlights=None, start=0, count=10, resultFields=None):
    requestObj = {"siteIds":[siteId], 'query':query, 'sort':sort, 'highlights':highlights, "start":start, "count":count,
"resultFields":resultFields}
    status, response = cxApi('/document/search', requestObj)
    errorHandling(status, response, "Unable to retrieve search result")
    return response

def genQuery(queryTerms, tokenOp='or', fieldBoosts={'_all':1}):
    fieldBoosting = (','.join(["%s^%s" % (k, str(v)) for k, v in fieldBoosts.items()]))
    return 'query(%s:"%s", token-op=%s)' % (fieldBoosting, queryTerms, tokenOp)

def genHighlight(field, start, stop, length=None):
    hightlight = {'field':field, 'start':start, 'stop':stop, 'length':length}
    return [{k:v for k,v in hightlight.items() if v != None}]

query = genQuery('Rio de Janeiro', tokenOp='phrase', fieldBoosts={'title':1})
highlights = genHighlight('og-title', '<em>', '</em>') + genHighlight('body', '<em>', '</em>')
print(highlights)

result = search(_siteId, query=query, highlights=highlights, count=1, resultFields=['og-title'])
print(json.dumps(result, indent=4, sort_keys=True))
```

下記の出力結果ではbodyとog-titleのフィールドで検索文字列をハイライト表示しています。

[{'field': 'og-title', 'start': '', 'stop': ''}, {'field': 'body', 'start': '', 'stop': ''}]
{
  "matches": [
    {
      "fields": [
        {
          "field": "og-title",
          "value": "Musical 'Hairspray' estreia no Rio de Janeiro"
        }
      ],
      "highlights": [
        {
          "field": "body",
          "highlight": " (10), no Rio de Janeiro, a vesão tupiniquim do trabalho. E quem assina a direção é Miguel Falabella."
        },
        {
          "field": "body",
          "highlight": "Quem quiser conferir o trabalho, deverá ir ao Oi Casa Grande, no Leblon, Rio de Janeiro, às quintas"
        },
        {
          "field": "og-title",
          "highlight": "Musical 'Hairspray' estreia no Rio de Janeiro"
        }
      ],
      "id": "568eae0bd491fa13e32dea3e5442375e3fee6585",
      "score": 5.396971,
      "siteId": "1133920500627333062"
    }
  ],
  "totalCount": 48
}

ファセット

ファセット機能は検索結果をドリルダウンするのに使います。
下記の例では検索文字列「F1」と、それに関連する人物名（F1ドライバーやチームオーナー名等が該当します。）のファセット検索を、ドキュメント内で、より多く(field 'kw-person')にある値から取得しています。

```
def search(siteId, query=None, sort=None, highlights=None, facets=None, start=0, count=10,
resultFields=None):
    requestObj = {"siteIds":[siteId], 'query':query, 'sort':sort, 'highlights':highlights, 'facets':facets, 'start':start,
'count':count, 'resultFields':resultFields}
    status, response = cxApi('/document/search', requestObj)
    errorHandling(status, response, "Unable to retrieve search result")
    return response

def genQuery(queryTerms, tokenOp='or', fieldBoosts={'_all':1}):
    fieldBoosting = (','.join(["%s^%s" % (k, str(v)) for k, v in fieldBoosts.items()]))
    return 'query(%s:"%s", token-op=%s)' % (fieldBoosting, queryTerms, tokenOp)

def genFacetConfig(type, field, count=None, min=None, latitude=None, longitude=None, unit=None,
ranges=None):
    facetConfig = {'type':type, 'field':field, 'count':count, 'min':min}
    return [{k:v for k,v in facetConfig.items() if k!= None}]

query = genQuery('Formula-1')
facetConfigs = genFacetConfig('string', 'kw-person', count=3, min=10)
print(facetConfigs)
result = search(_siteId, query=query, facets=facetConfigs, count=1, resultFields=['og-title'])
print(json.dumps(result, indent=4, sort_keys=True))
```

下記は検索結果のドキュメントに最も含まれるF1ドライバーの3つの人名が出力されています。

```
[{'count': 3, 'min': 10, 'type': 'string', 'field': 'kw-person'}]
{
   "facets": [
      {
         "buckets": [
            {
               "count": 138,
               "label": "lewis hamilton"
            },
            {
               "count": 133,
               "label": "felipe massa"
            },
            {
               "count": 115,
               "label": "sebastian vettel"
            }
         ]
      }
   ],
   "matches": [
      {
         "fields": [
            {
               "field": "og-title",
               "value": "Veja os 10 piores pilotos da Fórmula 1 na década"
            }
         ],
         "id": "6c99281e2227b652762bc2e03a2645ff0c1ad4aa",
         "score": 1.3482366,
         "siteId": "1133920500627333062"
      }
   ],
   "totalCount": 539
}
```


ファセット検索で表示された結果をユーザーが選択した場合には、新しく検索が行われますが、クエリは同じものが利用されます。(新たなフィルタリングが不要な場合を除く。)
フィルタについては次のセクションで解説します。

フィルター

フィルタとクエリは同じ機能を実行しますが、以下の三つの点で異なります。

- フィルタリングでは検索結果のランキング／ソートには影響を与えません。
- フィルタリングでは言語的な処理は行われません。
- フィルタリングは全体を検索可能です。(検索時に正規表現で用語を検索する場合と同様。)

クエリとフィルタを併用した場合、[AND]式では全てを含みます。

    (query) AND (filter)

フィルタはファセット検索との組み合わせで利用されます。

例えばロケーションでファセット検索を行うと仮定します。検索結果プレビューには"Rio de janeiro"、"São Paulo" と "Brasilia"の3都市が返って来ています。
ユーザーは"São Paulo"に関してのみ更にドリルダウン検索を行いたいと考えているとします。
以下のようにクエリとフィルタを組み合わせましょう:

(同じクエリ条件) AND (フィルターに"São Paulo")

ユーザーが他のファセットからの結果から更に詳細に検索結果を絞る場合、ファセットはフィルタに追加されます。

(同じクエリ条件) AND (地名フィルタ：São Paulo AND カテゴリフィルタ：ショッピング AND .....)

次の例は、前のセクションで出てきた検索文字列「F1」に基づいています。
まず最初に「F1」を検索し、人名のファセット条件を追加することで結果をドリルダウンさせます。
この時、133ページの検索結果を持つブラジル人ドライバーの「Felipe Massa」 を利用します。
コードサンプル：

```
def search(siteId, query=None, filter=None, sort=None, highlights=None, facets=None, start=0, count=10,
resultFields=None):
   requestObj = {"siteIds":[siteId], 'query':query, 'filter':filter, 'sort':sort, 'highlights':highlights, 'facets':facets,
'start':start, 'count':count, 'resultFields':resultFields}
   status, response = cxApi('/document/search', requestObj)
   errorHandling(status, response, "Unable to retrieve search result")
   return response

def genQuery(queryTerms, tokenOp='or', fieldBoosts={'_all':1}):
   fieldBoosting = (','.join(["%s^%s" % (k, str(v)) for k, v in fieldBoosts.items()]))
   return 'query(%s:"%s", token-op=%s)' % (fieldBoosting, queryTerms, tokenOp)

def genStringFilter(field, terms):
   terms = '"%s"' % terms if type(terms) is str else terms
   return "filter(%s:%s)" % (field, terms)

query = genQuery('Formula-1')
filter = genStringFilter('kw-person', 'felipe massa')
print(filter)
result = search(_siteId, query=query, filter=filter, count=1, resultFields=['og-title'])
print(json.dumps(result, indent=4, sort_keys=True))
```

下記の”totalCount”でわかるように、「Felipe Massa」のファセット検索の数字と一致します。

```
filter(kw-person:"felipe massa")
{
   "matches": [
      {
         "fields": [
            {
               "field": "og-title",
               "value": "Só deu ela! Lindsey Vonn 'reina' no paddock de Silverstone"
            }
         ],
         "id": "241b6547d455831c85597f1ec123d5ed01e4dcd8",
         "score": 1.1744306,
         "siteId": "1133920500627333062"
      }
   ],
   "totalCount": 133
}
```

すべてのフィールドがフィルタに使えるわけではありません。スキーマに追加された新しいフィールドはフィルタに利用できますが、大部分の古いフィルタは利用できません。
そのフィールドがフィルタに使えるかどうかは、フィルタとして設定してみてエラーが返るかどうかで簡単に判別できます。
下記のエラーメッセージ "The field body is not filter enabled" は、bodyフィールドをフィルタで利用しようとした時に発生したものです。

```
filter(body:"Sâo Paulo")
Traceback (most recent call last):
  File "C:/Users/Morten/Desktop/search3.py", line 58, in <module>
    result = search(_siteId, query, filter, sort, highlights, facets, start, count, resultFields)
  File "C:/Users/Morten/Desktop/search3.py", line 26, in search
    errorHandling(status, response, "Unable to retrieve search result")
  File "C:/Users/Morten/Desktop/search3.py", line 21, in errorHandling
    raise Exception("%s (http status = %s, error details: '%s')" % (message, status, response['error']))
Exception: Unable to retrieve search result (http status = 400, error details: 'Invalid request: Member filter
failed: The field body is not filter enabled')
```

クエリ使用情報の取得

検索結果の向上や、どのような検索ワードが利用されているか確認するために /reports/search や /reports/search/usage を用いてクエリ使用状況のレポートを見ることができます。
このAPIは検索結果にヒットしたクエリの数(検索結果1個以上にヒットしたもの) と、失敗したクエリ (検索結果が0個だったもの)を提供します。
レポートの結果、パフォーマンスを向上させるための設定の対処ができます。以下はその例です。

- 検索文字列に「ビークル」が使われた場合は 0 ヒットの結果となります。しかし、「車」の場合は複数ヒットします。==>
  「ビークル」を同義語辞書の「車」にマッピングさせることで検索結果がヒットするようになります。
- 宝石の種類などではスペルミスにより検索結果が0となることがあります。==>
  もしかして?検索を実装する事で検索結果にヒットするようになります。
- すべての検索において、一つ以上の検索結果を返すようにクエリコンプリションに用語を追加します。

次のスクリプトは特定の日付のクエリレポートを取得するものです。ここで出てくるlogQuery パラメータに注意してください。
LogQueryに挿入されたものに基づいて総数が取得されます。冒頭で紹介した"Hello World"のドキュメントを探すためのコードサンプルです。

```
def search(siteId, query=None, logQuery=None, filter=None, sort=None, highlights=None, facets=None,
start=0, count=10, resultFields=None):
    requestObj = {"siteIds":[siteId], 'query':query, 'logQuery':logQuery, 'filter':filter, 'sort':sort,
'highlights':highlights, 'facets':facets, 'start':start, 'count':count, 'resultFields':resultFields}
    status, response = cxApi('/document/search', requestObj)
    errorHandling(status, response, "Unable to retrieve search result")
    return response

def genQuery(queryTerms, tokenOp='or', fieldBoosts={'_all':1}):
    fieldBoosting = (','.join(["%s^%s" % (k, str(v)) for k, v in fieldBoosts.items()]))
    return 'query(%s:"%s", token-op=%s)' % (fieldBoosting, queryTerms, tokenOp)

# First we make a query
logQuery = 'Hello World'
query = genQuery(logQuery)
result = search(_siteId, query=query, logQuery=logQuery, count=1, resultFields=['og-title'])
print(json.dumps(result, indent=4, sort_keys=True))

# And then we check to see if it gets reported (replace date with today's date)
date = "2015-10-21"
print(searchReportByDate(_siteId, date))
print(searchUsageByDate(_siteId, date))
```

以下、検索文字列 "Hello World" がヒットしたクエリ(matching query)としてレポートに出力されました。この検索文字列がインデックスになかった場合、失敗したクエリ(zero hit query)として出力されます。

```
{
 'siteId': '1137298549016178938',
 'matchingQueries': [
  {'query': 'Hello World', 'count': 1}
 ],
 'zeroHitQueries': []
}
{
 'siteId': '1137298549016178938',
 'usage': [
  {
   'timestamp': '2015-10-21T00:00:00.000Z',
   'zeroHitQueries': 0,
   'queries': 1
  }
 ],
}
```

辞書をベースとした機能

Cxense Searchは検索機能そのものに外部の辞書機能が付随します。
これらの機能はいずれもクエリが送信される前（同義語辞書もしくはクエリコンプリション）か、検索結果が返された後（スペル補正）にコールされます。

辞書管理方法

以下のスクリプトでは、辞書の作成方法とその後アップロードする方法（新バージョン）を提示しています。
初めてスクリプトを実行した際には、以降の辞書アップロードに使用するIDが発行されます。
ファイルにはExcelもしくはJSONを利用できます。このチュートリアルではJSONを利用します。

```
_username      = "<YOUR CXENSE USER ID>"
_secret        = "<YOUR CXENSE API KEY>"
_dictionaryId   = '<THE DICTIONARY ID OR AN EMPTY STRING (ON FIRST RUN)>'
_dictionaryName = '<THE NAME OF THE DICTIONARY>'
_description    = '<A VERY SHORT DESCRIPTION OF THE DICTIONARY>'
_exelDictFile   = '<PATH OF YOUR XLSX DICTIONARY FILE OR EMPTY STRING>'
_jsonDictFile   = '<PATH OF YOUR JSON DICTIONARY FILE OR EMPTY STRING>'

import json, base64, datetime, hmac, hashlib, http.client

def cxApi(path, obj):
   date = datetime.datetime.utcnow().isoformat() + "Z"
   signature = hmac.new(_secret.encode('utf-8'), date.encode('utf-8'), digestmod=hashlib.sha256).hexdigest()
   headers = {"X-cXense-Authentication": "username=%s date=%s hmac-sha256-hex=%s" % (_username, date,
signature)}
   connection = http.client.HTTPSConnection("api.cxense.com", 443)
   connection.request("POST", path, json.dumps(obj), headers)
   response = connection.getresponse()
   status = response.status
   responseObj = json.loads(response.read().decode('utf-8'))
   connection.close()
   return status, responseObj

def errorHandling(status, response, message):
   if status != 200:
      raise Exception("%s (http status = %s, error details: '%s')" % (message, status, response['error']))

def createDictionary(siteGroupId, dictionaryName, description):
   status, response = cxApi('/processing/dictionary/create', {"siteGroupId":siteGroupId, "name":dictionaryName,
"description":description})
   errorHandling(status, response, 'Unable to create dictionary')
   return response['dictionaryId']

def uploadDictionary(dictionaryId, filename, isJson):
   if isJson:
      requestObj = {"dictionaryId":dictionaryId, "entries":json.loads(open(filename,
'rb').read().decode('utf-8-sig'))}
   else:
      requestObj = {"dictionaryId":dictionaryId, "workbook":base64.b64encode(open(filename,
'rb').read()).decode('ascii')}
   status, response = cxApi('/processing/dictionary/update', requestObj)
   errorHandling(status, response, 'Unable to upload new version of dictionary')
   if 'messages' in response and response['messages']:
      print ('¥n'.join(response['messages']))

if not _dictionaryId:
   dictionaryId = createDictionary(_siteGroupId, _dictionaryName, _description)
   print('A new dictionary entry was created with id "%s"' % dictionaryId)
else:
   isJson = True if _jsonDictFile else False
   dictFilePath = _jsonDictFile if isJson else _exelDictFile
   uploadDictionary(_dictionaryId, dictFilePath, isJson)
   print('Dictionary with id "%s" has been updated' % _dictionaryId)
```

辞書の使い方

すべての辞書は同じように動作します。検索用語を投入すると、マッチしたリストが返されます。

```
def dictionaryLookup(dictionaryId, input):
   status, response = cxApi('/processing/dictionary/search', {"dictionaryId":dictionaryId, "input":input})
   errorHandling(status, response, "Unable to do dictionary lookup")
   return response

print(dictionaryLookup(_dictionaryId, 'tv'))
```

辞書の種類に応じてクエリの向上のため、戻り値を利用します。
以下の出力結果はさらに下の同義語辞書を引っ張っています。

```
{
 'matches': [
  {'sheet': 'sds-categories', 'value': 'television', 'key': 'tv'},
  {'sheet': 'sds-categories', 'value': 'telly', 'key': 'tv'}
 ]
}
```

辞書テンプレート

提供している辞書のテンプレートは「入力文字チェック」「クエリコンプリション」「同義語辞書」の3つの機能をベースとしています。

入力文字チェック - もしかして?検索

入力文字チェック機能は「もしかして?検索」に利用されます。
この機能は新しいCxense Searchではまだ実装されていません。"didyoumean"のモードを設定しアップロードを行おうとするとエラーが返ってきます。

下記の辞書には会社名Cxenseの正しい綴りが登録されています。

```
{
   "spell-whitelist": {
      "cxense":[]
   },
   "global-properties": {
  "tokenizer-context": "en",
  "mode": "didyoumean",
  "identity-match": false,
  "inherit-offsets": false,
  "leftmost-longest-match": true,
  "allow-complete-match": true,
  "key-normalization-flags": 4,
  "value-normalization-flags": 23,
  "swap-fields": false,
  "count": 1,
  "unique-count": 1,
  "count-field": "none",
      "group-prefix": "<THE SITE GROUP PREFIX>"
   },
}
```

この設定の違いは他のspell-whitelist 辞書名によって異なります。value:cxenseに対してkeyは空となり、モードは"didyoumean"に設定されています。

クエリコンプリション

クエリコンプリションはsuggest-whitelist と suggest-blacklistの二つの辞書を用いて行われます。
Whitelistはサジェスト機能のために、後者のblacklistはサジェストに表示させたくない用語に対して利用されます。(例えば不適切な用語などに対して)
suggest-whitelist に投入するリストには、過去のクエリの結果からマッチする検索結果を利用します。
過去の検索結果のレポートを取得する方法については上項で説明しています。
一つは長時間に渡って(レポートはUTC時間に基づいた24時間ごと)
ループを繰り返し、用語を蓄積していくことができます。また、もう一方は前日からのすべての新しい用語を追加しつつ、辞書の更新版をアップロードします。

```
{
   "suggest-whitelist": {
      "cxense":""
   },
   "global-properties": {
 "tokenizer-context": "en",
 "mode": "prefix",
 "identity-match": false,
 "inherit-offsets": false,
 "leftmost-longest-match": true,
 "allow-complete-match": true,
 "key-normalization-flags": 4,
 "value-normalization-flags": 23,
 "swap-fields": false,
 "count": 1,
 "unique-count": 1,
 "count-field": "none",
      "group-prefix": "<THE SITE GROUP PREFIX>"
   }
}
```

上記と他の設定との違いは suggest-whitelist 辞書名とmode にて prefix を指定しています。

```
{
   "suggest-blacklist": {
      "damn":""
   },
   "global-properties": {
 "tokenizer-context": "en",
 "mode": "overlap",
 "identity-match": false,
 "inherit-offsets": false,
 "leftmost-longest-match": true,
 "allow-complete-match": true,
 "key-normalization-flags": 4,
 "value-normalization-flags": 23,
 "swap-fields": false,
 "count": 1,
 "unique-count": 1,
 "count-field": "none",
      "group-prefix": "<THE SITE GROUP PREFIX>"
   }
}
```

上記と他の設定との違いは suggest-blacklist 辞書名と mode にて overlap を指定しています。

同義語登録

同義語辞書はユーザーが検索窓で用語を入力した際に、追加でよく似た用語を示すためのクエリ用語そのものを追加するのに使われています。

以下の辞書の例では、ユーザーが television という語句を入力した際に tv という語句を返し、元の検索ワードと or で表示されます。

```
{
   "synonyms": {
      "tv": ["television", "telly"],
      "television": "tv"
   },
   "global-properties": {
      "identity-match": false,
      "inherit-offsets": false,
      "leftmost-longest-match": true,
      "allow-complete-match": true,
      "key-normalization-flags": 4,
      "value-normalization-flags": 23,
      "swap-fields": false,
      "count": 1,
      "unique-count": 1,
      "count-field": "none",
      "group-prefix": "サイトグループのプリフィックスを入力"
   }
}
```

他の設定との違いは、synonym 辞書名と mode または tokenizer-context パラメータの設定がないことです。
以下では同義語辞書へのオリジナル用語の追加方法を紹介しています。

```
_synonymDictionaryId = '<YOUR SYNONYM DICTIONARY ID>'

def dictionaryLookup(dictionaryId, input):
    status, response = cxApi('/processing/dictionary/search', {"dictionaryId":dictionaryId, "input":input})
    errorHandling(status, response, "Unable to do dictionary lookup")
    return ' '.join([match['value'] for match in response['matches']])

userInput = 'tv'
queryTerms = "%s %s" % (userInput, dictionaryLookup(_synonymDictionaryId, userInput))
query = genQuery(queryTerms, tokenOp='or')
print(query)

OUTPUT:
query(_all^1:"tv television telly", token-op=or)
```

Cxense Searchについて(旧バージョン)

以下は、現在も従来のCxense Searchをご利用のお客様の為のドキュメントです。
将来的にこれらのAPIは廃止される予定です。

新しいSearchのドキュメントについてはこちらをご参照ください。

Cxense Search API ガイド

> 廃止
> 従来のCxense Search API ガイド は新しいAPI /document/search に置き換えられ、廃止されます。

概要

このページは、Cxense Search API についての情報を提供いたします。

コンテンツ

デモ環境

デモ環境を http://sandbox.cxsearch.cxense.com/ で提供しております。このデモ環境へは、外部ネットワークからもアクセスが可能で、APIなどのサンプルを実行することも可能です。

また、この環境のデータは、BIRTサンプルデーターベースの一部を利用しております。

API プロトコル

Cxense Search は、データ圧縮およびメディアのタイプのようなオプションをサポートするREST形式のAPIによってアクセスできます。全てのAPIリクエストは、UTF-8でエンコードされなければいけません。

⚠ 3500文字以上のURLはサポートしておりません。

スタートガイド

初めにCxense Search - スタートガイドをご参照下さい。Cxense Searchをご利用頂く際の大まかな流れを把握することができます。また、詳細な情報は、それぞれの項目にリンクも追加しておりますので、ご利用下さい。

Cxense Search - スタートガイド

概要

Cxense Search をお使い頂く際の流れの概要を説明いたします。それぞれのより詳細な説明は、各リンク先をご参照下さい。

コンテンツ

索引スキーマの定義

Cxense Search を利用するために、初めに必要とする各フィールド情報について、スキーマ索引の定義を行います。下記の例では、索引スキーマを定義するために、HTTP POST リクエストを送信します。(curl は、Unix のコマンドラインツールです)

```
curl -XPOST "http://sandbox.cxsearch.cxense.com/api/index/birt2" -d '{
   "fields" : {
      "dateManufactured" : { "type" : "date", "format" : "yyyyMMdd'¥'T¥"HHmmss", "indexOps" : "sort" },
      "line" : { "type" : "string", "indexOps" : "facet" },
      "scale" : { "type" : "string", "indexOps" : "facet" },
      "vendor" : { "type" : "string", "indexOps" : "facet" },
      "msrp" : { "type" : "double", "indexOps" : "facet" }
   },
   "index":{
      "language":"en"
   }
}'
```

上記の例では、birt2 という名前の索引を定義して、5つのフィールドを定義して、標準の言語として英語(en)を設定しております。

- dateManufactured - date 型フィールド, expecting values to be in the format `yyyyMMddTHHmmss` (e.g. 20130313T114355) and sortable
- line - string 型フィールド, ファセットを有効化
- scale - string 型フィールド, ファセットを有効化
- vendor - string 型フィールド, ファセットを有効化

- msrp - double 型フィールド, ファセットを有効化

より詳細な情報は下記ページをご参照下さい

- Cxense Search - 索引
- Cxense Search - ファセット
- Cxense Search - 並び替えとランキング

ドキュメントのフィード

索引スキーマを定義が終了すれば、次のステップは、索引の作成となります。

Cxense Search には、ドキュメントと呼ばれる概念があり、それが非常に重要です。ドキュメントは、各索引に入っている個々のコンテンツとなります。

各ドキュメントは、ユニークなドキュメントIDを持ちます。ドキュメントIDは、整数とハッシュからなります。/content/<index>
に対して、ドキュメントのユニークIDを利用して更新などの処理を行います。.
それぞれのドキュメントの情報の取得は、<base_url>/api/content/<index>/<document id>で行えます。

例えば、検証環境において、"birt" の索引に対して、"S12_3891" のドキュメントIDを取得する際には下記のURLにアクセスします。

http://sandbox.cxsearch.cxense.com/api/content/birt/S12_3891

> Document Updates
> ドキュメントIDは、更新時にも利用されます。同じドキュメントIDで既存のドキュメントを更新する際に、自動的にドキュメントが既に存在するかどうか調べて、新規のバージョンとしてドキュメントを更新します。

1ドキュメントを追加するために、POSTを利用します。例えば、Unixのcurlコマンドを利用する場合、下記のコマンドを実行することにより、ドキュメントIDが "2007_SUBY_WRX_STI"

```
$ curl -XPOST "http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI" -d '{
   "fields" :
   {
      "name"        : "2007 Subaru WRX STi",
      "line"        : "High Performance Cars",
      "scale"       : "1:18",
      "vendor"      : "Model Cars Inc",
      "description" : "Subaru Impreza WRX STI S204 (Metallic Gray) 1:18 Scale Diecast Car Model By Autoart.
Detailed interior, exterior, engine compartment. Opening doors, trunk, and hood, mounted on plastic stand",
      "quantity"    : "1234",
      "buy price"   : "34.56",
      "msrp"        : "45.67"
   }
}'
```

ドキュメントの取得を行うためには、ドキュメントIDを指定して、GETリクエストを使用します。下記が例となります。

```
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI"
```

ドキュメントを削除するためには、ドキュメントIDを指定して、DELETEリクエストを使用します。下記が例となります。

```
$ curl -XDELETE "http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI"
```

より詳細な情報は下記ページをご参照下さい

- Cxense Search - フィードと削除

検索クエリの送信

フィードが終了したら、次に検索クエリを実行します。

ドキュメントを取得するために /api/search/<index> に対して、HTTP GET リクエストをクエリパラメーターを付与して送信します。

下記の例は、"birt" 索引に対して、"ford" がドキュメントに含まれるドキュメントを検索しています(全てのパラメーターは、UTF-8 のURLエンコードする必要があります。)。

```
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%22)"
```

birt 索引に対して、"ford"を含み、scale フィールドが、"1:18"のドキュメントを取得します。これには、lineフィールドのファセットも含まれます。

```
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%22)%20and%20filter(scale%3A%221%3A18%22)&p_f=line"
```

より詳細な情報は下記ページをご参照下さい

- Cxense Search - 検索

参考情報

柔軟なJSONフォーマット

JSONのパーサーは中括弧やダブルクォーテーションを省略することも可能です。例えば、下記のように2つのフォーマットを同等に扱うことができます。

フォーマット1:

```
k1:v1,k2:v2
```

フォーマット2:

```
{"k1" : "v1", "k2" : "v2"}
```

これらのフォーマットは、ランクモデルの定義などのリクエストのパラメーターで利用することが可能です。

レスポンス・フォーマット

メディアタイプ

全てAPIは、JSONかXMLで結果セットを返せます。標準の`"Accept":"JSON"`

JSON:

```
$ curl -XGET -H "Accept: application/json"
"http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI"
{"id":"2007_SUBY_WRX_STI","fields":{"scale":"1:18","buy price":"34.56","msrp":"45.67","vendor":"Model Cars Inc","description":"Subaru Impreza WRX STI S204 (Metallic Gray) 1:18 Scale Diecast Car Model By Autoart. Detailed interior, exterior, engine compartment. Opening doors, trunk, and hood, mounted on plastic stand","name":"2007 Subaru WRX STi","line":"High Performance Cars","quantity":"1234"}}
```

XML:

```
$ curl -XGET -H "Accept: text/xml"
"http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI"
<document>
 <fields>
  <entry>
   <string>scale</string>
   <string>1:18</string>
  </entry>
  <entry>
   <string>buy price</string>
   <string>34.56</string>
...
</document>
```

データ圧縮

下記のように、Cxense Searchが結果セットをgzipで圧縮するようにリクエストすることができます。

```
$ curl -XGET -H "Accept: text/xml" -H "Accept-Encoding: gzip" -o test.json.gz
"http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI"
```

Cxense Search - API 名前空間

API名前空間

Cxense Search
では、操作の種類毎に異なるタイプのAPIの名前空間を利用します。これで明確に操作を指定することにより、ヒューマンエラーやアプリケーションでの誤ったリクエストを削減します。

名前空間一覧:

- index:
  - 索引の作成と削除
  - 索引の更新
  - 索引の取得
- content:
  - ドキュメントのフィード
  - ドキュメントの削除
  - ドキュメントの取得
- search:
  - クエリの実行

それぞれの名前空間は、REST API の /api/ の後に追加することで利用可能です。

例:

http://sandbox.cxsearch.cxense.com/api/search/_all?p_aq=query%28%22car%22%29

Cxense Search - /api/index - 索引

概要

このページでは、索引の管理方法について記述します。複数の索引を作成可能です。また、それぞれの索引毎に異なるスキーマを利用可能です。

コンテンツ

索引スキーマの定義

ドキュメントをフィードするために、対象のデータを理解し要件を満せるような索引スキーマを定義する必要があります。

スキーマを定義するために、/api/index/のURLにPOSTします（索引がまだ無い場合、下記のコマンドを実行後に作成されます）。

```
curl -XPOST "http://sandbox.cxsearch.cxense.com/api/index/birt2" -d '{
   "fields" : {
      "dateManufactured" : { "type" : "date"},
      "line" : { "type" : "string", "indexOps" : "facet" },
      "scale" : { "type" : "string", "indexOps" : "facet" },
      "vendor" : { "type" : "string", "indexOps" : "facet" },
      "msrp" : { "type" : "double", "indexOps" : "facet" }
   }
}'
```

索引スキーマに定義されていないフィールドをドキュメントに付与してフィードすることは可能です。その場合、このフィールドが自動的に定義されることに注意してください。しかし、意図する設定を行うために、常に明確にフィールドを定義してから利用いただくことを強く推奨いたします。それは、適切なスキーマ定義がされない可能性があるからです。例えば、numeric型として利用したかったフィールドが文字型として定義されてしまうなどです。

"fields" は、索引スキーマ定義のリクエストで必須となります。下記は最低限の索引スキーマの定義となります。

```
curl -XPOST "http://sandbox.cxsearch.cxense.com/api/index/birt3" -d '{
 "fields" : {}
}'
```

索引の標準の言語設定

レマタイズやステミング、文字の標準化などのような高度な言語機能を利用するために、適切な言語を索引スキーマに定義する必要があります。方法としては、下記のように索引スキーマでlanguageパラメーターに適切な言語を設定しなければなりません。

```
curl -XPOST "http://sandbox.cxsearch.cxense.com/api/index/spanishindex" -d '{
   "fields" : {
      "title" : { "type" : "string" },
      "body" : { "type" : "string" }
   },
   "index":{
      "language":"es"
   }
}'
```

言語コードは、ISO 639-1 two letter language codes の中から1つを選択します。例えば、日本語であれば、jaとなります。

> ⓘ 索引スキーマの中に言語パラメーターを指定しない場合、言語処理として英語が利用されます。

フィールド・タイプ

cX::search では、下記のフィールド・タイプを利用可能です。

フィールド・タイプ	設定可能な値	概要
Numeric	long, integer, double, float	数値の型

Date	date	日付と時間からなる型。値は、UTCタイムゾーンとして保存されます。ISO8601形式で指定 (例: 2014-05-17T11:22:22+0000)
String	string	テキストの文字列の型
Geospatial	geo_point	緯度・経度を示す型

索引オペレーション

フィールド毎にオペレーションを定義することができます。これらは、`indexOps` 属性で定義されます。また、複数のオペレーションをカンマ区切りで定義することもできます。

Operation	Description
facet	結果セット中の一意のフィールドの値を動的にカウントする機能を有効にします。他のプラットフォームでは、ディープナビゲーターとも言われています。cX::search のファセット機能は、#Facets をご参照ください。
storeOnly	設定したフィールドに対して索引を作成を無効化します。結果的に、索引容量の削減と検索パフォーマンスを高めることができます。この設定は、検索対象の項目として利用しないが、検索結果として値は取得したい場合に利用します。
sort	設定したフィールドに対してソート機能を有効化します。ソートフィールドで複数値をフィードした場合は、このフィールドでは、最も低い値が利用されることに注意して下さい。

結果セットオペレーション

結果セットに対する機能を有効化するようにスキーマを定義することができます。それは、resultCfg 属性として下記が利用可能です。

Value	Description
hl	フィールドのハイライトを有効化

既存の索引の変更

下記の索引スキーマの変更に関しては、再フィードや索引の切り替えが必要ありません。

- 新規フィールドの追加
- フィールドの削除。ただし、既存のドキュメントが再フィードされるか削除されるまで、そのフィールドでの検索にヒットすることに注意して下さい。

下記の索引スキーマの変更に関しては、再フィードや索引の切り替えが必要です。新しく索引を作成する必要があります。

- float型からlong型のような互換性のないフィールド・タイプの変更

システムの稼働

ドキュメントの追加、削除、更新を行う際には、ダウンタイムは発生しません。また、変更はAPI実施後すぐ反映されます。

互換性がない設定変更時には、別の索引を用意し、その索引に対してフィードを行った後、切り替えを行う必要があります。#Aliases を利用することによりシームレスなマイグレーションを行うことができます。

別名（エイリアス）

索引に対して、複数の別名を付与することができます。これを利用し、フロントエンドのアプリケーションで、直接索引名を指定する代わりに別名を指定することができます。それで、フロントエンドアプリケーションの設定の変更をしなくても、対象の索引を切り替えることができます。

別名を作成するために、別名をパラメーターとして索引のURLに対してPOSTします。

```
curl -XPOST "http://sandbox.cxsearch.cxense.com/api/index/birt/_aliases?add=myalias"
```

("add=alias1,alias2" のようにカンマで区切り、複数の別名を追加することができます。)

別名が正常に作成されたかどうか、下記のような索引の構成の取得により確認することができます。

```
curl -XGET "http://sandbox.cxsearch.cxense.com/api/index/birt"
```

別名の情報は、下記のようにJSON（もしくはXML）の情報として取得できます。


```
{
 "documentCount":105,
 "operationCount":0,
 "name":"birt",
 "configuration":{
 "index":{
 "number_of_replicas":0,
 "number_of_shards":1
 },
 "fields":{
 "scale":{
 "indexOps":"facet",
 "type":"string"
 },
 "msrp":{
 "indexOps":"facet",
 "type":"double"
 },
 "vendor":{
 "indexOps":"sort,facet",
 "resultCfg":"hl",
 "type":"string"
 },
 "line":{
 "indexOps":"facet,sort",
 "type":"string"
 }
 }
 },
 "aliases":["myalias"]
}
```


既存の別名を削除するために、削除したい別名をパラメーターとして索引URLに対してPOSTします。

```
curl -XPOST "http://sandbox.cxsearch.cxense.com/api/index/birt/_aliases?rem=myalias"
```

1つの同じ名前の別名を複数の索引に対して持つことができます。その場合、その別名に対するクエリは、全ての対象の索引に対して実行されます。

1つの別名が複数の索引を参照している場合には、それらの索引が同じ、または類似の索引スキーマを持つべきです。

索引の削除

索引を削除するためには、索引名に対して、下記のようにDELETEを実行します。

```
$ curl -XDELETE http://sandbox.cxsearch.cxense.com/api/index/myindexname
```

索引定義の取得

索引定義の取得を行うために、索引名に対して、下記のようにGETを実行します。

```
$ curl -XGET http://sandbox.cxsearch.cxense.com/api/index/myindexname
```

全ての索引定義を取得するためには、_all を指定します。

```
$ curl -XGET http://sandbox.cxsearch.cxense.com/api/index/_all
```

Cxense Search - /api/content - フィードと削除

概要

コンテンツは、ドキュメントとしてCxense Search
で索引付けされます。それぞれのドキュメントは、任意のフィールドを保持します。これらのフィールドは、索引スキーマで定義されます。このページでは、コンテンツを管理するためのAPIについて説明いたします。

コンテンツ

コンテンツは、ドキュメントとしてcX::searchで索引づけされます。それぞれのドキュメントは任意のフィールドを持ち、索引スキーマでフィールドタイプやファセットなどを定義することができます。このセクションでは、コンテンツを管理するためのAPIについて説明します。

フォーマット

フィードするフォーマットは、JSONかXStreamによってシリアライズされたXMLとなります。

フォーマットを下記のチェックによって判別します。

1. リクエストのメディアタイプ (`application/json` もしくは `text/xml`)
2. 送信データの1文字目 (JSONの場合は{ もしくは XMLの場合は<)

1つのドキュメントをフィード

1ドキュメントを追加するために、POSTを利用します。例えば、Unixのcurlコマンドを利用する場合、下記のコマンドを実行することにより、ドキュメントIDが "2007_SUBY_WRX_STI"

```
$ curl -XPOST "http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI" -d '{
   "fields" :
   {
      "name"        : "2007 Subaru WRX STi",
      "line"        : "High Performance Cars",
      "scale"       : "1:18",
      "vendor"      : "Model Cars Inc",
      "description" : "Subaru Impreza WRX STI S204 (Metallic Gray) 1:18 Scale Diecast Car Model By Autoart.
Detailed interior, exterior, engine compartment. Opening doors, trunk, and hood, mounted on plastic stand",
      "quantity"    : "1234",
      "buy price"   : "34.56",
      "msrp"        : "45.67"
   }
}'
```

Document Updates
ドキュメントIDは更新時に利用します。同じドキュメントIDで既存のドキュメントを更新する際に、自動的にドキュメントが既に存在するかどうか調べて、新規のバージョンとしてドキュメントを更新します。

複数ドキュメントの一括フィード

JSONオブジェクトの属性としてドキュメントIDを含めて、"_batch"に対してPOSTリクエストを実行することにより、ドキュメントを一括でフィードすることができます。ドキュメントは、1行あたり1ドキュメントを記述します。

例:

```
$ curl -XPOST "http://sandbox.cxsearch.cxense.com/api/content/birt/_batch" -d
'{"id":"2007_SUBY_WRX_STI","fields":{"name":"2007 Subaru WRX STi","line":"High Performance
Cars","scale":"1:18","vendor":"Model Cars Inc","description":"Subaru Impreza WRX STI S204 (Metallic Gray) 1:18
Scale Diecast Car Model By Autoart. Detailed interior, exterior, engine compartment. Opening doors, trunk, and
hood, mounted on plastic stand","quantity":"1234","buy price":"34.56","msrp":"45.67"}}
{"id":"2006_SUBY_FOZ_XT","fields":{"name":"2006 Subaru Forester XT","line":"High Performance
SUVs","scale":"1:18","vendor":"Model Cars Inc","description":"Subaru Forester XT (Gold) 1:18 Scale Diecast Car
Model By Autoart. Detailed interior, exterior, engine compartment. Opening doors, trunk, and hood, mounted
on plastic stand","quantity":"1234","buy price":"34.56","msrp":"45.67"}}'
```

gzipで圧縮したファイルをフィードすることもできます。

```
$ curl -XPOST "http://sandbox.cxsearch.cxense.com/api/content/birt/_batch" -T <file>.gz -H
'Content-Encoding: gzip'
```

圧縮したファイルは、上記の一括処理の例と同様に1行に1ドキュメント記述します。

一括処理で失敗したドキュメントは、レスポンスを確認することでわかります。また、それらのドキュメントは、索引付けされません。その一括処理で成功した他のドキュメントは、索引づけされます。

ドキュメントの取得

ドキュメントの取得を行うためには、ドキュメントIDを指定して、GETリクエストを使用します。下記が例となります。

```
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI"
```

ドキュメントの削除

ドキュメントを削除するためには、ドキュメントIDを指定して、DELETEリクエストを使用します。下記が例となります。

```
$ curl -XDELETE "http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI"
```

ドキュメントの一括もサポートしております。1行に1ドキュメントのドキュメントIDを含んだJSONオブジェクトで、"_delete"に対してPOSTします。

```
$ curl -XPOST "http://sandbox.cxsearch.cxense.com/api/content/birt/_delete" -d '{ "id" :
"2007_SUBY_WRX_STI"}
{ "id" : "2006_SUBY_FOZ_XT"}'
```

⚠ パフォーマンスの関係上、削除処理は、ドキュメントが存在しなくても200番のレスポンスを返すことに気をつけて下さい。

位置情報を持つコンテンツ

Cxense Search
は、緯度経度のようなデータを使ってフィルターしたり、geo_pointフィールドを利用して、距離によってランキングすることもサポートしています。

これらの機能を利用するには、下記の3つのフィールドが必須です。

- `geo_location` (フィールド・タイプ: geo_point)
- `latitude` (フィールド・タイプ: double)
- `longitude` (フィールド・タイプ: double)

以下のような索引スキーマを定義することでこの機能を利用することが可能です。この例では、countriesという索引に定義を行なっております。

```
$ curl -XPOST http://sandbox.cxsearch.cxense.com/api/index/countries -d '{
   "fields": {
      "geo_location": { "type": "geo_point" },
      "latitude": { "type": "double" },
      "longitude": { "type": "double" },
      "country": { "type": "string", "indexOps": "facet" }
   }
}'
```

位置情報の詳細な情報はCxense Search - 緯度経度での検索(英語)をご参照下さい。

複数の値を持つフィールド

cX::searchのフィールドは複数の値を保持することができます。例えば、ECサイト上である服のサイズを定義する際に、それは、SやMやLなどの複数のサイズがある場合に利用できます。定義する際には、XMLを利用する際にはXStreamのコレクションを定義するかJSONを利用する場合には、JSONの配列を利用します。

下記は、Subaruが複数のscaleの値を持つ場合の例です。

```
$ curl -XPOST "http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI" -d '{
   "fields" :
   {
      "name"        : "2007 Subaru WRX STi",
      "line"        : "High Performance Cars",
      "scale"       : ["1:18", "1:25"],
      "vendor"      : "Model Cars Inc",
      "description" : "Subaru Impreza WRX STI S204 (Metallic Gray) 1:18 Scale Diecast Car Model By Autoart.
Detailed interior, exterior, engine compartment. Opening doors, trunk, and hood, mounted on plastic stand",
      "quantity"    : "1234",
      "buy price"   : "34.56",
      "msrp"        : "45.67"
   }
}'
```

ドキュメントは、複数の値を持つフィールド内のどの値と適合してもヒットします。その場合、AND条件ではなくOR条件として適合します。

Cxense Search - /api/search - HTTP API

> 従来のCxense Search - /api/search - HTTP API は新しいAPI /document/search に置き換えられ、廃止されます。
>
> 2016年2月以降に作成されたサイトに対してはこのAPIは有効化されません。

概要

本ページでは、Cxense Search に検索クエリを実行する際の利用可能なパラメータについて説明します。

検索 HTTP API クエリパラメータ

検索クエリにてサポートしているパラメータは下記の表のとおりです。

Cxense Searchには2つの環境があります。

基本（サイトサーチ）：Insightタグを挿入し、データを収集したページを検索対象とする。索引は事前定義されたものを利用します。

カスタム（カスタムクラスター）：お客様毎に自由に索引を定義し、APIにてデータをフィードいただけます。

パラメーター	利用可能な環境	概要	標準	例
`p_aq`	基本 & カスタム	検索を行う際のキーワードを指定するパラメーターです。詳細は、Cxense Search - p_aq - 高度なクエリ言語をご参照ください。 参考：Cxense Search - p_aq - 日付のフィルター	n/a	`p_aq=query("ford")` (searches across all fields) `p_aq=query(name:"ford")` (searches the name field) `p_aq=query(category:"plumber") and query(location:"melbourne")` (searches the `category` field for plumber and the `location` field for melbourne)
`p_sm`	基本 & カスタム	並び替えを行う際に指定するパラメーターです。 参考： • Cxense Search - p_sm - 並び替えとランキング • Cxense Search - p_sm - 緯度経度での検索	n/a (will default to relevance)	`p_sm=title:asc` (orders by the title field in ascending order)
`p_rs`	基本 & カスタム	結果セットの指定を行うパラメーターです。ヒットした結果の文字列のハイライトを行うこともできます。 参考: Cxense Search - p_rs - 結果セットの設定	n/a	`p_rs=hl:{description:{p:PRE,s:SUF}}`

p_s	基本 & カスタム	結果セットの開始位置を指定するパラメーターです。 参考:Cxense Search - p_c / p_s - ページ操作 Maximum Value (p_c + p_s) <= 10000	0	p_s=10 (starts from 11th result)
p_c	基本 & カスタム	返される結果の数を指定するパラメーターです。 参考:Cxense Search - p_c / p_s - ページ操作 Maximum Value p_c=2000	10	p_c=20 (20 results)
p_f	基本 & カスタム	ファセットデータを結果に含めるように指定するパラメーターです。複数指定する場合には、カンマでファセット名を区切ります。 参考:Cxense Search - p_f - ファセット	n/a	p_f=msrp,scale (return facets on the msrp and scale fields)
p_dr	基本 & カスタム	重複排除を行うフィールドを指定するパラメーターです。 参考:Cxense Search - p_dr - 重複排除	n/a	p_dr=title (remove duplicates for items having the same title)
p_uq	基本 & カスタム	ユーザーによって指定されたクエリを指定するパラメーターです。これによって、アプリケーションによって追加されたフィルターなどをクエリログやレポートから除外することができます。	n/a	p_aq=query("ford") and filter(msrp>0)&p_uq=ford この場合、p_uqで指定されている”ford"が、レポートやクエリログで利用されます。
rank	カスタムのみ	事前定義されたスコア設定を指定するパラメーターです。 これを利用したい場合には、support@piano.ioにご連絡をください。	n/a	rank=publishfreshhigh

すべてのパラメーターは、UTF-8でエンコードされたURLでアクセスします。

例:(エンコード前)

```
http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query("ford") and filter(msrp>0)
```

アプリケーションでは、下記を実行する必要があります。

```
http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%22)%20and%20filter(msrp%3E0)
```

Cxense Search - p_aq - 高度なクエリ言語

> 廃止
> 従来のCxense Search - p_aq - 高度なクエリ言語 は新しいAPI /document/search に置き換えられ、廃止されます。

概要

このページでは、検索リクエストを実行するとき利用可能なパラメーターについて説明します。

コンテンツ

検索APIクエリパラメーター

下記が検索APIで利用可能なパラメーターです。

パラメーター	概要	デフォルト値	例
`p_aq`	検索対象のキーワードなどを指定 フィールドを特定するプリフィックスをオプション	なし	`p_aq=Ford` (全てのフィールドに対して検索) `p_aq=name:Ford` (nameフィールドに対して検索) p_aq=query(category:"plumber") and query(location:"melbourne") (categoryフィールドとlocationフィールドにそれぞれのキーワードで検索)
`p_sm`	結果のランキングの方法を指定 詳細は、Cxense Search - 並び替えとランキングを参照	なし（スコアが適用される）	`p_sm=title:asc` (タイトルフィールドで昇順にソート) p_sm=_score:desc (スコアの降順にソート) p_sm=[{year:desc},{code:asc}] (yearで降順ソート。ただし、yearが同じ値の場合には、codeの昇順でソート)
`p_rcs`	MVEL 式で計算されたカスタムスコア 詳細は、Cxense Search - 並び替えとランキングを参照	なし	
`p_rs`	結果セットの追加属性をJSONオブジェクトで定義 詳細は、Cxense Search - 結果セットの設定を参照	なし	`p_rs=hl:{description:{p:PRE,s:SUF}}`

p_s	結果セットのスタート位置 詳細は、Cxense Search - ページ操作 を参照 Maximum Value このパラメータの最大値は、p_s=25000	0	p_s=10 (11番目の検索結果から返す)
p_c	結果セットに含まれるドキュメント数 詳細は、Cxense Search - ページ操作 を参照 Maximum Value このパラメータの最大値は、p_c=2000		p_c=20 (20個の検索結果を返す)
p_f	結果セットに含まれるファセット。複数ある場合にはカンマで区切ります 詳細は、Cxense Search - ファセットを参照	なし	p_f=msrp,scale ("msrp" と "scale" フィールドのファセットを返す)
p_dr	同一フィールドの重複を行います 詳細は、Cxense Search - 重複排除を参照	なし	p_dr=title (同じタイトルの重複排除を行う)
p_uq	アプリケーション側で付与したフィルターなどを外したユーザーが入力した実際のクエリを設定します。ここで設定したパラメーターは、クエリーログやレポートなどで利用されます。	なし	p_aq=query("ford") and filter(msrp>0)&p_uq=ford (このクエリは、"ford" で検索して、msrpが0より大きいものという制約を付与しています。ただし、p_uqが "ford"なので、実際にユーザーが入力したクエリは、"ford"のみということになります)

全てのパラメーターは、UTF-8 のURLエンコードする必要があります。

例:

```
http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query("ford") and filter(msrp>0)
```

上記のクエリを実行するには下記のクエリをリクエストします。

```
http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%22)%20and%20filter(msrp%3E0)
```

検索対象の定義

主に下記の2つの対象があります。

- どの索引に対して?
- どのフィールドに対して?

任意の索引に対するクエリ

任意の索引に対してクエリは、基本のURLに "/api/search/" を付与し、カンマで区切られた索引名を指定します。

下記のように、"_all"を索引名に指定することで、全ての索引に対してクエリを実行します。

```
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/search/_all?p_aq=query(%22ford%22)"
```

下記の例は、"birt" 索引に対して、"ford" がドキュメントに含まれるドキュメントを検索しています。

```
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%22)"
```

下記の例は、"birt"と"birt2"の2つの索引に対して"ford"が含まれる結果を得るためのクエリを実行します。

```
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/search/birt,birt2?p_aq=query(%22ford%22)"
```

任意のフィールドに対するクエリ

任意のフィールドに対して検索をするために、フィールド名とクエリ文字列でフィルターしたい値を追加します。例えば、下記は、birt
索引に対して、nameフィールドが"mustang" を含むクエリです。

```
# エンコード前のクエリパラメータ: query(name:"mustang")
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(name%3A%22mustang%22)"
```

複数のフィールドに対して実行する場合には、, (カンマ)を利用します。

```
# エンコード前のクエリパラメータ: query(name,description:"mustang")
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(name%2Cdescription%3A%22mustang%22)"
```

p_aqパラメータに関しては、より多くのオプションがあります。それについては、次のセクションで説明します。

高度なクエリ言語

クエリ言語の主な利点の一つは、複雑な検索の要望に対しても対応ができる柔軟性があるということです。Cxense Search
のクエリ言語を理解するために、下記の項目を説明いたします。

- 単一のキーワードかフレーズかの指定
- 各フィールドの文字列でのフィルタリング
- 数字や日付データの範囲指定によるフィルタリング
- ランキングに対して、フィールドを指定した重み付け
- 上記のネストされた複雑な組合せ

⚠ Handling special characters in user queries
p_aqパラメーターにおいて、大抵の特殊な文字は、アルファベットの文字として使用されます。ただし、下記の表の文字は除きます。したがって、検索アプリケーションを作成する場合には、下記の文字列は特別な文字列として扱う必要があります。

Character	Description

\	バックスラッシュは、下記の文字をエスケープ文字として扱うために利用されます。もし、クエリの内でバックスラッシュを利用したい場合には、\\
*	ワイルドカードは”どのキーワードでも”ということを意味します。もし、*を検索したい場合には、¥* （バックスラッシュを前に追加）で検索をしてください。
"	ダブルクォーテーションで囲んで、検索キーワードを指定します（例："ford")。したがって、検索キーワードの中にダブルクォーテーションが含まれると無効な文法となります。もし、"を検索したい場合には、¥" （バックスラッシュを前に追加）で検索をしてください。

検索演算子

演算子の種類	演算子	概要	例
クエリ演算子	query	これは、ユーザーによって入力されたキーワードでドキュメントを検索するための主要な検索演算子です。	query("ford") query(name:"mustang")
フィルター演算子	filter	フィルター演算子ではランキングスコアの計算を行わず、結果の絞り込みを行いたい際に利用されます。 ファセットの絞込時にも、フィルター演算子を利用し絞り込みを行います。	filter(published=1) filter(department:"Books")
論理型演算子	and, or, not	複雑なクエリを利用する際に、複数の条件を組み合わせるために、論理型演算子を利用する事が可能です。	query("ford") and filter(msrp>0)

クエリ演算子

クエリ演算子を使用することにより、複数の検索ワードをどのように組み合わせて検索するかを指定することができます。また、このクエリ演算子にて、対象のフィールドや重み付けなども行うことが可能です。

クエリ演算子は、下記のフォーマットとなります。

```
query(<対象フィールド名>^<重み付け>:"<検索キーワード>", token-op=<トークン演算子>)
```

下記の表で、上記のパラメーターの説明をします。

項目名	標準値	概要	例
対象フィールド名	_all	どのフィールドに対して検索を実施したいか? _all は、全てのフィールドを対象とします。カンマを使用すると複数のフィールドの指定ができます。	query("ford") query(title:"ford") query(_all,body,description:"ford")

重み付け	1	指定したフィールドにヒットした場合の重み付けを指定します。重み付けを付与することにより、そのフィールドにヒットし場合により上位に表示したり、通常よりも下位に表示することも可能です。上記で複数フィールドを指定した場合も利用可能です。詳細は、Cxense Search - 並び替えとランキングをご参照下さい。	query(name^2:"mustang") query(title^5,description^2:"mustang")
検索キーワード	なし	1つ以上のキーワードを指定します。複数のキーワードを指定した場合、標準のトークンオペレーターでは、全てのキーワード含むAND条件として処理されます。*(アスタリスク)をキーワードとして、利用した場合には、全コンテンツを返します。	query("market research") query("*")
トークン演算子	and	"and","or", または "phrase" の中から選択します。 and (標準): ドキュメント中に全てのキーワードを含む場合 or: ドキュメント中に最低1つ以上のキーワードを含む場合 phrase: ドキュメント中に全てのキーワードを含む場合で、かつ、キーワードが同じ順番である場合	query("classic dark green",token-op=phrase)

例1：name または、 description フィールドに"1940s"
が含まれるものを取得します。また、nameフィールドでヒットしたドキュメントのランキングを高く設定します。

```
# エンコード前のクエリパラメーター: query(description,name^2:"1940s")
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(description%2Cname%5E2%3A%221940s%22)"
```

例2："ford" と "truck" をAND条件にて検索します。

```
# エンコード前のクエリパラメーター: query("ford truck",token-op=and)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%20truck%22%2Ctoken-op%3Dand)"
```

例3：nameフィールドに "truck" か "pickup" が含まれるドキュメントを取得します。

```
# エンコード前のクエリパラメーター: query(name:"truck pickup",token-op=or)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(name%3A%22truck%20pickup%22%2Ctoken-op%3Dor)"
```

例4："classic dark green" のフレーズが含まれるドキュメントを取得します。

```
# エンコード前のクエリパラメーター: query("classic dark green",token-op=phrase)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22classic%20dark%20green%22%2Ctoken-op%3Dphrase)"
```

上記のクエリは、フレーズを利用しているので、キーワードが順番どおりになっているかの判断も行なっております。上記クエリでは、結果が取得できますが、下記のように "dark green classic" のフレーズ検索をすると結果は0件となります。

```
# エンコード前のクエリパラメーター: query("dark green classic",token-op=phrase)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22dark%20green%20classic%22%2Ctoken-op%3Dphrase)"
```

フィルター演算子

フィルターは、クエリと類似したフォーマットを利用します。しかし、フィルターは検索結果の絞り込みはクエリと同じようにしますが、その検索結果の重み付けは行いません。フィルターとクエリはANDオペレーターで組み合わせられます。複数のフィルターを論理式を用いて複雑に指定することも可能です。
文字列フィルター

文字列フィルターは下記のフォーマットとなります。

```
filter(<対象フィールド名>:"<値>")
```

例：scale フィールドが、"1:18"のドキュメントを取得します

```
# エンコード前のクエリパラメーター: filter(scale:"1:18")
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=filter(scale%3A%221%3A18%22)"
```

Working with term filters on string fields
string 型フィールドに文字列フィルターを使用する際には、下記の注意が必要です。

- string型フィールドで文字列フィルターを利用するためには、索引スキーマのindexOpsでfacetを指定する必要があります。
- 指定した値は大文字小文字の標準化などの各種言語処理を行いません。
    - 言語解析(文字列の標準化やトークナイズなど)は行われませんので、索引に含まれるデータと大文字・小文字など含め完全に一致する必要があります。"Classic Car"が索引づけされているのであれば、filter(type:"classic car")
    ではなく、filter(type:"Classic Car")と記述しなければなりません。

例として、line フィールドのファセット用の索引スキーマの指定方法は下記のとおりです。

```
curl -XPOST "http://sandbox.cxsearch.cxense.com/api/index/birt2" -d '{
   "fields" : {
...lines omitted for brevity...
      "line" : { "type" : "string", "indexOps" : "facet" },
      "scale" : { "type" : "string", "indexOps" : "facet" },
...lines omitted for brevity...
   },
   "index":{
      "language":"en"
   }
}'
```

例1：line フィールドが、"Classic Cars" の結果を取得します。

```
# エンコード前のクエリパラメーター: filter(line:"Classic Cars")
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=filter(line%3A%22Classic%20Cars%22)"
```

例2：line フィールドが、"classic cars"
の結果を取得します。しかし、この場合、全て小文字で索引に含まれるデータとは完全に一致しないため結果は0件となります。

```
# エンコード前のクエリパラメーター: filter(line:"classic cars")
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=filter(line%3A%22classic%20cars%22)"
```

範囲フィルター

フィールドのタイプが、"numeric"か"date"の場合に、下記の2種類の範囲指定のフィルターを利用することができます。

- 開値域フィルター
- 閉値域フィルター

開値域フィルター

開値域フィルターは下記のフォーマットとなります。

```
filter(<対象フィールド名><オペレーター><値>)
```

オペレーターでは、下記の演算子を利用できます。

オペレーター	概要
>	より大きい
>=	以上
<	より小さい
<=	以下

下記は、msrpが100より大きいドキュメントを取得するためのクエリです。

```
# エンコード前のクエリパラメーター: filter(msrp>100)
$ curl -XGET "http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=filter(msrp%3E100)"
```

閉値域フィルター

閉値域フィルターは下記のフォーマットとなります。

```
filter(<対象フィールド名>:range<左式のオペレーター><値1>,<値2><右式のオペレーター>)
```

オペレーターでは、下記の演算子を利用できます。

オペレーター	概要
(	より大きい
[	以上
)	より小さい
]	以下

下記は、msrpが50.31以上で、60.54より小さいドキュメントを取得するためのクエリです。

```
# エンコード前のクエリパラメーター: filter(msrp:range[50.31,60.54))
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=filter(msrp%3Arange%5B50.31%2C60.54))"
```

論理演算子

論理演算子は、複雑な検索リクエストにおいて、フィルターやクエリを組合せるために利用されます。

オペレーター	概要
AND	全てのキーワードが必須となります。
OR	1つ以上のキーワードがあれば、ヒットになります。
NOT	次の演算子がマッチしない場合、ヒットになります。

例1:"ford" が含まれかつ `msrp 0`

```
# エンコード前のクエリパラメーター: query("ford") and filter(msrp>0)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%22)%20and%20filter(msrp%3E0
)&p_uq=ford"
```

例2:"delicate" が含まれかつ lineフィールドが "Classic Cars" か msrpが100より大きいドキュメントを取得します

```
# エンコード前のクエリパラメーター: query("delicate") and filter(line:"Classic Cars") or filter(msrp>100)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22delicate%22)%20and%20filter(line%3
A%22Classic%20Cars%22)%20or%20filter(msrp%3E100)&p_uq=delicate"
```

例3:"delicate" が含まれかつ msrpが0以外のドキュメントを取得します。

```
# エンコード前のクエリパラメーター: query("delicate") and not filter(msrp=0)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22delicate%22)%20and%20not%20filter
(msrp%3D0)&p_uq=delicate"
```

例4:"delicate" が含まれるが、"green" を含まないドキュメントを取得します。

```
#エンコード前のクエリパラメーター: query("delicate") and not query("green")
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22delicate%22)%20and%20not%20query(%22green%22)&p_uq=delicate"
```

例5:"pizza" をtitleフィールドに含むかつ、`lastupdate` "2013-02-16" 以降 "2013-02-18" より前のドキュメントを取得します。

```
# エンコード前のクエリパラメーター: query(title:"pizza") and
filter(lastupdate:range["2013-02-16T00:00:00.000Z","2013-02-18T00:00:00.000Z"))
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/listings?p_aq=query(title%3A%22pizza%22)%20and%20filter(lastupdate%3Arange%5B%222013-02-16T00%3A00%3A00.000Z%22%2C%222013-02-18T00%3A00%3A00.000Z%22))&p_uq=pizza"
```

ユーザーが直接入力したクエリをp_uqパラメーターに含めることを推奨いたします。

Cxense Search - p_aq - 日付のフィルター

廃止
従来のCxense Search - p_aq - 日付のフィルター は新しいAPI /document/search に置き換えられ、廃止されます。

概要

ある事前に決められた日付範囲で検索結果を絞り込みたい場合にこのページをご参照下さい。

利用方法

下記のような日付のフィルターを定義するとします。

- 本日
- 今週
- 今月
- 今年
- 昨年以前 (オプション)

上記のように特定の期間でフィルターを利用する利点として、検索の範囲を狭めることができます。これにより、検索リクエストのパフォーマンスもあげることができます。

例えば、下記の索引スキーマについて考えてみます。

```
curl -XPOST "http://sandbox.cxsearch.cxense.com/api/index/sampledateintervalindex" -d '{
   "fields" : {
      "title" : { "type" : "string" },
      "body"  : { "type" : "string" },
      "publicationdate" : { "type" : "date"},
   },
   "index":{
      "language":"ja"
   }
}'
```

検索アプリケーションで、事前定義された日付のフィルターを追加したい場合には、本日を2013-02-20とすると下記のように設定を行います。

索引スキーマで日付フォーマットを定義している場合、その日付フォーマットにしたがって日付を記述して下さい。

例:

日付フォーマット:　<format>yyyy/MM/dd HH:mm:ss</format>

日付条件の記述:　"2013/02/20 00:00:00"

本日

- `2013-02-20 <= publicationdate <` 2013-02-21
- フィルターのリクエスト: filter(publicationdate:range["2013-02-20T00:00:00.000Z","2013-02-21T00:00:00.000Z"))
- フィルターのリクエストを変更する間隔: 24 時間

今週

- `2013-02-17 () <= publicationdate <` 2013-02-24 (次の日曜日).
- フィルターのリクエスト: filter(publicationdate:range["2013-02-17T00:00:00.000Z","2013-02-24T00:00:00.000Z"))
- フィルターのリクエストを変更する間隔: 7日間

今月

- 2013-02-01 (月の始め) `<= publicationdate <` 2013-03-01 (次月の始め).
- フィルータのリクエスト: filter(publicationdate:range["2013-02-01T00:00:00.000Z","2013-03-01T00:00:00.000Z"))
- フィルターのリクエストを変更する間隔: 1ヶ月

今年

- 2013-01-01 (今年の1/1) <= publicationdate < 2014-01-01 (次年の1月1日).
- フィルータのリクエスト: filter(publicationdate:range["2013-01-01T00:00:00.000Z","2014-01-01T00:00:00.000Z"))
- フィルターのリクエストを変更する間隔: 1 年

昨年以前 (オプション)

- `publicationdate <` 2013-01-01 (前年の1月1日).
- フィルターのリクエスト: filter(publicationdate<"2013-01-01T00:00:00.000Z")
- フィルターのリクエストを変更する間隔: 1 年

この場合、ユーザが上記の間隔でフィルターするなら、下記のようなクエリを実行します。(p_aqパラメータは、URLエンコードされております。)

```
#本日
curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/sampledateintervalindex?p_aq=query(%22sample%22)%20
and%20filter(publicationdate%3Arange%5B%222013-02-20T00%3A00%3A00.000Z%22%2C%222013-02-21
T00%3A00%3A00.000Z%22))"

#今週
curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/sampledateintervalindex?p_aq=query(%22sample%22)%20
and%20filter(publicationdate%3Arange%5B%222013-02-17T00%3A00%3A00.000Z%22%2C%222013-02-24
T00%3A00%3A00.000Z%22))"

#今月
curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/sampledateintervalindex?p_aq=query(%22sample%22)%20
and%20filter(publicationdate%3Arange%5B%222013-02-01T00%3A00%3A00.000Z%22%2C%222013-03-01
T00%3A00%3A00.000Z%22))"

#今年
curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/sampledateintervalindex?p_aq=query(%22sample%22)%20
and%20filter(publicationdate%3Arange%5B%222013-01-01T00%3A00%3A00.000Z%22%2C%222014-01-01
T00%3A00%3A00.000Z%22))"

#昨年以前
curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/sampledateintervalindex?p_aq=query(%22sample%22)%20
and%20filter(publicationdate%3C%222013-01-01T00%3A00%3A00.000Z%22)"
```

Cxense Search - p_uq - ユーザーが検索したキーワードを認識

ユーザーが検索したキーワードを認識

高度なクエリを利用する上で、非常に重要なトピックの1つとして、ユーザーによって入力されたキーワードを明確にシステムに認識させることがあります。特に、これは、クエリーログからデータを生成する場合に重要となります。例えば、オートコンプリーションなどのために、クエリログからクエリの候補を生成する場合について考えてみます。この際に、ユーザーが入力したクエリ以外にも、アプリケーションが自動的に付与したクエリ式をCxense Search にリクエストした場合は、これらのクエリも一緒にクエリの候補となってしまいます。これを避けるために、ユーザーが入力したオリジナルのクエリを取得する必要があります。

下記の例についてみてみます。"ford" と msrp が 0 より大きいものを検索しています。

```
# エンコード前のクエリパラメーター: query("ford") and filter(msrp>0)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%22)%20and%20filter(msrp%3E0
)"
```

この場合、ユーザーが入力している検索キーワードは、"ford"のみで、msrpについては、検索アプリケーションが自動的に付与しているとします。これをシステムに認識させるために、ユーザーの入力しているそのままのキーワードを p_uq パラメーターで記述します。

```
# エンコード前のクエリパラメーター: query("ford") and filter(msrp>0)
$ curl -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=query(%22ford%22)%20and%20filter(msrp%3E0
)&p_uq=ford"
```

Cxense Search - p_rs - 結果セットの設定

廃止
従来のCxense Search - p_rs - 結果セットの設定 は新しいAPI /document/search に置き換えられ、廃止されます。

概要

結果セットについて、ハイライトなどのカスタマイズ設定を行うことができます。このページでは、それらの設定などについて説明します。

コンテンツ

結果セットの設定

"p_rs" パラメーターは、結果セットに対する追加のオプション設定をJSONの配列で指定したものになります。

現在利用可能なオプションは、下記のとおりです。

- hl - ハイライトの設定
- fl - 結果セットに含まれるフィールドの設定

結果のハイライト

下記のようにhlで指定した結果をハイライト表示します。

```
hl:{description:{p:PRE,s:SUF}}
```

上記は、descriptionフィールドでマッチした箇所の前後に"PRE""SUF"

下記のクエリが実行された場合、highlights属性では、Subaruの前後に"PRE"と"SUF"が挿入されます。

http://sandbox.cxsearch.cxense.com/api/search/birt?p_q=Subaru&p_rs=hl:{description:{p:PRE,s:SUF}}

```
<highlights>
  <field name="description">
    <fragment>PRESubaruSUF Forester XT (Gold) 1:18 Scale Diecast Car Model By Autoart. Detailed interior,
exterior</fragment>
  </field>
</highlights>
```

スニペットとして利用するための文字列長を指定することもできます。(標準は100文字です)

```
hl:{description:{p:PRE,s:SUF,l:100}}
```

lを70に設定することで、上記と異なるスニペットが帰ってきます。

http://sandbox.cxsearch.cxense.com/api/search/birt?p_q=Subaru&p_rs=hl:{description:{p:PRE,s:SUF,l:70}}

```
<highlights>
 <field name="description">
  <fragment>PRESubaruSUF Forester XT (Gold) 1:18 Scale Diecast Car Model By Autoart</fragment>
 </field>
</highlights>
```

スニペットは、指定した文字列長よりも長くなる可能性もありますので注意して下さい。

フィールドリスト

flキーで結果として返されるフィールドを制御することができます。

```
fl:[description,line]
```

上記は、`descriptionline`

下記のクエリでは、"Subaru"が含まれるドキュメントの結果セットのハイライト部と`description`フィールドと`line`

http://sandbox.cxsearch.cxense.com/api/search/birt?p_q=Subaru&p_rs=hl:{description:{p:PRE,s:SUF}},fl:[description,line]

```
<match>
    <index>birt</index>
    <document>
     <id>2007_SUBY_WRX_STI</id>
     <version>1</version>
     <fields>
      <field name="description">Subaru Impreza WRX STI S204 (Metallic Gray) 1:18 Scale Diecast Car Model
By Autoart. Detailed interior, exterior, engine compartment. Opening doors, trunk, and hood, mounted on
plastic stand</field>
       <field name="line">High Performance Cars</field>
     </fields>
    </document>
    <score>0.81188864</score>
    <sortValues/>
    <highlights>
     <field name="description">
      <fragment>PRESubaruSUF Impreza WRX STI S204 (Metallic Gray) 1:18 Scale Diecast Car Model By
Autoart. Detailed</fragment>
     </field>
    </highlights>
   </match>
```

Cxense Search - p_f - ファセット

廃止
従来のCxense Search - p_f - ファセット は新しいAPI /document/search に置き換えられ、廃止されます。

概要

ファセットを利用することで、検索結果の中に含まれる指定したメタデータの値(Department = Books)を取得したり、各メタデータが含まれるドキュメント数を確認することができます。このページでは設定方法や機能に関して説明いたします。

コンテンツ

ファセット・タイプ

2つのタイプがあります。

ファセットには以下のように2種類があります。

1. シャロー -
   検索結果の一部から計算します。数はサンプルをもとにカウントしています。シャローファセットは、索引スキーマで"facet"として定義の必要がありません。このファセットは索引サイズも小さくなります。
2. ディープ - 正確な数をカウントしてファセットとして利用可能です。ただし、Cxense Search - 重複排除を利用している際には例外です。このフィールドは"facet"として索引スキーマに定義する必要があります。シャローファセットよりも高負荷となり、レスポンス時間がシャローファセットよりも多くなります。

> ファセットの値は、クエリ毎に算出されます。索引中のファセットを得るためには、全てのドキュメントがヒットするようなクエリを実行します(例: `p_aq=query("*") p_f`) 。もし、ファセットの結果だけを得たいのであれば、パラメーターにp_c=0を付与することにより、検索結果なしで、ファセットのみを得ることができます。

ファセットの定義

ファセットを指定するJSONオブジェクトのフォーマットは下記のとおりです。

1. フィールド名 (注意：大文字と小文字を区別します)
2. (オプション) 下記の表にあるファセットの設定を参照

設定	概要	フォーマット	デフォルト値	例
d	ファセットのカウントをいくつの結果から算出するかを定義。"all"の場合には、ディープファセットとして利用する。	Integer OR "all"	200	`p_f={"line": {"d": "all"}}`
c	表示するファセットのラベルの最大数	Integer	100	`p_f={"line":{ "c": 2 }}`
r	numeric か date ファセットに対してのバケット範囲の指定. 詳細は、日付フォーマットを参照 > The lower bound (f) is inclusive, the upper bound (t) is exclusive.	Json Object	なし	`p_f={"msrp" : { "r" : [{ "f" : "0", "t" : "50"},{ "f" : "50", "t" : "100"},{ "f": "100", "t" : "150"},{ "f" : "150"}] }}`

lf	ファセットのラベルを指定したカウント未満でフィルター	Integer	1	p_f={"line" : {"lf" : "3"}}

ファセットの値でドリルダウンするためには、フィルター演算子を使用します。

ファセットの例

例1：`line` と `scale` フィールドにおいて、シャローファセットを定義しています

```
p_f=line,scale
```

例2：ageフィールドを "1-3" , "3-6" , "6-" の3つの範囲にわけて定義しています

```
p_f={ "age" : { "r" : [ {"f" : "1", "t" : "3"}, {"f" : "3", "t" : "6"}, {"f" : "6"} ] } }
```

"d" パラメーターを利用して、シャローファセットの深さを制御できます。
"all" は、ディープファセットを利用する場合に指定します。

例3：例2をディープファセットとして利用する場合の定義は下記のとおりです。

```
p_f={ "age" : {"d": "all", "r" : [ {"f" : "1", "t" : "3"}, {"f" : "3", "t" : "6"}, {"f" : "6"} ] } }
```

また、"lf" パラメーターでは、最低限必要なカウントを指定します。"c" パラメーターでは、ファセットのラベルの最大数を指定します。

下記の例では、シャローファセットとして、はじめの150の結果から、最低5以上のカウントを持つトップ7のラベルを取得します。

```
p_f={ "age" : { "d" : "150", "lf" : "5", "c" : "7" }
```

日付フィールドのファセット

日付フィールドに対して:

- 2010-10-28T12:34:56　のような時刻指定
- 2y や 6M のような現在時刻からのオフセット指定

例:

```
{ "eventDate" : { "r" : [ {"f" : "6M"}, {"f" : "-0s" , "t" : "-2s"}, {"f" : "-12h", "t" : "-36h"} ] } }
```

上記の例では:

- 6ヶ月前から今まで
- 今から2秒後と12時間後から36時間後

以下が利用可能な単位です。

- s = 秒
- m = 分
- h = 時
- d = 日
- M = 月
- y = 年

本日、今週、先週などのあらかじめ決められた期間を利用したい場合には、Cxense Search - p_aq - 日付のフィルターをご参照下さい。

Cxense Search - p_sm - 並び替えとランキング

廃止
従来のCxense Search - p_sm - 並び替えとランキング は新しいAPI /document/search に置き換えられ、廃止されます。

概要

このページは、並び替え機能について説明をいたします。

並び替えと関連性のチューニングに関しては、こちらのページもご参照下さい。

コンテンツ

ソート（並び替え）

ソートモデルは、JSONの配列にフィールド名と昇順/降順を対にして定義します。

下記のように、複数のフィールドを指定してソートを行うことができます。

```
[{year:desc},{code:asc}]
```

位置情報のソートは、特別なフィールド名である"_geo-dist" を使用します。

```
[{_geo-dist:{f:geo_location,lon:149.13,lat:-35.29,unit:km,order:asc}},{code:asc}]
```

上記の例では、以下の2つのレベルでソートします。

1. `-35.29,149.13geo_location`
2. codeフィールドを昇順にソート

位置情報でのソートやフィルターについては、Cxense Search - 緯度経度での検索をご参照下さい。

スコア

ソートパラメーターが指定されていない場合、結果はTF-IDFによって計算されたスコアの降順でソートされます。一般的に、より多くの検索キーワードを含むドキュメントに高いスコアが付けられます。リクエスト中に複数のクエリや条件がある場合には、それぞれスコアに影響を与えます。フィルターは、スコアに影響を与えないことに注意して下さい。また、スコアは相対値となります。それぞれの結果セットの値を比較しても意味がありません

Cxense Search - p_sm - 緯度経度での検索

廃止
従来のCxense Search - p_sm - 緯度経度での検索 は新しいAPI /document/search に置き換えられ、廃止されます。

Introduction

For some applications, it is necessary not only to find results that match the user query, but also to filter results to be located within a specific geographic area (e.g. locations of store X in my city), or in some cases, to return results sorted by geographic area (e.g. closest location for store X from where I'm now).

This page provides more in-depth instructions on how to use geo locations in Cxense Search, complementing what is described in the main page for the Cxense Legacy Search API.

Implementation

There are three major steps needed in order to start using geo location in your search application:

- Preparing your index schema to be able to store geo locations
- Feeding documents that contain geo locations
- Executing queries using geo locations

Preparing your index schema

The first thing you will need in order to be able to use geo locations in your queries, is to make sure your index schema is configured to accept that kind of information. You will need the following three fields (with these exact names):

- `geo_location` (type: geo_point)
- `latitude` (type: double)
- `longitude` (type: double)

Below is an example of how you can create an index with these three fields:

```
curl -XPOST http://sandbox.cxsearch.cxense.com/api/index/mygeotest -d '{
   "fields": {
      "geo_location": { "type": "geo_point" },
      "latitude": { "type": "double" },
      "longitude": { "type": "double" },
      "country": { "type": "string", "indexOps": "facet" },
      "city": { "type": "string", "indexOps": "facet" }
   }
}'
```

Feeding documents with geo location

After you have your index created and ready to receive geo location information, the next step is to index your data, along with its corresponding latitude/longitude, as shown in the example below:

```
curl -XPOST http://sandbox.cxsearch.cxense.com/api/content/mygeotest/_batch -T geo_samples.json
14/14 successful.
```

The contents of the file geo_samples.json are listed below:

{"id":"sfo" ,"fields":{"city":"San Francisco","country":"USA", "latitude": 37.77, "longitude": -122.42}}
{"id":"sacramento","fields":{"city":"Sacramento","country":"USA", "latitude": 38.58, "longitude": -121.49}}
{"id":"paloalto" ,"fields":{"city":"Palo Alto","country":"USA", "latitude": 37.44, "longitude": -122.14}}
{"id":"denver" ,"fields":{"city":"Denver","country":"USA", "latitude": 39.74, "longitude": -104.98}}
{"id":"la" ,"fields":{"city":"Los Angeles","country":"USA", "latitude": 34.05, "longitude": -118.24}}
{"id":"oslo" ,"fields":{"city":"Oslo","country":"Norway", "latitude": 59.91, "longitude": 10.75}}
{"id":"paris" ,"fields":{"city":"Paris","country":"France", "latitude": 48.85, "longitude": 2.35}}
{"id":"tokyo" ,"fields":{"city":"Tokyo","country":"Japan", "latitude": 35.69, "longitude": 139.69}}
{"id":"melbourne" ,"fields":{"city":"Melbourne","country":"Australia", "latitude":-37.81, "longitude": 144.96}}
{"id":"warsaw" ,"fields":{"city":"Warsaw","country":"Poland", "latitude": 52.23, "longitude": 21.01}}
{"id":"krakow" ,"fields":{"city":"Krakow","country":"Poland", "latitude": 50.06, "longitude": 19.94}}
{"id":"saopaulo" ,"fields":{"city":"Sao Paulo","country":"Brazil", "latitude": -23.55, "longitude": -46.64}}
{"id":"rio" ,"fields":{"city":"Rio De Janeiro","country":"Brazil", "latitude": -22.90, "longitude": -43.21}}
{"id":"elverum" ,"fields":{"city":"Elverum","country":"Norway", "latitude": 60.89, "longitude": 11.56}}

How many decimal places should my latitude/longitude have?
One very important thing when working with geo location is to make sure your latitude/longitude coordinates have the proper number decimal places to provide you with the accuracy you will need for your search application. It is quite common for people to want to work with as many decimal places as possible, which may look like a good idea, but in fact may end up impacting your search performance without adding any value to your search solution.

The general recommendation is that you always use latitude/longitude coordinates with up to 4 decimal places, which already provide you with an accuracy of around 11 meters. This is more than enough for tracking specific buildings such as restaurants, coffee shops, etc. And if your search application is working only with cities, as shown in the example above, you can even reduce that to only 2 decimal places and still have enough accuracy.

For additional cases, here you can find a table with the accuracy for each additional decimal place.

Searching using geo location

Now that you already have your index schema ready and your documents indexed with the proper geo location information, the next step is to decide how you want geo location to influence your results:

- Sorting by distance, to find the closest document to point X
- Filtering search results, to only return documents found within a specific geographical area

It is important to note that when passing latitude/longitude coordinates as search parameters you must follow the same rules about the number of decimal places to use as discussed above, as those can have a great influence in the performance of your search application.

Sort by distance

The first scenario for using geo locations is when you want to sort the results to be based on their proximity to a specified point (e.g. find the coffee shop closest to point X). You can achieve that by using the parameter `p_sm` and defining geo distance as your sorting option, as shown below:

```
p_sm={_geo-dist:{f:geo_location,lon:11.56,lat:60.89,unit:km,order:asc}}
```

Sample request:

http://sandbox.cxsearch.cxense.com/api/search/geo_test?p_aq=query%28%22%2A%22%29&p_sm=%7B_geo-dist%3A%7Bf%3Ageo_location%2Clon%3A11.56%2Clat%3A60.89%2Cunit%3Akm%2Corder%3Aasc%7D%7D

Limit Search Results to a Geographical Area

In this case, you want to results to be filtered (not sorted) by a geographical area, so that for example a user could move a map and have the search results show only documents that are contained within the area displayed in the map. Such implementations are typically displayed on a (rectangular) map, hence geo filering is done by using a bounding box surrounding the center of the search, as shown

below:

```
filter(longitude:range(-123.0,-121.2)) and filter(latitude:range(37.0,39.0))
```

Example:

http://sandbox.cxsearch.cxense.com/api/search/geo_test?p_aq=query(%22*%22)%20and%20filter(longitude:range(-123.0,-121.2))%20and%20filter(latitude:range(37.0,39.0))

Cxense Search - p_dr - 重複排除

> 廃止
> 従来のCxense Search - p_dr - 重複排除 は新しいAPI /document/search に置き換えられ、廃止されます。

概要

重複排除は、結果セットの指定したフィールドに対して、重複している項目がある場合にそれらをまとめあげて1つのドキュメントとして結果を返します。このページでは、利用方法について説明します。

> Important notes
> - はじめの1,000件の結果セットのみ重複排除として扱われます。
> - 例えば、300件の重複がある場合に、最大700件の結果セットをリクエストできます（スタート位置が0になるので、p_s=699まで利用可能）
> - 総ヒット件数は、重複排除機能を利用しない時の総ヒット件数となります。
> - デープタイプのファセットと同時に利用すると,
> ファセットのカウントは概算値となります。また、結果セット内の重複件数によって算出されるため、p_c,
> p_sによって変化する可能性があります。

コンテンツ

titleフィールドに同一の値が複数入っている場合に、その重複したドキュメント排除した結果が欲しい場合には、下記のパラメーターを付与して検索を行います。

この場合、titleフィールドに同じ値を持つドキュメントは重複しているとみなして1ドキュメントとして返ってきます。

```
p_dr=title
```

下記は、デモ環境に2007_SUBY_WRX_STI_Duplicateというドキュメントidで、既存のドキュメントと重複する"2007_SUBY_WRX_STI"

```
$ curl -XPOST "http://sandbox.cxsearch.cxense.com/api/content/birt/2007_SUBY_WRX_STI_Duplicate" -d '{
   "fields" :
   {
      "name"        : "2007 Subaru WRX STi",
      "line"        : "High Performance Cars",
      "scale"       : "1:18",
      "vendor"      : "Model Cars Inc",
      "description" : "Subaru Impreza WRX STI S204 (Metallic Gray) 1:18 Scale Diecast Car Model By Autoart.
Detailed interior, exterior, engine compartment. Opening doors, trunk, and hood, mounted on plastic stand",
      "quantity"    : "1234",
      "buy price"   : "34.56",
      "msrp"        : "45.67"
   }
}'
```

"High Performance Cars" で検索すると name フィールドが、"2007 Subaru WRX STi" のドキュメントが2件ヒットします。

```
$ curl -H "Accept: application/xml" -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=filter(line%3A%22High%20Performance%20Cars
%22)"

<results>
 <totalMatched>2</totalMatched>
 <start>0</start>
 <matches>
...lines omitted for brevity...
 </matches>
 <facets/>
 <annotations/>
</results>
```

p_dr=nameで、重複排除のオプションを付与して検索を行います。その結果、結果が1件のみになったことが確認できます。また、重複削除されたことが、

<annotation name="duplicateRemoval">のセクションの情報から確認できます。

```
$ curl -H "Accept: application/xml" -XGET
"http://sandbox.cxsearch.cxense.com/api/search/birt?p_aq=filter(line%3A%22High%20Performance%20Cars
%22)&p_dr=name"

<results>
 <totalMatched>1</totalMatched>
 <start>0</start>
 <matches>
...lines omitted for brevity...
 </matches>
 <facets/>
 <annotations>
  <annotation name="duplicateRemoval">
   <value extra="originalTotalMatched" score="0.0">2</value>
   <value extra="removed" score="0.0">1</value>
  </annotation>
 </annotations>
</results>
```

Cxense Search - p_c / p_s - ページ操作

> 廃止
> 従来のCxense Search - p_c / p_s - ページ操作 は新しいAPI /document/search に置き換えられ、廃止されます。

概要

ここでは、検索結果のページ操作の方法について説明します。

利用方法

ページ操作を行うためには、下記の2つのパラメーターを利用します。

- p_c - 検索結果1ページあたりのドキュメント数

- p_s - 検索結果のスタート地点

下記の表の値をパラメータとして利用すると各ページごとに20ドキュメントの結果を得ることができます。

	p_c	p_s
ページ 1	20	0
ページ 2	20	20
ページ 3	20	40
ページ 4	20	60
ページ 5	20	80

上記のように常に同じドキュメント数を検索結果として利用する場合、p_c
は、一定の値となります。p_sパラメーターで設定される検索結果のスタート地点を増減することで、ページ操作を行うことができます。

```
p_s = 1 * ( - 1)
```

下記は、1ページあたり20ドキュメントを取得する際の4ページ目のp_sの算出方法です。

```
p_s = 20 * (4 - 1)
```

```
p_s = 20 * 3
```

```
p_s = 60
```

⚠ How many pages should be displayed?
ページの操作を何ページまで行わせるかということを決めることは非常に重要です。Cxense Search では、各パラメータの上限は、p_c=2000 と (p_c + p_s) <=10000

しかし、20ページを目安にすることをおすすめしております。理由は、検索時に多くのページ操作を行うユーザは非常に限られているためです。また、Webクローラーがより多くのサイト内の検索結果のページをクロールすることを防ぎ、Webサーバのリソースの節約やより適したページにユーザーをランディングさせることができます。

Cxense Search よくあるお問い合わせ

Cxense Search 利用フロー

概要

Cxense Search の検索対象として、大きく2つの対象にわかれます。

1. Web ページの検索
2. データベースの検索

それぞれを利用する場合のフローは下記のとおりです。

Webページの検索

1. Cxense Analytics にアカウントならびにサイトを作成します。
2. Cxense Analytics のタグを対象となるページに挿入します。
3. カスタムのメタデータを利用したい場合には、それらを付与します。詳細については、カスタムコンテンツ属性の付与 をご参照下さい。
4. 追加メタデータの反映には、必ず再クロールが必要となります。/profile/content/push を利用して各URLの再クロールを実施します。
5. 検索クエリでどのようなパラメータを利用するかなどを決めます。

   ※ 下記のURLにアクセスすることで検索を実行できます。

   アクセスポイント：sitesearch.cxense.com
   索引名：Cxense AnalyticsのサイトID

サイトIDは、下記のようにCxense Analyitcs UIから確認頂くか、1.で挿入したタグから確認することができます。

例:
http://sitesearch.cxense.com/api/search/xxxxxxx?p_aq=query(url:"*")&media=json
※ 検索APIのパラメーターなどの詳細については、Cxense Search - 検索 をご参照下さい。

6. アプリケーションから5で決めたクエリを実行します。

データベースの検索

1. 契約終了後にサーバーの準備と設定を行います。
2. 1.が完了次第、アクセスポイントをご案内します。
3. 索引スキーマを定義します。
4. 2.のアクセスポイントに対して、3.で定義されたスキーマをAPIで作成します。
5. 定義されたスキーマに対して、フィード処理を行います。
6. 検索クエリでどのようなパラメータを利用するかなどを決めます。

   ※ 下記のURLにアクセスすることで検索を実行できます。

   > アクセスポイント:2.の情報
   > 索引名:4で作成した索引名

   例:アクセスポイントが cx.search.cxense.com で、索引名がnewsの場合
   http://cx.search.cxense.com/api/search/news?p_aq=query(title:"*")&media=json
7. アプリケーションから6で決めたクエリを実行します。

frontpageに分類されたページの検索

概要

Cxense Insight に収集されたページは、自動的に「frontpage」か「article」に分類されます。

> frontpage: トップページや各カテゴリのトップページなど
>
> article: 記事コンテンツのページ

Cxense Search のWeb検索では、標準では、「article」のみが検索対象となります。

「frontpage」も検索対象としたい場合には、検索クエリで下記のパラメーターを追加します。

```
p_include_nonrecs=true
```

Webページのタイトル

概要

Webページを検索する際に、よくtitleフィールドがフロントエンドのアプリケーションで利用されると思います。

titleフィールドには通常、下記のようにog:titleが設定されている場合には、その値が利用されます。

```
<meta property="og:title" content="1"/>
```

og:titleがない場合には、titleタグに設定している値が、titleフィールドに設定されます。

<title>タイトル1</title>

参考：Document parsing

また、上記で設定されるタイトルと異なる値を利用したい場合には、下記のタグを挿入頂くことにより

その値をCxense Content/Search で利用することもできます。

```
<meta name="cXenseParse:title" content="2"/>
```

参考：Cxense Content - Review and Refinement

og:titleにタイトルは設定しているけども、titleタグに設定している値をCxense Content/Searchで

利用した場合には、それぞれの結果に含まれる下記のフィールドをご利用下さい。

recs-rawtitle

ただし、上記のフィールドは、表示用となっております。検索対象ではありません。検索する場合には、

titleフィールドに対してクエリを実行して下さい。

Webページの索引付け対象

概要

Analyticsタグをページに挿入しそのページを索引付けする場合には、

下記の条件に合致したものが索引付けされます。

1. og-url と url が一致するページ
2. og-url がないページ

下記のようなページの場合には索引付けされません。

1. og-url と url が一致しないページ

> 注意
> og-urlを設定しているページは、適切なURLをページに指定する必要があります。

索引付けされたページは、Cxense Search ならびに Cxense Content で利用可能になります。

同義語辞書の注意事項

- 概要
- 1. 辞書のフォーマット
    - 1-a. 1カラム目に同義語の元となるワードを、2カラム目に同義語展開したいワードを入力
    - 1-b. 他のワードでも同じ同義語展開をしたい場合は、別の行に記載
    - 1-c. 展開したいワードが複数ある場合は、2カラム目以降の別のカラムに記載
    - 1-d. 双方向展開する場合には、全ての組合せを記載
    - 1-e. xlsxファイルに「ふりがな」が含まれていると正常に動作しない
    - 1-f. ヘッダ行は 必要ありません。
    - 1-g. 空行や「//」で始まる行は無視されます。
    - 1-h. 同義語展開で可能なキーワード数は 32 (ExcelのAF)までです。
- 2. 文字の正規化
- 3. 辞書作成時の注意事項
- 4. 同義語展開

概要

検索時に、登録されている同義語辞書が必ず適用されます。

同義語辞書は、サンプルの同義語ファイルを追加/更新して下さい。

辞書は、Excelシートで管理され、下記のルールにしたがって設定します。

・xlsxフォーマット

・ファイル名は任意

・シート名：「@@xxx-synonym」で同義語を管理（xxxはカスタマープリフィックス）

・シート名：「@@synonym-properties」は、そのまま利用

> シート名のフォーマットが変更されました。Excelシートをアップロードする際には、必ずカスタマープリフィックスをシート名の先頭に付与して下さい。

1. 辞書のフォーマット

1-a. 1カラム目に同義語の元となるワードを、2カラム目に同義語展開したいワードを入力

1カラム目　例: BODYSHOP

2カラム目　例: BODY SHOP(スペースあり)

エクセルシートでの記述例:

```
| BODYSHOP | BODY SHOP |
```

文章内に BODY SHOP があった場合に BODYSHOP と同義語に扱われることになり、検索にかかりやすくなります。

1-b. 他のワードでも同じ同義語展開をしたい場合は、別の行に記載

例:

```
| BODYSHOP | BODY SHOP |

| ボディショップ | BODY SHOP |
```

文章内に BODY SHOP があった場合に BODYSHOP でも ボディショップでも同義語に扱われることになり、検索にかかりやすくなります。

1-c. 展開したいワードが複数ある場合は、2カラム目以降の別のカラムに記載

例:

```
| メンズ | 男性用 | 紳士服 |
```

文章内に 男性用や紳士服があった場合に メンズと同義語に扱われることになり、検索にかかりやすくなります。

1-d. 双方向展開する場合には、全ての組合せを記載

```
| メンズ | 男性用 | 紳士服 |
| 男性用 | メンズ | 紳士服 |
| 紳士服 | メンズ | 男性用 |
```

相互に検索ワードを補完したい場合は同義語として全ての組合せを記載します。

1-e. xlsxファイルに「ふりがな」が含まれていると正常に動作しない

エクセルに漢字などを入力すると「ふりがな」も一緒に付与されます。標準の設定では、この「ふりがな」は非表示となっております。「ふりがな」を追加したまま、このエクセルデータで同義語辞書を更新しても正常に動作しません。以下いずれかの方法で「ふりがな」データを削除して下さい。

- Excel上の別の場所に入力し、コピーした後に「形式を選択してペースト」で「値と数値の書式」を選択してペーストする。
- 別のテキストエディタ上でデータを入力し、コピー＆ペーストする。
- CSVへエクスポートした後で、再度、xlsxにインポートする。
- マクロを利用して、「ふりがな」を削除する。

1-f. ヘッダ行は 必要ありません。

1-g. 空行や「//」で始まる行は無視されます。

1-h. 同義語展開で可能なキーワード数は 32 (ExcelのAF)までです。

2. 文字の正規化

辞書の文字列は正規化され、下記がすべて適用されます。

・全角英数　→　半角英数

・大文字　→　小文字

・半角カナ　→　全角カナ

(検索時のキーワードも同じ正規化が行われる)

```
| BODYSHOP | BODY SHOP |
```

この場合、実際には辞書には bodyshop -> body shop が登録されます。

検索時のキーワードが「BODYSHOP」や「bodyshop」や「BODYSHOP」のいずれの場合でも「body shop」に修正されます。

3. 辞書作成時の注意事項

辞書の中に、1カラム目には、同一の値を含んではなりません。

また、その「同一」確認は正規化した後に判定されます。

下記はいずれも正しくありません。

例1: 同一値の場合

```
| BODYSHOP | BODY SHOP |
        :
        :
| BODYSHOP | BODY SHOP |
```

この場合は、全く同じ値なので、どちらかを削除する必要があります。

例2: 1カラム目が同一値の場合

```
| メンズ | 男性用 |
| メンズ | 紳士服 |
```

この場合には、下記のように記述する必要があります。

```
| メンズ | 男性用 | 紳士服 |
```

例3: キーの正規化後に同一値(両方bodyshopになる)の場合

```
| BODYSHOP | BODY SHOP |
| bodyshop | BODY SHOP |
```

4. 同義語展開

4-a. 文字列は正規化された後に、同義語展開されます。

例:「メンズ」が同義語登録されている場合に、「ﾒﾝｽﾞ」で検索した際にも、2.のルールで正規化が行われ同義語展開されます。

4-b. スペース区切りでマッチングを行います。

例:「メンズ」が同義語登録されている場合に、検索クエリが「メンズ　ファッション」の場合は展開されますが、「メンズファッション」の場合は展開されません。

> 同義語辞書のExcel

同義語辞書の管理方法

概要

同義語辞書を同義語辞書の注意事項を参考に作成します。

その後、以下の手順で作成した同義語辞書の登録を行います。

> APIの実行権限
> 辞書の管理を行うためには、辞書登録対象のサイトグループに対する「更新」権限が必要となります。

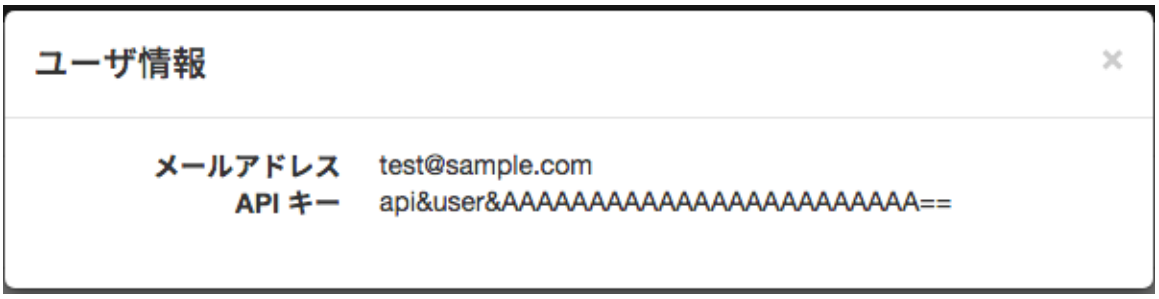
事前準備

カスタム辞書の準備が終わったら、次のステップとして、設定を反映するためにシステムにアップロードします。

下記の2つのスクリプトを利用して作業を行うため、これらを同じディレクトリに保存します。

- cxbundle.sh
- cx.py

まず、cx.pyのusername(ユーザー名)とsecret(APIキー)を編集する必要があります。
これらの値は 、insight.cxense.com にログインして、画面右上のユーザー名をクリックすることで表示されますが、
デフォルトでは表示されず、表示するためにはAPIキーの参照権限が必要です。詳しくは APIキー確認方法 を参照ください。



例：cx.js の 57 - 59行目

```
# Default configuration.
username = 'test@sample.com'
secret = 'api&user&AAAAAAAAAAAAAAAAAAAAAAAA=='
```

APIキーは、"api" からはじまり "==" で終わります。

辞書の更新

下記のスクリプトを実行し、辞書を更新します。API実行後、辞書はすぐに反映されます。

```
sh cxbundle.sh upload {辞書ファイル名} {worker_group_id}
例: sh cxbundle.sh upload synonym.xlsx 123456789
以下の様なメッセージが出れば正常に更新されています。
---------------------------
{
 "messages": [
   "INFO      [2014-02-07T04:40:56.219Z] Currently processing data for sheet 'synonym-properties'...",
   "INFO      [2014-02-07T04:40:56.219Z] Currently processing data for sheet 'synonym'...",
   "INFO      [2014-02-07T04:40:56.245Z] Compiling dictionary 'synonym' having 1 entries...",
   "INFO      [2014-02-07T04:40:56.754Z] Successfully pushed new configuration bundle for worker group
123456789."
 ],
 "workerGroupName": "Synonym",
 "workerGroupId": "123456789"
}
---------------------------

重複エントリなどエラーがあれば、メッセージが出力されますので、修正後再度更新します。
```

※　同じシート名は、上書き更新されます。したがって、現在の同義語辞書に同義語を追加登録する場合には、現在登録している同義語も辞書ファイルに含めてください。

worker_group_id
worker_group_idがわからない際には、support_jp@cxense.comまでご連絡下さい。

辞書のダウンロード

下記のスクリプトを実行し、登録している辞書データをダウンロードできます。出力ファイルは、zip形式となります。出力ファイルを解凍後、input.xlsxファイルで確認することができます。

```
sh cxbundle.sh download {出力ファイル名(zip)} {worker_group_id}
例: sh cxbundle.sh download dictionary.zip 123456789
```

文字の正規化

概要

Cxense Search で扱われる文字列は、下記のように正規化されます。

正規化

・全角英数　→　半角英数

・大文字　→　小文字

・半角カナ　→　全角カナ

・㈱, ①など　→　(株), 1

検索結果

上記の正規化が行われた文字列が結果として返されます。

ただし、タイトルに関しては、recs-rawtitleに元データが格納されているので、必要に応じてご利用下さい。

検索クエリ

上記の正規化が行われ、検索が実行されます。

したがって、大文字や小文字などの区別なく検索することができます。

索引からのドキュメントの削除

概要

Cxense Searchでは、Webコンテンツをクロールして作成される索引と、

要件にあわせてデータベースなどのデータをフィードするカスタムフィードの2つの方法があります。

それぞれによって、削除する方法が異なりますので、本ページではそれについて説明いたします。

Webコンテンツ

WebページにAnaltyicsのタグを挿入することによって、1時間に3アクセスのあるページへ対して、クロールが行われます。

クロールに関しての詳細はクロールに記載があります。

クロールされたページは、サイトIDを索引名とする索引に追加されて、検索対象となります。

また、ページが削除され、その後、該当のページが再クロールされた際に、HTTPステータスコードが404となれば、該当のページは索引から削除されます。

手動で索引からドキュメントを削除する際には、/document/delete を利用します。

> 注意
> ただし、手動で行った場合には下記の注意が必要です。
>
> ページがWebページ上から削除されずに、1時間に3PV以上のアクセスがある場合、再クロールされるタイミングで索引にそのドキュメントが追加され、検索対象となります。

カスタムフィード（DBなど）

下記のページに記載しているAPIを用いてドキュメントの削除を行います。

Cxense Search - フィードと削除

もしくは、索引に削除フラグ用のフィールドを追加して、削除ドキュメントは、それにtrueなどに設定し、not+query(delete-flag:"true")などとします。(フィールド名がdelete-flagの場合)

Cxense Search 追加情報

このセクションと子ページには、推奨／もしくは必須の利用方法と、Cxense Searchに利用されるソリューション例を管理しています。

Cxense Search - 辞書管理

概要

Cxense Search でご利用いただける辞書に関してこのページで説明いたします。また、その辞書の管理方法や利用ステップについても説明いたします。

もし、辞書管理機能が無効の場合には、support_jp@cxense.com にご連絡ください。

> カスタム辞書は、全ての索引に対して反映されます。現在、個別の索引のみへの反映はサポートされておりません。

辞書管理

辞書管理を行うために下記を行う必要があります。

1. それぞれの辞書のフォーマットを理解します
2. 辞書のそれぞれのタイプ(おすすめ辞書のホワイトリストやシノニムなど)の特徴を理解します。
3. APIを使用して、辞書の更新を行います。

辞書のフォーマット

Excel ワークブック (.xslx) にて辞書を管理します。それぞれのワークシート名によって、辞書の種類が定義されます。
下記が現在有効なワークシートの名前です。(@@ は、必ずワークシート名に記述する必要があります)

- @@suggest-whitelist
- @@suggest-blacklist
- @@synonym
- @@spell-whitelist

それぞれのワークシートには、<シート名>-properties (例 @@synonyms-properties)のような設定が記述されているワークシートが必要です。このワークシートでは、言語や文字の標準化や照合ルールなどが記載します。このページの最後の部分にサンプルファイルへのリンクがあるので、そのリンクからダウンロードして、再利用して下さい。

Define Dictionary Language
サンプルファイルは、既にそれぞれの辞書の種類に応じた設定が行われております。したがって、プロパティシートでは、言語コードをISO 639の2文字のコードにしたがって変更する必要があります。設定した言語によって、適切な言語処理が行われます。サンプルファイルでは、en(英語)となっております。

// Defines the language of the data in the worksheet, typically. Required for some languages so that, e.g., tokenization	
// and other linguistic operations will work as expected.	
tokenizer-context	**en**

設定可能なオプションは、Processing をご参照下さい。

辞書のタイプ

クエリコンプリーション辞書（日本語未対応）

1日毎に人気のあるクエリをもとにクエリコンプリーションの辞書を作成いたします。この辞書に対して、さらに下記の2つの辞書を用いて制御します。

- @@suggest-whitelist - クエリコンプリーションに必ず追加したいキーワードをリスト化します。（※ クエリが、前に実行されていなくてもかまいません）
- @@suggest-blacklist - クエリコンプリーションに追加したくないキーワードをリスト化します。

もし、自動生成したくない場合には、support_jp@cxense.com にご連絡ください。

スペルチェック辞書（日本語未対応）

1日毎に人気のあるクエリをもとにスペルチェックの辞書を作成いたします。この辞書に対して、さらに下記の2つの辞書を用いて制御します。

- @@spell-whitelist - スペルチェックに必ず追加したいキーワードをリスト化します。（※ クエリが、前に実行されていなくてもかまいません）
- @@suggest-blacklist - スペルチェックに追加したくないキーワードをリスト化します。

もし、自動生成したくない場合には、support_jp@cxense.com にご連絡ください。

同義語辞書

@synonym

同義語辞書を利用することで、ユーザーのクエリを展開することができます。エクセルのワークシートで下記ルールに基づいて定義する必要があります。

- 1列目は、キー（元のキーワード）となります。キーは、重複することができません。
- 2列目以降は、バリューとなります。展開したいキーワードを記述します。
- 片方展開になります (e.g. apple => orange)。もし、両方展開したい場合には、他の行にキーとバリューを逆にしたものを追加します (例 orange => apple)。

あわせて、同義語辞書の注意事項もご確認下さい。

下記が同義語辞書の例となります。

Dictionary

キー	値
apple	orange
i pod	ipod
apple juice	martinelli's apple juice

上記の場合に、クエリは下記のように実行されます。

結果

クエリ	結果	参考
query("apple")	query("apple") or query("orange")	
query("i pod")	query("i pod") or query("ipod")	
query("apple discount i pod")	query("apple discount i pod") or query("orange discount i pod") or query("apple discount ipod") or query("orange discount ipod")	
query("apple juice")	query("apple juice") or query("martinelli's apple juice")	一番長いものが適合します。したがって、apple => orange では展開されません。
query("ipod")	query("ipod")	片方展開のみなので、値 => キーには変換されません。

ワークブックのサンプル

下記はそれぞれのタイプのサンプルファイルになります。プロパティワークシートは、すでに辞書のタイプに合わせて設定済みです。また、これらにはいくつかのサンプルワードが含まれます。

- 同義語のサンプルファイル
- おすすめのためのホワイトリスト辞書のサンプルファイル
- おすすめためのブラックリスト辞書のサンプルファイル
- スペルチェッキングのホワイトリストのサンプルファイル

Cxense Search - 関連性のチューニング

概要

検索アプリケーションを開発する際に、結果のランキングチューニングやクエリ式など考慮すべき点がいくつかあります。このページでは、それらの項目について触れます。

目次

- 概要
- 目次
- 関連性の基本コンセプト
    - クエリの種類
        - AND クエリ
        - フレーズ クエリ
        - OR クエリ
    - 検索対象
    - フィールドの重み
    - 並び順

関連性の基本コンセプト

クエリの種類

ユーザーがどのように検索を行いたいかを考える必要があります。

そのために、まず下記の3点を理解する必要があります。

AND クエリ

```
?p_aq=query("venture capital", token-op=and)
```

これは、全てのキーワードが含まれる場合にヒットと判断します。上記例では、venture と capital
がドキュメント中にどちらも含まれる必要があります。ただし、その順番は特に関係ありません。例えば、次のような文章であればヒットします。

例1: raised more venture capital money...

例2: is initiating a new venture with capital raised...

例3: for his new venture he raised capital from..."

これらは検索アプリケーションで利用される最も一般的なオペレーターです。Cxense Search - 検索もご参照下さい。

フレーズ クエリ

```
?p_aq=query("venture capital", token-op=phrase)
```

上記のクエリは、"venture capital" が完全に一致するドキュメントのみ取得できます。例えば、"raised more venture capital money..." はヒットします。"is initiating a new venture with capital raised..." の場合にはヒットしません("with" が "venture" と "capital" の間に入っているから)。

よく検索アプリケーションでは、検索ボックスにおいて、"(ダブルクオート) や
'(シングルクォート)で囲まれたキーワードに対して、フレーズ検索を実行します(※フロントエンドのアプリケーションによって、判断してクエリを作成する必要があります)。このフレーズ検索は、ユーザーにとって、非常に便利な機能になります。

OR クエリ

```
?p_aq=query("venture capital", token-op=or)
```

これは、指定したクエリが1つでも含まれれば、そのドキュメントをヒットと判定します。上記のクエリの場合には、"venture" か "capital"
が含まれているドキュメントがヒットします。例えば、 "he decided to venture down the hall..." や "Brasília is the federal capital of Brazil"
がヒットします。

検索対象

どのトークンタイプの演算子(and, phrase, or
)を利用するかを決めたら、次に、どのフィールドを検索対象とするかを決める必要があります。多くの場合、「全て」となるかもしません。しかし、より良い検索のためには、検索したいフィールドを指定することは重要です。

```
?p_aq=query("financial systems", token-op=and)
```

上記のクエリの場合には、システム側で自動的にクエリに"_all"
をフィールドとして追加変換後実行します。これにより、上記のクエリはフィールドを意識せずに全体に対して検索を行います。

```
?p_aq=query(_all:"financial systems", token-op=and)
```

"_all" オプションは、全てのフィールドを検索対象とする場合に利用します。ただし、索引スキーマの定義で、"storeOnly"
に設定しているフィールドは除きます。もし、title や body だけを対象にしたい場合でも、_all を設定するとそれ以外のフィールド(例:`category`, `related_content`, `unitsInStock`)も検索対象とします。

下記のようにフィールドを指定することで、より正確な検索を行うことができます。

```
?p_aq=query(title,body,description,tags,url,author:"financial systems", token-op=and)
```

上記の例では、どのフィールドに検索するかを明確に指定しております。このようにすることで、「なぜ、このドキュメントがヒットしたの？」という問い合わせに対してより簡単に説明することができます。

フィールドの重み

次のステップとして、検索結果のランキング（表示順）について考えてみます。まずは、下記の表をご参照下さい。

	title	body	tags
Document 1	Market Research Findings - 2012	This document summarizes the findings from the 2012 market research study...	research, 2012
Document 2	About the market crash of 1929	All the available research on the market crash of 1929...	stock, market, 1929
Document 3	XYZ begins to explore new market	After a few years focused on research, company XYZ began exploring a new market...	XYZ

上記のドキュメントに対して、下記のクエリを実行します。

```
?p_aq=query(title,tags,body:"market research", token-op=and)
```

この場合、どのドキュメントが上位のランキングにくるべきでしょうか？多くの人は、"Document 1" がより高くなると期待しています。それは、検索キーワードが body ではなく、title にヒットしているからです。

body よりも他のフィールドに対して重み付けする設定を行うと下記のようになります。

```
?p_aq=query(title^5,tags^3,body:"market research", token-op=and)
```

上記のクエリの場合、下記の処理を行います。

- market と research を含むドキュメントを検索します。
- これらのキーワードは、`title`, `tags`, `body` フィールドに含まれます。
- title フィールド中にキーワードが見つかった場合にはbodyに含まれる場合と比較して、5倍 (`title^5`) の重み付けにします (標準は1です)
- tags フィールド中にキーワードが見つかった場合にはbodyに含まれる場合と比較して、3倍 (`tags^3`) の重み付けにします (標準は1です)

並び順

最後のステップとしては、取得した結果がどのような順番で並び替える必要があるかを考慮することです。これにより、検索結果のトップページにどのドキュメントを表示するかが決まります。

```
?p_aq=query(title^5,tags^3,body:"market research")&p_sm=publication_date:desc
```

上記の例では、`publication_date` の降順に並び替えています。この場合には、フィールドで重み付けをしていても、それらを考慮して計算されるスコアが完全に無視され、日付フィールドの順でソートされます。

現行のSearchと旧バージョンのSearchの対比

Cxense

searchは現在2つのソリューションで提供しています。従来までの方法は現在段階的に廃止されつつあります。新しいソリューションについては2015年9月からリリースされました。
双方とも内部コア部分については全く同じく、ElasticsearchをCxenseのシステムに内包しています。以下の項目で双方の相違点を解説します。

Searchソリューションの違い

認証方法が変わりました。

旧バージョンのSearchはIPマスクを利用していました。特定のIPアドレスか設定されたIPからのみクエリとドキュメントの処理ができました。
現行のSearchでは他のCxense製品同様の認証方法を利用します。

個別のSearchクラスタが不要になりました。

旧バージョンのSearchではSiteSearchだけでは利用できない高度な検索を使いたい場合、個別のSearchクラスタが必要で、都度開発部門に依頼して作業する必要がありました。
現行のSearchでは全ての機能を個々にクラスタを作成することなく利用できます。(従来のSearchの高度な機能は新しいSearchで利用ができません。下記の「以下の機能は移行されていません」をご参照ください。)

スキーマは不要です。

旧バージョンのSearchはスキーマをベースにしていました。
現行のSearchではインデックスフィールドを予め指定する必要がありません。

新しいクエリとドキュメントのフィードについて

旧バージョンのSearchでは検索はパラメータを全て含むURLを作成してリクエストします。また、ドキュメントをフィードするにはHTTP POSTコマンドを用いて行われます。
現行のSearchでは他のCxense APIと同じようにパスとリクエストオブジェクトのパターンでリクエストされます。
/document/search と /document/updateが利用されます。この2つのAPIの使用例については document feeding と search のリンク先をご参照ください。

以下の機能は移行されていません。

以下の機能は旧バージョンのSearchでは動作していましたが、現行のSearchでは動作しません。
これらは将来的に追加されていく予定です。

- 現行のSearchではランキングは一つの観点でしか動作しません。例えば関連性と最新ランキングの結合はできません。
- 検索結果が0件だった時以外に自動生成されるクエリコンプリション辞書は、現行のSearchでは一部動作しません。
- もしかして?検索 (スペルチェック) 辞書の利用はできません。