Search Settings API Overview

Storefront search capabilities within the Kibo Composable Commerce Platform (KCCP) are supported by Solr 8 and an extensive set of configurable options that can be customized for the implementation. This guide explains several of the concepts about relevancy, filtering, that should be understood before configuring search strategies.

In the the catalog/admin resource, catalog/admin/schemaDefinition contains the API calls to manage the schema and catalog/admin/search contains the API calls to configure individual search settings. In the catalog/storefront resource, use catalog/storefront/productsearch to make the actual Site Search and Visual Search calls. The instructions for configuring product search settings via API are also available in the Catalog guides.

Adjustable Search Settings

The core search settings that can be adjusted are listed here, though not all are supported by every type of search. See the appropriate catalog/admin/search resource to see which of the below settings are supported by each kind of search.

Search Setting	Description
`fieldWeights`	The weighted relevancy settings that you want Kibo to use. Refer to the Field Weights section below for more information about field weights.
`fieldValueBoosts` and `customBoosts`	The boost functions that you want Kibo to use. Refer to the Boosting Fields section below for more information about boosting.
`minimumMatchPercents`	The minimum match (MM) percent is a percentage of the number of search terms that must match or can be missing. Kibo's default MM percent is 75%. Refer to the Minimum Match Percent section below for more information about minimum match percents.
`searchSynonymSettings`	The search synonym settings that you want Kibo to use. Search synonyms are synonyms to user search queries that you want Kibo to match on. For example, if a user searches for `blouse`, you want Kibo to show all search results that would appear if a user searched for `shirt` as well. In this example, `blouse` and `shirt` are search synonyms. Refer to the Search Synonyms section below for more information about search synonym settings.

How Solr Filters Attributes

When defining attributes in the product catalog, they are flagged to indicate whether they allow filtering and sorting. These options enable attributes to be indexed in the search schema and used as search terms, by filtering and sorting all catalog data based on the search terms for certain attributes that the user submitted a query for. This provides the backbone of the Solr-based search functionality, as it determines how the queries are processed in KCCP and Solr to return search results for the user.

Different dynamic field patterns can be used for filtering and sorting actions, and are case-sensitive.

Type	Action	Definition
string	Filter	<dynamicField name=”*_ss” type=”string” indexed=”true” stored=”true” multiValued=”true” docValues=”true” />
string	Sort	<dynamicField name=”*_s” type=”docstring” indexed=”true” stored=”true” docValues=”true” />
boolean	Filter and Sort	<dynamicField name=”*_b” type=”boolean” indexed=”true” stored=”true” docValues=”true” />
float/number	Filter	<dynamicField name=”*_fs” type=”pfloat” indexed=”true” stored=”true” multiValued=”true”/>
float/number	Sort	<dynamicField name=”*_f” type=”pfloat” indexed=”true” stored=”true” />
date	Filter	<dynamicField name=”*_dts” type=”pdate” indexed=”true” stored=”true” multiValued=”true” />
date	Sort	<dynamicField name=”*_dt” type=”pdate” indexed=”true” stored=”true” />

Filtering and sorting attributes uses the pattern __. For example, the string tenant~brand-name would reference the following two fields:

- Facet field: tenant_brand-name_ss
- Single-value sort field: tenant_brand-name_s

By default, only attributes defined in the catalog and the indexed in the search schema can be queried for as a search term. However, _txtin can be used to insert fields into the term search without storing or indexing the field. Thus, it can be used in the below format to perform a search for custom attributes as if it were a valid term.

Use	Definition
all	<dynamicField name=”*_txtin” type=”string” indexed=”false” stored=”false” multiValued=”true”/>

To perform searches with custom attributes, use the pattern __txtin such as in the example tenant~brand-name_txtin.

Example

This example walks through a full search case to illustrate how these patterns are used to generate the final query for Solr. The example schema defines a “brand name” field to be used as both a default text search field and a ngram field for the product suggest function. The schema object would look like this:

Field	Definition
tenant~brand-name	default
tenant~brand-name	ngram

Thus, the Solr schema would include the following entries that define the brand name field.

```
<field name="tenant_brand-name_default" type="text_lang_en" indexed="true" stored="false"/>

<field name="tenant_brand-name_ngram" type="edgytext" indexed="true" stored="false"/>
<copyField source="tenant_brand-name_txtin" dest="tenant_brand-name_default"/>
<copyField source="tenant_brand-name_txtin" dest="tenant_brand-name_ngram"/>
```

If a site search was performed for “mystic,” then the searchSettings object would be used to determine what fields and weights should be used in the Solr query. In this example, the following weights are defined for the fields:

Field	Definition	Weight
productName	default	7
tenant~brand	default	5

So, the resulting query would be q=mystic&qf=productname_default^7 tenant_brand-name_default^5.

Field Relevance

Relevance determines how search results are ranked and displayed to the user, and can be customized at the attribute level to prioritize certain terms by adjusting the “weight” of each attribute. Results that match the search terms with higher total weights will be ranked higher (towards the top of the results) when displayed on the storefront than other results with lower total weights. For instance, if a customer searches for a blue shirt in a particular brand and the brand attribute is weighted higher than the color attribute, then an item that matches the brand will be more relevant and thus ranked higher in the search results than an item that is only blue.

Weighting Fields

Each searchable field in the schema can be weighted with a different value to prioritize certain fields when checking for search term matches. If a search was being made based on brand and product name, and the product name was weighted higher, then items that match the queried product name will be listed higher in the search results than items that only match the queried brand. Items that match both the queried product name and brand will be ranked highest.

The fieldWeights array is a list of objects, each entry of which identifies a searchFieldName and the appropriate weighting values either as a simple weight on the search term or as a phrase. The basic weight value raises the relevancy score of a search result that includes this particular term, while phraseWeight raises the relevancy score of a search result that has the exact search phrase with all terms next to each other. Note that the search field being weighted must be indexed in the schema.

For example, the below sample would set the field for the brand attribute to a weighting of 10. These weights are relative to the weights of the other results – KCCP does not enforce a strict scale such as 1-10. This snippet would be part of the site search settings object included in a larger POST or PUT call to the Search Settings API as detailed in that guide.

```
"siteSearchSettings": {
    "fieldWeights": [
      {
       "phraseWeight": 0,
       "searchFieldName": "tenant_brand_desc_lenient",
       "weight": 10
      },
```

Boosting Fields

Further adjustments can be made by boosting certain values within each attribute. Boosting further fine-tunes weights and relevancy by adjusting the weight of a result based on the attribute’s value relative to other possible values of the attribute, allowing certain options to be prioritized within the same product. For instance, boosting “blue” above “red” for the color

attribute would allow blue shirts to be ranked higher in the search results than red shirts of the same brand and design, if the user did not specify color in their query.

Boost particular values of attributes to raise their search result ranking and prioritize those values in relation to each other, instead of changing the actual weight of the whole field which does not differentiate between values. For instance, an attribute for color can have a number of possible values that describe the product's different color options. These boosting options can be applied as the fieldValueBoost object when configuring site search settings, product suggest settings, and listing settings.

Parameter	Type	Description
boostType	string	The type of boost being applied.
fields	array	A list of the fields that boosting is being applied to. Limit of 20 entries.
fieldName	string	Contained in each entry of the *fields* array. The name of the field, validated against the catalog attributes.
valueExpressions	array	Contained in each entry of the *fields* array. A list of objects that define the expressions being used to set the boosts of this field relative to its possible values. Limit of 20 entries.
value	string	Contained in each entry of the *valueExpressions* array. The value of this field.
boost	integer	Contained in each entry of the *valueExpressions* array. The actual amount by which to boost the weight of this value on the attribute. Must be greater than or equal to 0.
operator	string	Contained in each entry of the *valueExpressions* array. The operator by which to compare the value of the field against the search term. The options are: gt, gte, lt, lte, eq, neq.

While the boost data is also documented in the Search Settings API guide, this snippet provides a quick look at what this data looks like in those requests. This is an example that would boost the "Mystic" brand above the "Mundane" brand.

```
{
  "boostType": "simple",
  "fields": [
    {
      "fieldName": "tenant~brand",
      "valueExpressions": [
        {
          "value": "mystic",
          "boost": 20,
          "operator": "eq"
        },
        {
          "value": "mundane",
          "boost": 10,
          "operator": "eq"
        }
      ]
    }
}
```

Minimum Match Percent

The minimum match (MM) percent indicates the percentage of terms in a search query that must be contained in the search results. In order to calculate how many terms are matched, Kibo

multiplies the MM value by the number of terms in the search query and either rounds down or up depending on the MM value. The default value is 75%.

You can specify a minimum match percent using the `minimumMatchPercent` field.

If you have search synonyms that match on a user's search query, Kibo ignores the `minimumMatchPercent` value and uses a minimum match of 100% instead. Refer to Minimum Match and Synonyms for more information.

Valid values for the MM percent are integers between 0-100, including negative integers. A positive integer indicates how many terms are required to match, rounded down. A negative integer indicates how many terms can be missing, rounded up.

The higher the MM percent the more narrow the product results Kibo returns.

Refer to the following table for some example outcomes of different MM percents:

MM Percent	Number of Search Terms	Outcome
75%	5	3 terms must match (5 * 0.75 = 3.75, rounded down to 3)
50%	5	2 terms must match (5 * 0.5 = 2.5, rounded down to 2)
25%	5	1 term must match (5 * 0.25 = 1.25, rounded down to 1)
-25%	5	4 terms must match (1 term can be missing) (5 * 0.75 = 3.75, rounded up to 4)
-50%	5	3 terms must match (2 terms can be missing) (5 * 0.5 = 2.5, rounded up to 3)

Search Synonyms

You can add search synonyms to Kibo's search settings to expand Kibo's search capabilities. Search synonyms are synonyms to user search queries to which Kibo expands the search results. For example, a user may search for *boots*, but you want the search results to also include results for *shoes*, *slippers*, and *pumps*. You can use search synonyms to accomplish this.

Enable Search Synonym Settings

To enable the search synonyms settings, in the search settings set `expandSynonyms` to `True`.

Synonym Definitions

Synonym definitions are individual definitions of synonyms. Synonym definitions are scoped to a particular tenant, site, and locale code.

A synonym definition can include either a key and a list of synonyms, or just a list of synonyms.

Refer to Synonym Expansion Types for more information about using a key.

Multiple synonym definitions for a site are placed in a site's synonym definition collection. Kibo automatically adds any synonym definitions you create to the applicable synonym definition collection. Refer to Synonym Definition Collections for more information.

Synonym Definition Operations

You can perform the following operations on synonym definitions:

- AddSynonymDefinition
- DeleteSynonymDefinition
- GetSynonymDefinition
- GetSynonymDefinitions
- UpdateSynonymDefinition

Synonym Definition Body

A synonym definition has the following body:

```
{
   "key": "string",
   "synonymId": "int",
   "synonyms": "string"
}
```

Synonym Definition Collections

Synonym definition collections are collections of multiple synonym definitions. Each site can have one synonym definition collection. Kibo automatically adds any synonym definitions you create to the applicable synonym definition collection.

Synonym Definition Collection Operations

You can perform the following operations on synonym definitions:

- GetSynonymDefinitionCollection
- UpdateSynonymDefinitionCollection

Synonym Definition Collection Body

A synonym definition collection has the following body:

```
{
   "localeCode": "string",
   "siteId": "int",
   "synonymDefinitions": [
      {
         "key": "string",
         "synonymId": "int",
         "synonyms": "string"
      }
   ],
   "tenantId": "int"
}
```

Synonym Expansion Types

There are two search synonym expansion types: one way and two way.

One Way

The one way expansion type includes a key and a set of synonyms to which Kibo expands the search results. This is a one way expansion; Kibo does not expand the search results to the key or

any other listed synonyms.

You are not required to provide a key. Do not provide a key for two way expansion.

```
{
   "localeCode": "en-us",
   "siteId": "22327",
   "synonymDefinitions": [
      {
         "key": "blouse",
         "synonymId": "1",
         "synonyms": [
            "shirt", "top", "T-shirt"
         ]
      }
   ],
   "tenantId": "18263"
}
```

In the above example, if a user searches for *blouse*, then Kibo expands the search results to include the results for *shirt*, *top*, and *T-shirt*. However, if a user searches for *top*, Kibo does not expand the search results to include the results for *blouse*, *shirt*, or *T-shirt*.

Two Way

The Two Way expansion type does not include a key, but instead only contains a list of synonyms to which Kibo expands. This is a two way expansion; for any listed synonym, Kibo expands the search results to all other listed synonyms.

For example, you specify the following:

```
{
   "localeCode": "en-us",
   "siteId": "22327",
   "synonymDefinitions": [
      {
         "synonymId": "1",
         "synonyms": [
            "shirt", "top", "T-shirt", "sweater", "blouse", "pullover"
         ]
      }
   ],
   "tenantId": "18263"
}
```

In the above example, if a user searches for *top*, then Kibo expands the search results to include the results for *shirt*, *T-shirt*, *sweater*, *blouse*, and *pullover*.

Kibo only matches multi-word search terms if they are contained within quotes in the search query. If a multi-word search term is not contained within quotes in the search query, Kibo uses each word in the search to find synonyms. For example, if a shopper searches for `blue shoes` without quotes, Kibo expands both `blue` and `shoes` to any applicable synonyms.

Boosting Synonyms

Kibo also allows you to set boost values for synonyms using the `searchSynonymSettings`. Boost values are separated into the following two different parts.

- **Main part:** The terms that the user enters in their search query. Use the `mainPartBoost` field to set boost values for the main part.
- **Synonym part:** The synonym terms to which Kibo expands the search query. Use the `synonymPartBoost` field to set boost values for the synonym part.

If you set the `mainPartBoost` higher than the `synonymPartBoost`, then Kibo displays the search results that match on a user's search terms higher in the search results. If you set the `synonymPartBoost` higher, then Kibo displays the search results that match on an expanded

synonym higher in the search results.

For example, you specify the following:

```
"searchSynonymSettings": {
   "mainPartBoost": 1.2,
   "synonymPartBoost": 1.1
}
```

In the above example, any results that match the user's search query terms appear higher in the search results list than any results that match an expanded synonym.

The following are typical boost values:

- `mainPartBoost`: 1.2
- `synonymPartBoost`: 1.1

Another multiplier of mainPartBoost and synonymPartBoost that can be applied is **phraseBoost**. PhraseBoost can further raise the relevancy of results in which the phrase is found with both words next to each other, and is set to a default of 1.0 that will still score those results higher. The phraseBoost value can be changed when needed, allowing more tuning if the search results are not as expected based on the data distribution.

Match On Any Term

The setting **matchOnAnyTerm** can be set to allow for a looser query. As detailed below, minimum match percents do not work with synonym expansion which means that every term must exist in a document for it to be displayed as a search result. However, matchOnAnyTerm can reduce this requirement when synonyms are in use. With matchOnAnyTerm, a document can be returned by the search as long as at least one term exists within it.

For example:

- Search Phrase: "bbq grill"
- Synonym: "bbq" = "propane"
- MatchOnAnyTerm: Set to either true (enabled) or false (disabled, by default)

If matchOnAnyTerm is not enabled, then only documents that contain both ("bbq" and "grill") or ("propane" and "grill") are returned. When matchOnAnyTerm is enabled, then any document that contains ("bbq" or "grill" or "propane") will also be returned. Relevancy is still taken into account, so the documents that contain more of the terms or the full phrase will be ranked higher.

Minimum Match and Synonyms

When you use both a minimum match percent and a synonym definition list, Kibo uses a minimum match of 100% for the search query and its synonyms.

Refer to the following table for some examples of this behavior:

Search Query	Synonym Definition List	Expanded Search

Search Query	Synonym Definition List	Expanded Search

Gucci shoes	<pre>"synonymDefinitions": [
	  {
	  "key": "shoes",
	  "synonymId": "1",
	  "synonyms": [
	  "boots", "booties", "pumps", "heels", "sandals",
	"sneakers", "flats", "loafers", "oxfords"
	  ]
	  }
	]</pre>
		Gucci shoes, Gucci boots, Gucci booties, Gucci pumps,
		Gucci heels, Gucci sandals, Gucci sneakers, Gucci flats,
		Gucci loafers, Gucci oxfords

Red Shoes	<pre>"synonymDefinitions": [
	  {
	  "key": "shoes",
	  "synonymId": "1",
	  "synonyms": [
	  "boots", "booties", "pumps"
	  ]
	  },
	  {
	  "key": "red",
	  "synonymId": "2",
	  "synonyms": [
	  "magenta", "ruby", "pink"
	  ]
	  }
	]</pre>
		Red shoes, red boots, red booties, red pumps,
		magenta shoes, magenta boots, magenta booties, magenta pumps,
		ruby shoes, ruby boots, ruby booties, ruby pumps,
		pink shoes, pink boots, pink booties, pink pumps

If you do not have search synonyms that match on a user's search query, Kibo uses the `minimumMatchPercent` value. Refer to Minimum Match Percent for more information.

Default Relevancy Weights

Kibo's search settings include the following default relevancy weights:

```
"siteKeywordRelevancy": {
  "productCodeWeight": 10,
  "nameWeight": 8,
  "descriptionWeight": 4,
  "keywordsWeight": 4,
  "attributesWeight": 1,
  "upcWeight": 2,
  "mpnWeight": 2,
  "categoryNamesWeight": 2
},
"sitePhraseRelevancy": {
  "nameWeight": 5,
  "descriptionWeight": 3,
  "attributeWeight": 1
}
```

Deep Paging

When you're paging through thousands or more of products using the Kibo API, Kibo recommends using deep paging, which optimizes your retrieve requests. Traditional paged requests require Kibo to create an internal queue of paged product results using a combination of `pageSize` and `startIndex`, which requires a large amount of memory and subsequently degrades Kibo's performance.

With deep paging, on your first request you do not specify a `startIndex`, and instead only specify a `pageSize` and initialize the `cursorMark` parameter using an asterisk (`*`). Kibo then returns the first set of product results, and provides an encoded `nextCursorMark` value, which you can then

use for the `cursorMark` in the next request. This allows Kibo to optimally page through products without using the same amount of memory as traditional paged requests.

You can use deep paging in the storefront Search and GetProducts operations.

Deep Paging Examples

For example, you want to page through 5,000 thousand products 50 at a time in a search request for black and sorted by product name. To optimize this request, you perform the following initial request:

GET `api/commerce/catalog/storefront/productsearch/search/?sortBy=productName+asc&pageSize=50&query=black&cursorMark=*`

The above request returns the following response:


```
{
   "facets": [],
   "nextCursorMark": "AoI%2fGXZlcnRpY2FsIGNvbnZlcnQgMi4wIGludGVyY2hhbmd",
   "startIndex": 0,
   "pageSize": 50,
   "pageCount": 20,
   "totalCount": 5000,
   "items": [
      {
         "productCode": "119Cfg002",
         "productSequence": 1109,
         "productUsage": "Configurable",
         "fulfillmentTypesSupported": [
            "DirectShip"
         ],
         "goodsType": "Physical",
         "content": {
            "productName": "1 Gallon Black paint",
            "productFullDescription": "New",
            "productShortDescription": "TEST text",
            "metaTagTitle": "1 Gallon Black paint",
            "seoFriendlyUrl": "1-gallon-black-paint",
            "productImages": []
         },
         "purchasableState": {
         "isPurchasable": false,
         "messages": [
         {
            "severity": "Info",
            "source": "ConfigurableProduct",
            "message": "Not done configuring",
            "validationType": "IncompleteProductConfiguration"
         }
      ]
   },
   ......
}
```


In the above response, Kibo returns an encoded value for `nextCursorMark`. In the next request, set `cursorMark` to the same value you received for `nextCursorMark`:

GET `api/commerce/catalog/storefront/productsearch/search/?sortBy=productName+asc&pageSize=50&query=black&cursorMark=AoI%2fGXZlcnRpY2FsIGNvbnZlcnQgMi4wIGluc`

You can repeat this process of setting `cursorMark` to the same value you receive for `nextCursorMark` to page through all your product search results. When `nextCursorMark` is null, you've reached the end of the paged results.