SITECORE

Search and Recommendation API January 24, 2022

Discover API - Search and Recommendation

Sitecore Discover

Search and Recommendation API Specification Document

v. 3.0.43 20210523

Table of Contents

Chapter 1 Introduction 7
Chapter 2 Overview 8
2.1 Acronyms 9
2.2 Search and recommendation 10
2.2.1 Search and recommendation examples 10
2.3 Basics 14
2.3.1 API URL 14
2.3.2 API Request 14
2.3.3 API Request (Deprecated) 15
2.4 Developer Resources 17
2.4.1 API Explorer 17
2.5 Cookies and Personalization Information 18
2.6 Annotations and Terms 19
2.6.1 Angle brackets 19
2.6.2 Type of Object 19
2.6.3 Type of Value Object 19
2.6.4 Type of ID 20
2.6.5 Data structure snippets 20
2.7 Data structure in Request.data 22
2.8 Data structure in Response 23
Chapter 3 Get Started 25
3.1 Querying Content 26
3.1.1 Example: Search for red products. Return 3 items per page 26
3.1.2 Example: Search for 2 articles about movies 27
3.1.3 Example: Search for product with incomplete keyphrase 27
3.1.4 Example: Recommendation for one widget 28
3.1.5 Example: Recommendation for two widgets 29
3.1.6 Example: Recommendation with product context widget 30
3.1.7 Example: Get all possible product fields 31
3.2 Pagination 32
3.2.1 Example: Search next 3 (page 2 of) red products 32
3.3 Facet 33
3.3.1 Example: Getting 3 facet values from each facet type from product content 33
3.3.2 Example: Get 100 facets for a facet type 34
3.3.3 Example: Get price range facet with 3 buckets 35
3.3.4 35
3.3.5 Example: Getting category nested facet with 5 leaf category 35
3.3.6 Examples: Get the top 3 values for all facets from a dynamic attributes collection ... 36
3.3.7 Example: Get all values for a specific attribute from a dynamic attributes collection 37
3.3.8 Examples: Apply diamond filter on primary_stone_type for wedding keyphrase in product 40
3.3.9 Examples: Apply range between 1000 and 2500 on price for wedding keyphrase in product 41
3.3.10 Example: Filter on category Mens > Shirt for red product 42
3.3.11 Examples: Filter pants on a dynamic attribute for red products 42
3.3.12 Filter for Recommendation 43
3.3.13 Example: Filter on homepage widget for Medium or Large size products 44

3.3.14 Examples: Filter on PDP widget for Red or Blue products for french variant of the site 44
3.3.15 Example: Filter on PDP widget for products between 10 and 50 final price 44
3.3.16 Examples: Filter on cart widget for products matching a specific fitment (automotive industry) 44
3.4 Suggestion 46
3.4.1 Examples: Getting 5 keyphrase suggestions for keyphrase shirt 46
3.4.2 Example: Getting 3 trending category suggestion 47
3.4.3 Examples: Getting 3 autocomplete suggestion for mov in article 47
3.5 Appearance 49
3.5.1 Example: Get appearance templates for all sections for the default device 49
3.5.2 Example: Get appearance templates for all sections and all devices, and do not substitute variables in templates and do not add css selectors to CSS 50
Chapter 4 Detailed Structure of Request.data 53
4.1 query 54
4.1.1 Data structure 54
4.1.2 Default value 54
4.1.3 Examples 54
4.2 exact_match 55
4.2.1 Data structure 55
4.2.2 Default value 55
4.3 facet 56
4.3.1 Data structure 56
4.3.2 Default value 58
4.3.3 Examples 58
4.4 filter 60
4.4.1 Data structure 60
4.4.2 Example 60
4.5 suggestion 61
4.5.1 Data structure 61
4.5.2 Default value 61
4.5.3 Examples 61
4.6 widget_id (deprecated) 62
4.6.1 Data structure 62
4.6.2 Example 62
4.7 widget 63
4.7.1 Data structure 63
4.7.2 Examples 63
4.8 content 64
4.8.1 Data structure 64
4.8.2 Default value 64
4.8.3 Example 64
4.9 appearance 66
4.9.1 Data structure 66
4.9.2 Examples 68
4.10 batch 69
4.10.1 Data structure 69
4.10.2 Example 69
4.11 sort 70
4.11.1 Data structure 70
4.11.2 Default value 70

4.11.3 Example 70
4.12 n_item 71
4.12.1 Data structure 71
4.12.2 Default value 71
4.12.3 Max value 71
4.12.4 Example 71
4.13 page_number 72
4.13.1 Data structure 72
4.13.2 Default value 72
4.13.3 Example 72
4.14 preview (internal) 73
4.14.1 Data structure 73
4.14.2 Default value 73
4.14.3 Examples 74
4.15 context 75
4.15.1 Data structure 75
4.15.2 Example 75
4.16 request_for 76
4.16.1 Data structure 76
4.16.2 Example 76
4.17 context_values (Coming Soon) 77
4.17.1 Data structure 77
4.17.2 Example 77
Chapter 5 Detailed Data Structure of Response 78
5.1 ts 79
5.1.1 Data structure 79
5.2 dt 80
5.2.1 Data structure 80
5.3 err 81
5.3.1 Data structure 81
5.4 msg 82
5.4.1 Data structure 82
5.5 rid 83
5.5.1 Data structure 83
5.6 query2id 84
5.6.1 Data structure 84
5.7 request_for 85
5.7.1 Data structure 85
5.8 facet 86
5.8.1 Data structure 86
5.8.2 Examples 86
5.9 Facet_names 89
5.9.1 Data structure 89
5.9.2 Example 89
5.10 filter 90
5.10.1 Data structure 90
5.11 suggestion 91
5.11.1 Data structure 91
5.11.2 Example 91
5.12 autocomplete 92
5.13 content 93

5.13.1 Data structure 93
5.13.2 Example 93
5.14 redirect url 94
5.14.1 Data structure 94
5.15 widget_title (Deprecated - this support has been removed) 95
5.15.1 Data structure 95
5.16 widget 96
5.16.1 Data structure 96
5.17 n_item 97
5.17.1 Data structure 97
5.18 total_item 98
5.19 page_number 99
5.20 total_page 100
5.21 url 101
5.22 batch 102
5.22.1 Data structure 102
5.22.2 Example 102
5.23 errors 103
5.23.1 Data structure 103
5.23.2 Example 103
5.24 context_values 104
5.24.1 Data structure 104
5.25 appearance 105
5.25.1 Data structure 105
5.25.2 Examples 105
Chapter 6 Avanced Use Cases 107
6.1 Limiting Returned Content Attribute 108
6.1.1 Example: Search for 2 articles about movies, only return the title 108
6.2 Querying Mixed Content 109
6.2.1 Example: Search for the keyphrase *new* in *product* and *article* 109
6.3 Content Result Sorting 110
6.3.1 Example: Sort by Name from A to Z 110
6.3.2 Example: Sort by Price from High to Low 110
6.3.3 Example: Sort by both price and name, first Price from High to Low, if the price is the same sort by name from A to Z 111
6.4 Boolean Operation 113
6.5 Boolean Operation in Query (Deprecated) 114
6.5.1 Example: Search for 3 products with red OR shorts 114
6.5.2 Example: Search for 3 products with red AND shorts 114
6.6 Facet Order 116
6.6.1 Example: Getting 3 facets from each facet type from product content. Sort all facet by text in ascending order (alphabetical) 116
6.6.2 Example: Getting 3 facets from each facet type from product content. Sort color facet by count in descending order 117
6.7 Facet Type Control 119
6.7.1 Example: Get 3 facet from color facet from product content 119
6.7.2 Example: Get only two facets from product content color and product_type 119
6.8 Facet Option Control 121
6.8.1 Example: Getting only white, red, black facet from color facet 121
6.8.2 Example: Getting 3 fixed facet (white,black, grey) from color facet regardless of item count 122

6.8.3 Example: Getting 3 fixed facet (white, red, black) from color facet and 2 dynamic color 122
6.8.4 Example: Getting 3 color facet where the facet have at least 10 item in it 123
6.8.5 Example: Getting total number of facet values for all facets 123
6.8.6 Example: Getting total number of facet values for one facet only 124
6.9 Boolean Operation in Filter (Deprecated) 126
6.9.1 Example: Filter on (red OR blue) color for shirt keyphrase in product 126
6.9.2 Example: Filter on (red AND blue) color for shirt keyphrase in product.................... 127
6.9.3 Example: Filter on ((red AND blue) OR yellow) color for shirt keyphrase in product127
6.10 Context.. 129
6.10.1 Example: Search for 3 products with red AND shorts with user context (gender information).. 129
Chapter 7 Appendix A: Types.. 130
7.1 Query Types 131
7.2 Content Types 132
7.3 Suggestion Types... 133
7.4 Facet/ Filter Types.. 134
7.5 Sort Types 135
Chapter 8 Appendix B: Context 136
8.1 Context Types.. 137
8.2 Context - User 138
8.3 Context - Browser 139
8.4 Context - Geo 140
8.5 Context - Store 142
8.6 Context – Fitment 143
8.7 Context - Page 145
8.8 Context - Campaign... 146
8.9 Context - Channel 147
8.10 Context - Weather (Deprecated).. 148
Chapter 9 Appendix C: Error 149
9.1 Error Format.. 150
9.2 Error Type 151
9.3 HTTP Status 154
Chapter 10 Appendix D: Internal or Load Test Requests 156
10.1 Flag Specification 157
10.2 Sample JS Implementation using a cookie 159

Chapter 1 Introduction

The Sitecore Discover Search and Recommendation API provides functionality to search and recommend any content you provide, including products, categories, stores, articles, banners, and more. It honors all rules that are setup through the portal including synonyms, content rules (for example, to boost, suppress, include or exclude specific content), redirect rules enabling redirects to specific landing pages, and so on...

Chapter 2 Overview

The Sitecore Discover Search and Recommendation API provides:

- Suggestions – Guidance and tips for next search query. Includes features for autocomplete, dynamic spelling correction, query suggestions, and related searches.
- Facets - Available content attributes to further narrow results.
- Filter - Selected facets applied to content results.
- Content Results - Search results for all requested type of content.
- Sort - Ordering of content results based on different attributes or algorithms.
- Multiple widget results: Ability to serve multiple widgets on a given page.
- Product Listing Pages: Ability to configure dynamic pages where recommended content can be filtered and grouped by any attribute of user's choice.
- Personalization - Results based on user purchase intent (e.g. based on user's behavior in real-time, past, or aggregated user behavior).
- Appearance - Information to render widget in browser.

2.1 Acronyms

Acronym	Description
CTR	Click through rate
CVR	Conversion rate
DPR	Direct Purchase Revenue
DOR	Direct Order Revenue
DPU	Direct Purchase units
DO	Direct Orders

2.2 Search and recommendation

The same API supports results that you can use for preview search, full page search, category page search, landing page search, special occasion page, similar items recommendations, you may also like recommendations, html content, banners and more.

The interaction involves JSON objects in a Request-Response format.

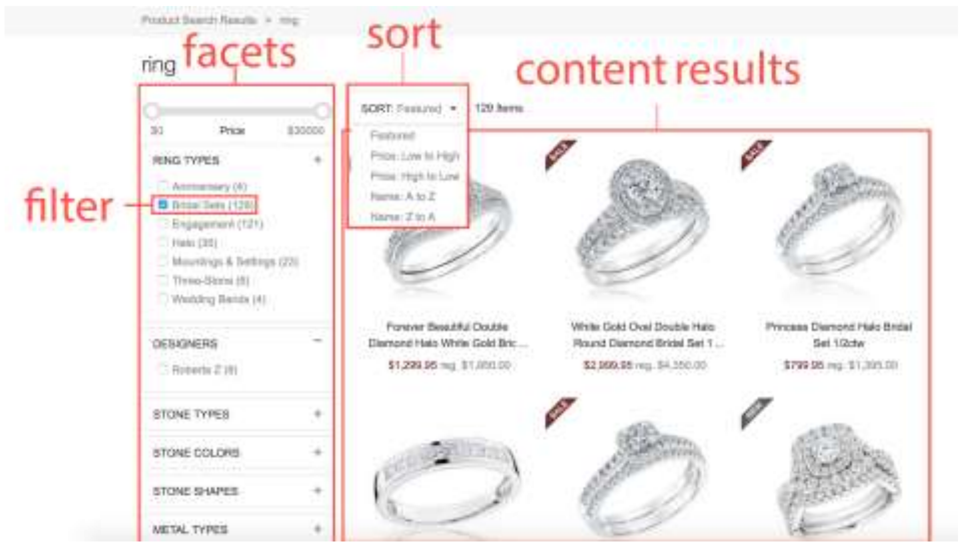
A *request* is a JSON object sent by the API user to provide information about a specific query to receive specific information about available facets on a search query, possible suggestions to direct the next query in different contents, and available matching content results.

A *response* is a JSON object sent by the Discover search service. It will include a set of available facets on the query, possible suggestions, and content results matching the request.

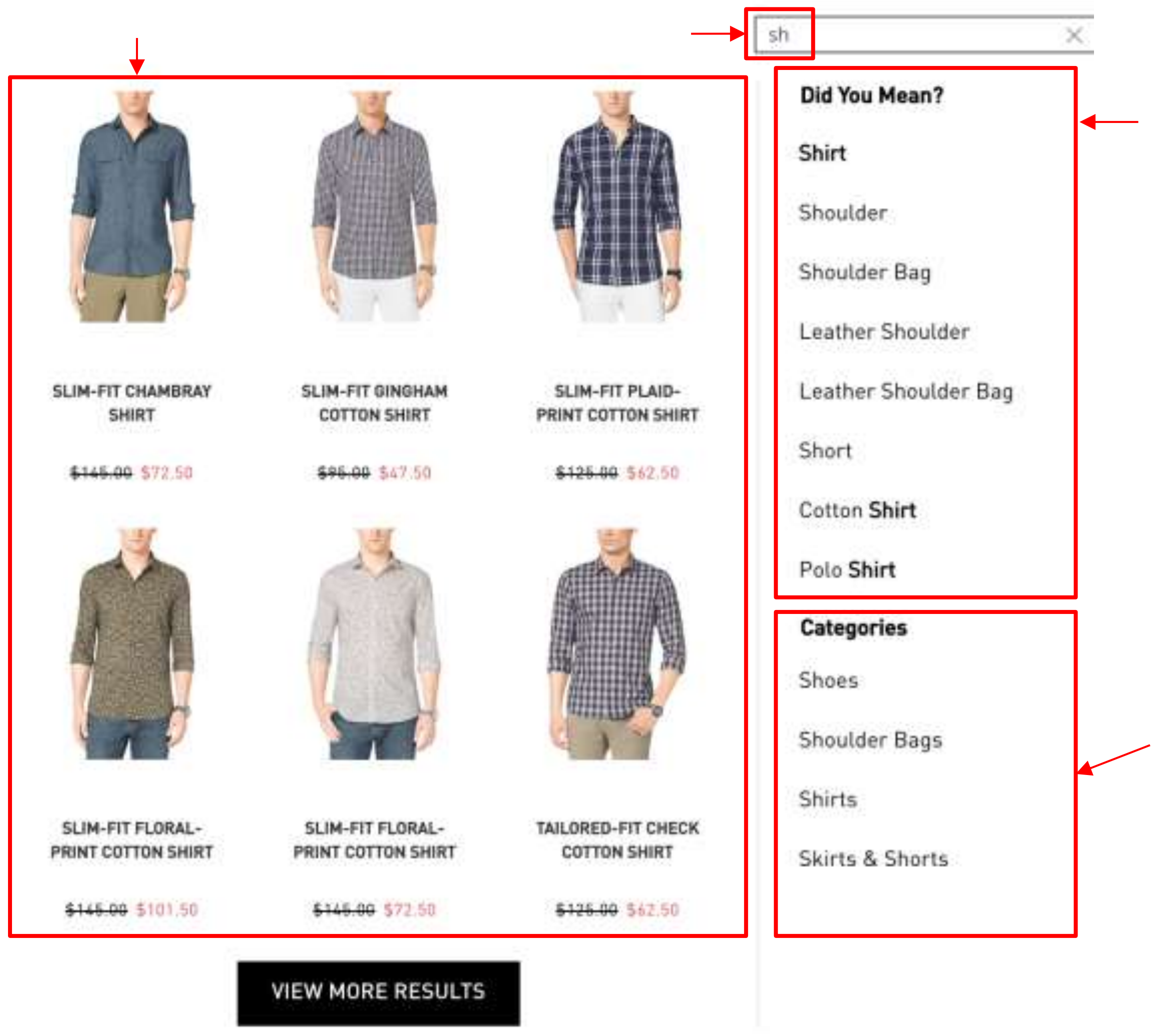
2.2.1 Search and recommendation examples

This section show some examples of search and recommendation you can achive on your website with the Sitecore Discover Search and Recommendation API.

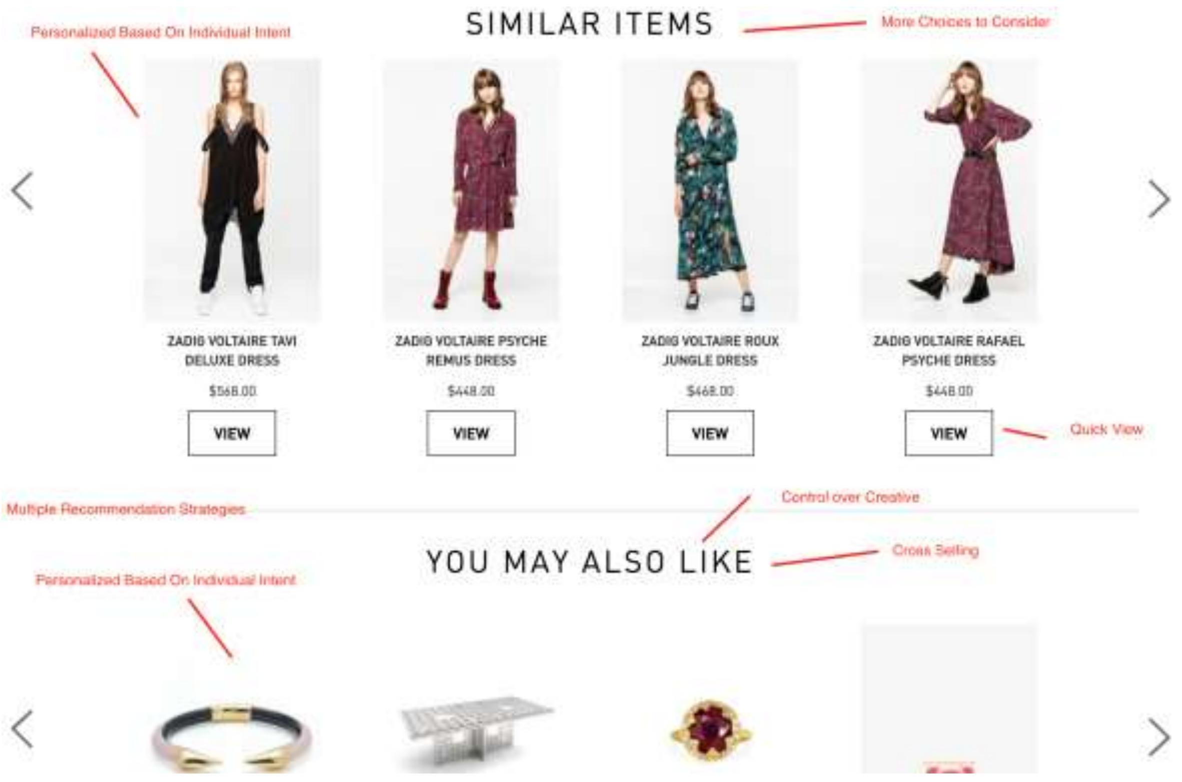
Search Page with Filtering



Preview Search



Multiple Widget Recommendations


Personalized Based On Individual Intent
SIMILAR ITEMS
More Choices to Consider
ZADIG VOLTAIRE TAVI DELUXE DRESS
$568.00
VIEW
ZADIG VOLTAIRE PSYCHE REMUS DRESS
$448.00
VIEW
ZADIG VOLTAIRE ROUX JUNGLE DRESS
$468.00
VIEW
ZADIG VOLTAIRE RAFAEL PSYCHE DRESS
$448.00
VIEW
Quick View
Multiple Recommendation Strategies
Control over Creative
YOU MAY ALSO LIKE
Cross Selling
Personalized Based On Individual Intent

Product Listing Pages

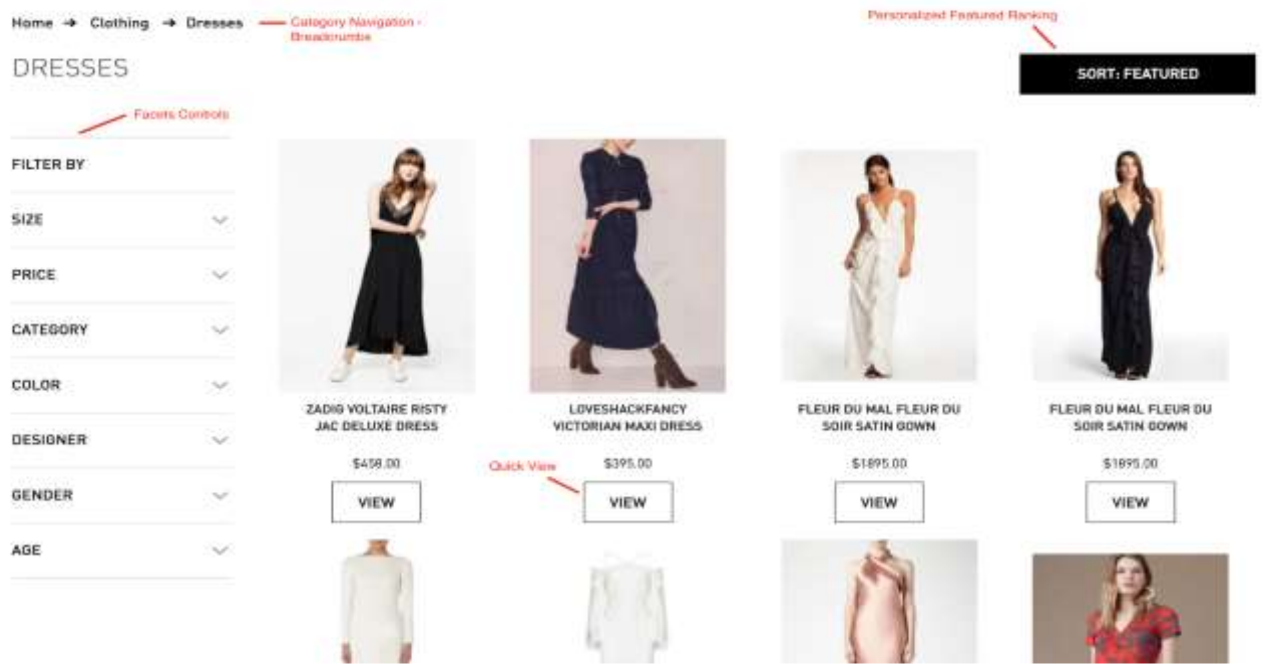
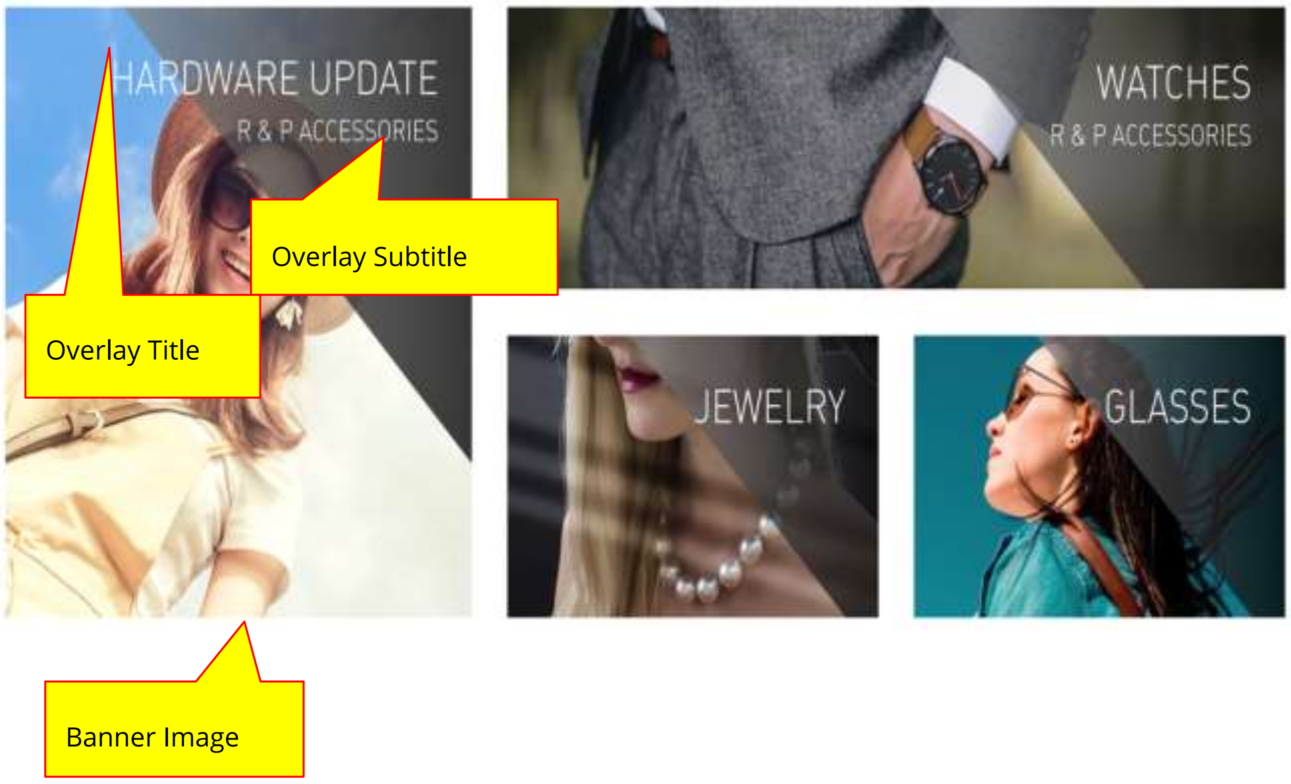
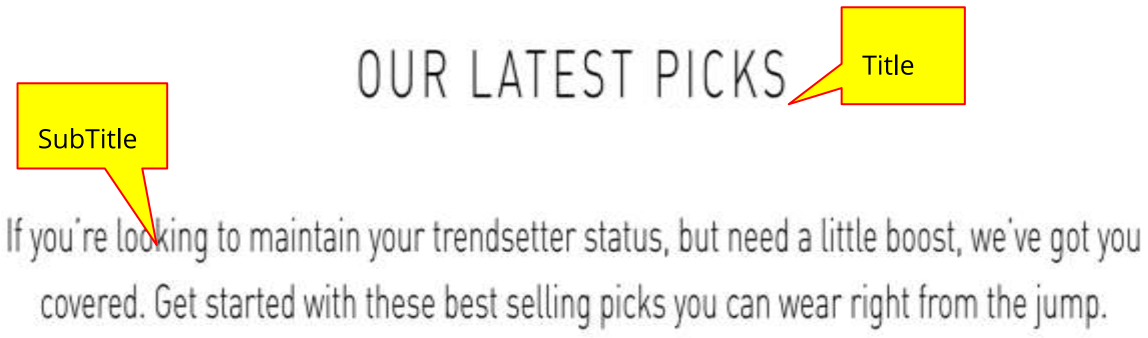

Home → Clothing → Dresses
Category Navigation - Breadcrumbs
Personalized Featured Ranking
DRESSES
SORT: FEATURED
Facets Controls
FILTER BY
SIZE
PRICE
CATEGORY
COLOR
DESIGNER
GENDER
AGE
ZADIG VOLTAIRE RISTY JAC DELUXE DRESS
$458.00
VIEW
Quick View
LOVESHACKFANCY VICTORIAN MAXI DRESS
$395.00
VIEW
FLEUR DU MAL FLEUR DU SOIR SATIN GOWN
$1895.00
VIEW
FLEUR DU MAL FLEUR DU SOIR SATIN GOWN
$1895.00
VIEW

Image Banner with overlay title and subtitle



Html Block Title and Description


OUR LATEST PICKS
Title
SubTitle
If you're looking to maintain your trendsetter status, but need a little boost, we've got you covered. Get started with these best selling picks you can wear right from the jump.

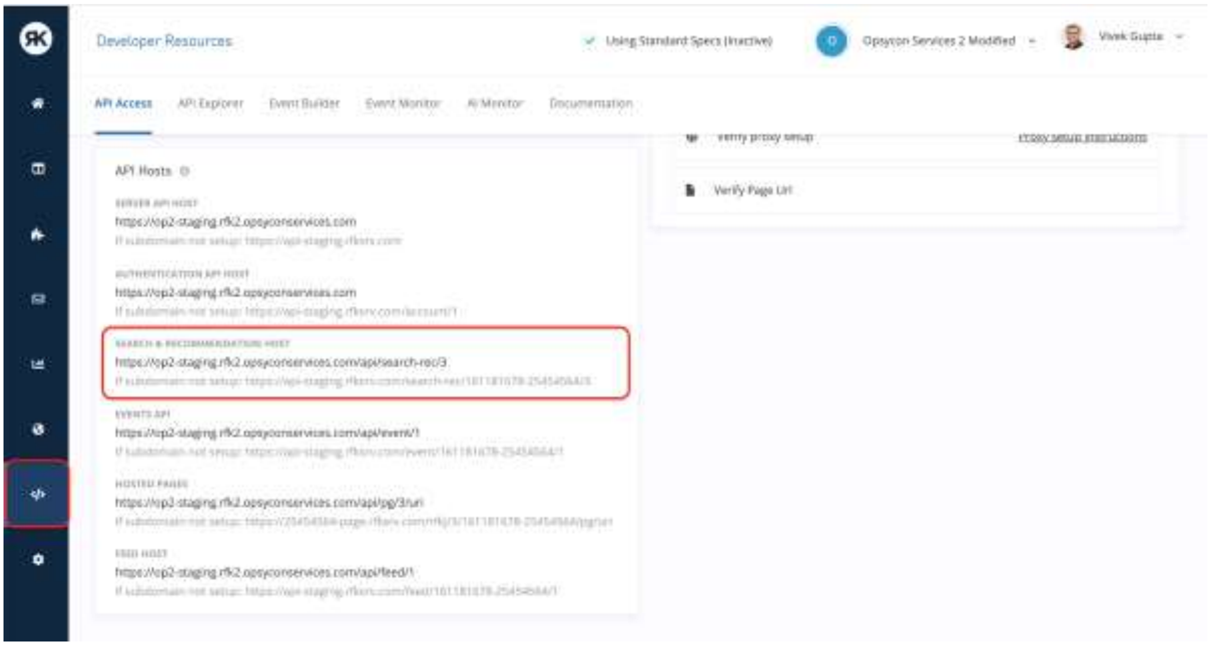
2.3 Basics

Sitecore Discover supports both HTTP and HTTPS protocols for the APIs. You can access APIs using GET or POST methods.

Our APIs accept requests in the form of JSON object. All information of inquiry to the APIs must be in a key data. The value of the key can either be a JSON string or a url encoded JSON string.

2.3.1 API URL

You can obtain the API URL from the Customer Engagement Console, under **Developer Resources > API Access**.



API Hosts

SERVER API HOST
https://op2-staging.rfk2.opsyconservices.com
If subdomain not setup: https://api-staging.rfksrv.com

AUTHENTICATION API HOST
https://op2-staging.rfk2.opsyconservices.com
If subdomain not setup: https://api-staging.rfksrv.com/account/1

SEARCH & RECOMMENDATION HOST
https://op2-staging.rfk2.opsyconservices.com/api/search-rec/3
If subdomain not setup: https://api-staging.rfksrv.com/search-rec/161181678-2545454/3

EVENTS API
https://op2-staging.rfk2.opsyconservices.com/api/event/1
If subdomain not setup: https://api-staging.rfksrv.com/event/161181678-2545454/1

HOSTED PAGES
https://op2-staging.rfk2.opsyconservices.com/api/pg/3/url
If subdomain not setup: https://2545454-page.rfksrv.com/rfkj/3/161181678-2545454/pg/url

FEED HOST
https://op2-staging.rfk2.opsyconservices.com/api/feed/1
If subdomain not setup: https://api-staging.rfksrv.com/feed/161181678-2545454/1

2.3.2 API Request

Method: Get or Post

Required keys:

Key	Data type	Description
data	<JSON string> OR <url encoded json string>	JSON that represents the request. If passed as a GET query parameter, the JSON object is html decoded before processing the request.

Example:

```
GET {API_URL}?data={<your_key_value_data>}
```

```
POST {API_URL}
{
  "data": "{<your_key_value_data>}"
}
```

Example in Python (Using Requests library):

```
import requests
import json

url = 'API_URL'
data = {
    "data": '{"query": {"keyphrase": ["example"]},"content": {}}'
}

response = requests.post(url, params=data)
print json.loads(response.text)
```

Note:
You must have set up a dedicated subdomain to access Sitecore Discover services. Contact Reflektion support for details.

2.3.3 API Request (Deprecated)

Required keys:

Key	Data type	Description
api_key	<base64 string>	An access token provided by Sitecore.
data	<JSON string>	JSON that represents the request.

When calling from the server, use the API key. The API key does not require an access token to make API calls. Set the following header:

```
Host: api.rfksrv.com
Header: Authorization: : <api-key>
```

`The <api-key>` can be assigned different scopes. A scope to call `/search-rec` and to call `/account/<version>/access-token`.

When calling from the client (browser), we recommend you use an access token that you obtain by using api-key and passing it to the client (browser):

```
Host: api.rfksrv.com
Header: Authorization: Bearer <access-token>
```

Note:
Some customers use api-key from browsers for public Sitecore Discover APIs. It is not recommended, and at your own risk.

Required keys:

Key	Data type	Description
api-key	string	An API key provided by Sitecore. This can be used to make calls from server. It should not be used to call from browser.

access-token	string	Access token obtained by calling `/account/<version>/access-token` using your api-key with auth scope from your server. You must use this token to make calls from browser in conjunction with your account-key. These tokens are short-lived.
data	<json string> OR <url encoded json string>	JSON that represents the request.

Example:

```
        GET {server_name}.rfksrv.com/search-
rec/{customer_key}/{version}?data={<your_key_value_data>}

        Header:
        Authorization: "<your api-key>"
        OR
        Authorization: Bearer <your access-token>
```

```
        POST {server_name}.rfksrv.com/search-rec/{customer_key}/{version}
        {
          "data": "{<your_key_value_data>}"
        }

        Header:
        Authorization: "<your api-key>"
        OR
        Authorization: Bearer <your access-token>
```

To obtain access token:

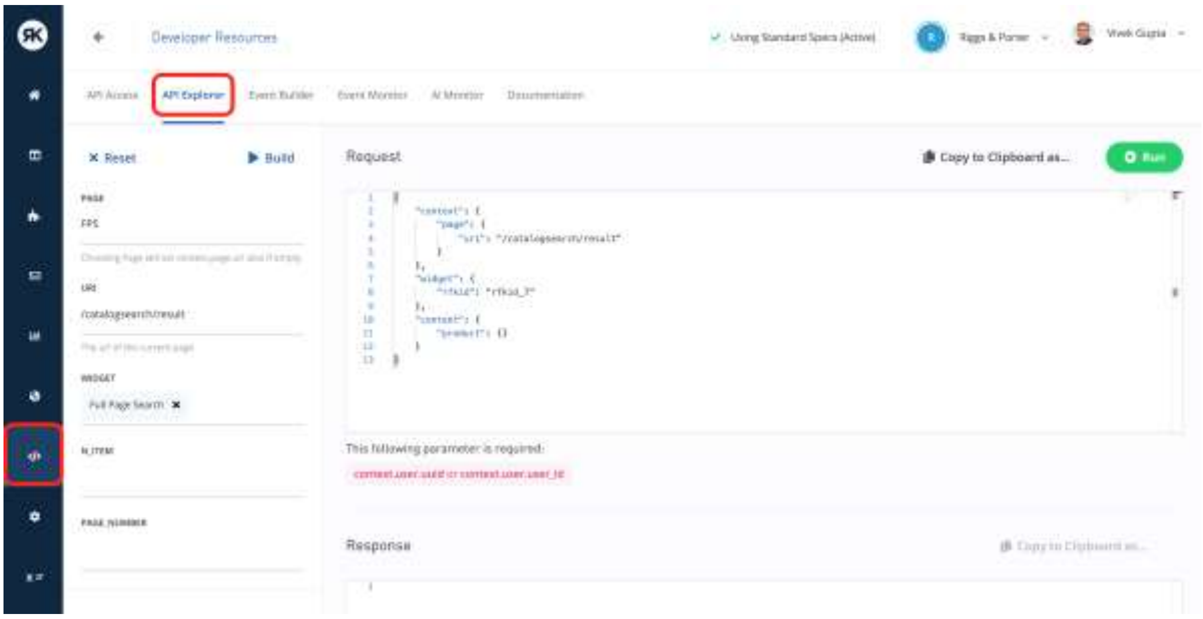
```
        POST {server_name}.rfksrv.com/account/1/access-token
        {
          "data": '{"scope":["search-rec"]}'
        }

        Header:
        Content-Type: application/json
        x-api-key: "<your api-key>"

        Curl example:

        curl -X POST https://api.rfksrv.com/account/1/access-token \
          -H 'Content-Type: application/json' \
          -H 'x-api-key: <api-key> \
          -d '{"scope":["search-rec"]}'
```

2.4 Developer Resources

2.4.1 API Explorer

Sitecore Discover provides tools within the Customer Engagement Console (CEC) to help you easily create precise request data, and also allows you to run the requests to ensure you are sending it correctly and receiving the expected results. In the CEC, you access **Developer Resources** -> **API Explorer**

The following shows a screenshot of the **API Explorer**.



PAGE
FPS
Choosing Page will set context.page.url and rfkid, if empty

URI
/catalogsearch/result
The url of the current page

WIDGET
Full Page Search

N_ITEM

PAGE_NUMBER

```
{
    "context": {
        "page": {
            "uri": "/catalogsearch/result"
        }
    },
    "widget": {
        "rfkid": "rfkid_7"
    },
    "content": {
        "product": {}
    }
}
```

This following parameter is required:
context.user.uuid or context.user.user_id

2.5 Cookies and Personalization Information

If you are making an API request from your server, then you must pass through several pieces of information to help Sitecore Discover personalize the experience for the user.

These include:

- `Uri`: Uri of the page the user is requesting.
- `Referrer`: referring uri (referrer) you received as part of the header in the request.
- `IP`: ip address of the user, i..e IP address of the request you received form user's browser or native application.
- `User Agent`: User agent of the browser that you received in the header of the request.
- `Cookie`: All cookies Sitecore Discover sets on the customer's domain Discover. All Sitecore Discover cookes cookies start with `__r` (two underscores characters, followed by the letter *r*)**.** These cookies are required to:
    - Track the user session
    - Preview settings if the customer (merchandiser) wants to change configuration in the Customer Engagement Console and test its impact on the website before publishing it.
    - Ramping up traffic on your website when moving to Sitecore Discover from another vendor.
- `uuid` or `user_id`: Usually, if you pass the cookies, you would have also passed the __ruid which contains the anonymous user id that Sitecore Discover has generated if you have included the beacon as part of your website. In that case, passing uuid is optional. However, if for any reason you have not included the beacon, you must pass a user identifier that you have generated and maintain to track the user anonymously.

Following is a sample request (cookies not shown):

```
        {
           "context": {
               "page": {
                   "uri": "/category/men/shirts?color=red",
                   "referrer": "https://riggsandporter.com/category/men?color=red"
               },
               "user": {
                   "uuid": "125757321-ph-eb-4w-1p-tvfxk0rs1iwgp5tsgdvt-1523635809203"
               },
               "geo": {
                   "ip": "20.230.240.14"
               },
               "browser": {
                   "user_agent": "user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 11_1_0)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90 Safari/537.36"
               }
           }
        …
```

2.6 Annotations and Terms

2.6.1 Angle brackets

In the JSON snippets provided in this document, we use angle brackets <> to denote the explanation of the value.

For example, the following means that the string should be a content type.

```
"<content_type>"
```

2.6.2 Type of Object

Object is a dictionary of key-value pairs. The key is a string, the data structure of the value have different types depending on the key. The data structure of the value could be an object, list, string or number (float or integer).

The following is an example of an object

```
{
  "<key_1>": <value_1>,
  "<key_2>": <value_2>
}
```

2.6.3 Type of Value Object

Value object is a dictionary which contains a key-value where key is value and value is an array of items. There might be other keys in the value object. We refer to this structure as *value object* in this document.

```
{
  "value": [
    <item_1>, <item_2>, ...
  ],
  "<key_1>": <value_1>
}
```

Short Representation

The value object can be represented in a short form if *value* is the only key in the value object. The short representation allows us to put the value of the value object without the enclosing object.

Full Representation	Short Representation
`"<key>": { "value": [ <value_1>, <value_2> ] }`	`"<key>": [ <value_1>, <value_2> ]`

`"<key>": {`
`  "value": [`
`    <value_1>`
`  ]`
`}`
	`"<key>": <value_1>`

Overwriting values

If the value object contains a list of objects, the keys in the object will overwrite the key in the enclosing object (with some exception that will be specified).

In the following example, type A has a `value_2` for `key_1` while type B uses the enclosing object's value (`value_1`).

```
        {
          "value": [
            {"type": "A", "<key_1>": <value_2>},
            {"type": "B"}
          ],
          "<key_1>": <value_1>
        }

        In the following example, product is sorted in ascending order while article is sorted
in descending order.

        Example
        {
          "value": [
            {"type": "product", "sort": "asc"},
            {"type": "article"}
          ],
          "sort": "desc"
        }
```

2.6.4 Type of ID

Certain keys in both request and response may have values with the type of <id> which is a string with no readable meaning. Type `<id>` refers to any ID specified from API that might be used or referred in subsequent requests.

2.6.5 Data structure snippets

Snippets in Detailed Data Structure sections, contain blue symbols to indicate structure.

The data structure snippets use three different symbols: **?**, **+**, and *****.

The `?` symbol indicates an optional group, that is, you can have 0 or 1 of the group contained within the square brackets [].Following is an example of an optional key symbol:

```
{
  ["<optional_key>": "<value>" ]?
}
```

The + symbol indicates a required group, that is, you must have 1 or more of the group contained within the square brackets `[]`. Following is an example of a list that must contain at least one <value>:

```
        {
          "<key>": [["<value>" ]+]
        }
```

The `*` symbol indicates that the group contained within the square brackets `[]` is optional and that you can have multiple instances, that is, you can have 0 or more of this group. Following is an example of a list which can be empty, or contain any number of <values>:

```
        {
          "<key>": [["<value>" ]*]
        }
```

2.7 Data structure in Request.data

The following is a list of keys that a request can contain:

- query: object that provides information about type and values of inquiries

  Note: The following keys are related to the request for query: exact_match: flag to only search for exact text
- filter: object provides filtering information that further narrows query results
- facet: object to request for receiving specific facets to be used for filtering in subsequent inquiries
- suggestion: object to request for receiving suggestions on query in different types
- widget_id: id of the widget assigned by Sitecore Discover that represents a specific real estate on the website to display search or recommendations (deprecated).
- widget: contains two ids: the rfkid and the widget_id. You need to specify only rfkid that represents a specific real estate on the website to display search or recommendations.
- batch: used only if multiple requests are grouped together into one request. A list of objects which can be used to override specific request key/values. Note: You must use batch for requesting results for multiple recommendation widgets on the same page to avoid duplicate responses.
- content: object to provide information regarding expected content in result of search. Followings keys are related to the request for content:
  - sort: objects to direct sorting of the items in response
  - n_item: total number of items expected per page to be returned in response
  - page_number: page number of the results that will be returned
- appearance: object to request appearance information including html, css and javascript templates and variables associated with the given widget(s).
- preview: object to indicate this request is a preview request
- context: object of additional dimensions that may affect the search results
  - user: information for user including user id, gender, email
  - browser: browser information including device, user agent, screen size
  - geo: geographic information of the user
  - store: store information if you want to limit the search to a specific store's inventory
  - fitment: fitment information representing user's fitment choice.
  - weather: weather information (Deprecated)
  - page: the page context such as referrer, page url, page title, list of skus (for cart page)
  - channel: the channel type such as notification bar, chatbot, web page
  - campaign: the paid campaign information to tie campaigns to rules
- context_values: object to request what keys from the current request context to return in the response

2.8 Data structure in Response

The following lists the keys in a response:

- ts: time of the server in millisecond (UTC)
- dt: response processing time, integer in millisecond
- err: any error message (Deprecated)
- msg: any special message for response (Deprecated)
- rid: response id, unique generated string
- query2id: object of unique b64 ids for each type in query
- result_for: object to show what is the result for
- facet: objects for available facets
- facet_names: strings representing the names of available facets
- filter: objects for selected filters (if any)
- suggestion: object for suggestion for types provided
- autocomplete: autocomplete keyphrase suggested (only returned with suggestion)
- content: result items for different types requested
- appearance: html/css templates and variables for rendering a widget

The following query.keyphrase related keys are present:

- redirect_url: url that the search should redirect to, depending on the keyphrase

The following widget related keys are present:

- widget_title: title of the widget returned (Deprecated)
- widget: the widget that was used to generate this response
- experiment_id: experiment_id, unique experiment identifier representing the split test entity in Customer Engagement Console.
- experiment_bucket: experiment_bucket, unique generated string representing a bucket to which the user is assigned to.
- variation_id: variation id, unique identifier representing the variation in Customer Engagement Console.

The following content related keys are presents along with `content` key:

- n_item: number of content items reported
- total_item: total number of content items available
- page_number: current page number of the content results
- total_page: number of content pages available
- url: url that represents this search request (human readable)

- batch: returned only if requests were batched. A list of response key/values that are specific to corresponding batched requests. It is a list of objects which can be used to override response specific parameters.
- errors: returned if any errors were encountered while processing the request. A list of all such errors, with a message, description, severity level and code for each of them is returned.
- context_values: context values requested in the request.

Chapter 3 Get Started

This section provides information to help you get started using the the Sitecore Discover Search and Recommendation APIs.

3.1 Querying Content

This section explains the basics of querying different content.

To query any content, the following two keys are required:

- The `query` key specifies the query object that includes query string.
- The `content` key specifies the content type (default: product) that the `query` applies to.

The `n_item` is an optional key that specifies the total number of items per page to be returned in the response. The default is 10.

This section includes the following examples of basic content queries:

- Querying product content
- Querying other content
- Search for product with incomplete keyphrase
- Recommendation for one widget
- Recommendation for two widgets
- Recommendation with product context widget
- Get all possible product fields

For more advanced use cases, see:

- Boolean Operation in Query
- Limiting Returned Content Attribute
- Querying Mixed Content
- Querying based on location availability
- Content Result Sorting

3.1.1 Example: Search for red products. Return 3 items per page

Request:

```
{
  "query": {
    "keyphrase": ["red"]
  },
  "n_item": 3,
  "content": {}
}
```

If there is no `<type>` defined in `content` key, the default value is used. The default value for `content` is product.

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_q1",
  "url": "/searchpage?keyphrase=red",
  "query2id": {
    "keyphrase": "keyphrase_id_red"
  },
  "content": {
    "product": {
      "value": [
        {"name": "shorts", "color": "red", "price": 25,
         "image_url": "", "rating": 5 },
        {"name": "red hat", "price": 10, "rating": 1},
        {"name": "pants", "price": 50, "category_name": ["red"] }
      ]
    }
  },
  "n_item": 3,
  "total_item": 5,
  "page_number": 1,
  "total_page": 2
}
```

The response returns any product that contains red in any attribute.

In this example, there are 2 pages of results in total. To query the next page, refer to Pagination.

Each response returns adefault set of attributes for an item (in this example we just reported a few attribute keys for simplicity). For instructions to get a different subset of the attributes in the response, refer to Limiting Returned Content Attribute.

3.1.2 Example: Search for 2 articles about movies

Request:

```
          {
            "query": {
              "keyphrase": ["movies"]
            },
            "content": {
              "article": {}
            },
            "n_item": 2
          }
```

Response:

```
          {
            "ts": 1480977544,
            "rid": "response_id_q2",
            "url": "/searchpage?keyphrase=movies&content=article",
            "query2id": {
              "keyphrase": "keyphrase_id_movies"
            },
            "content": {
              "article": {
                "value": [
                  {"title": "Top 10 movie for 2016", "author": "John Smith", "published_data":
"2016-12-31"},
                  {"title": "Finding memo review", "category_name": "movie review"}
                ]
              }
            },
            "n_item": 2,
            "total_item": 3,
            "page_number": 1,
            "total_page": 2
          }
```

3.1.3 Example: Search for product with incomplete keyphrase

If the specified keyphrase in the query yields no results, Sitecore Discover can look for results matching a suggested keyphrase. You must explicitly ask for suggestions in the request to return results for suggested keyphrase by including `suggestion` parameter. You may also ask to return query by including `request_for` parameter. In this case, the response will contain the parameter `request_for`, that details this strategy.

Request:

```
        {
          "query": {
            "keyphrase": ["shor"]
          },
          "content": {},
          "n_item": 2,
          "suggestion": {
            "keyphrase": {
              "max": 1
            }
          },
          "request_for": ["query"]
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_q2",
          "url": "/searchpage?keyphrase=shorts",
          "query2id": {
            "keyphrase": "keyphrase_id_shorts"
          },
          "suggestion": {
         "keyphrase": [
             {
                 "text": "short",
                 "in_content": "product",
                 "id": "suggestion_idc2hvcnQ="
             }
         ]
    },
          "content": {
            "product": {
              "value": [
                {"name": "swim short", "price": 25, "image_url": "", "rating": 5 },
                {"name": "casual short", "price": 10, "rating": 1}
              ]
            }
          },
          "request_for": {
            "query": {
              "keyphrase": "short",
              "orig_keyphrase": "shor"
            },
            "content": {},
            "n_item": 2
          },
          "n_item": 2,
          "total_item": 32,
          "page_number": 1,
          "total_page": 16
        }
```

3.1.4 Example: Recommendation for one widget

Request:

```
        {
          "widget": {
            "rfkid": "rfkid_1"
          },
          "content": {"product": {}},
          "n_item": 2
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_q2",
          "content": {
            "product": {
              "value": [
                {"name": "Opaque earrings", "id": "1024"},
                {"name": "Hanging earrings", "id": "1023"}
              ]
            }
          },
          "widget": {
            "rfkid": "rfkid_1"
          },
          "n_item": 2,
          "total_item": 2,
          "page_number": 1,
          "total_page": 1
        }
```

3.1.5 Example: Recommendation for two widgets

Request:

```
        {
           "batch": [
               {
                 "widget": {"rfkid": "rfkid_1"}
               },
               {
                 "widget": {"rfkid": "rfkid_2"}
               }
           ],
          "content": {"product": {}},
          "n_item": 2
        }
```

In this request the `widget` is overridden in the batch object.

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_q2",
          "batch": [
                {
                  "content": {
                     "product": {
                        "value": [
                             {"name": "Opaque earrings", "id": "1024"},
                             {"name": "Hanging earrings", "id": "1023"}
                        ]
                     }
                  },
                  "widget": {"rfkid": "rfkid_1"}
               },
               {
                 "content": {
                    "product": {
                       "value": [
                            {"name": "Opaque necklace", "id": "1022"},
                            {"name": "Hanging necklace", "id": "1021"}
                       ]
                    }
                 },
                 "widget": {"rfkid": "rfkid_2"}
              }
          ]
          "n_item": 2,
          "total_item": 2,
          "page_number": 1,
          "total_page": 1
        }
```

3.1.6 Example: Recommendation with product context widget

Request:

```
        {
          "widget": {
            "rfkid": "rfkid_2"
          },
          "context": {"page": {"sku": ["sku1", "sku2"]}},
          "content": {"product": {}},
          "n_item": 2
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_q2",
          "widget_title": "Your title here",
          "widget": {
            "rfkid": "rfkid_2"
          },
          "content": {
            "product": {
              "value": [
                {"name": "Opaque earrings", "id": "1024"},
                {"name": "Hanging earrings", "id": "1023"}
              ]
            }
          },
          "n_item": 2,
          "total_item": 2,
          "page_number": 1,
          "total_page": 1
        }
```

3.1.7 Example: Get all possible product fields

You can use the `get_fields` flag within content to obtain a list of all possible fields in a product. Note that some products may not have every field out of all the possible fields (ex. Shoe products may not have hat size, etc.)

Request:

```
{
    "content": {"product": {"get_fields": true}},
    "n_item": 0
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_712370513",
  "content": {
    "product": {
      "fields": [
        "name",
        "price",
        "image_url",
        "rating",
        "shoe_size"
      ]
    }
  },
  "n_item": 0,
  "total_item": 5,
  "page_number": 1,
  "total_page": 5
}
```

3.2 Pagination

There might be multiple pages of results for each query. This section will explain how to retrieve different pages of the response.

Below is a continuation of the Search for red products. Return 3 items per page example.

3.2.1 Example: Search next 3 (page 2 of) red products

Request

```
{
  "query": {
    "keyphrase": ["red"]
  },
  "n_item": 3,
  "page_number": 2,
  "content": {}
}
```

Response

```
{
  "ts": 1480977544,
  "rid": "response_id_q3",
  "url": "/searchpage?keyphrase=red&page=2",
  "query2id": {
    "keyphrase": "keyphrase_id_red"
  },
  "content": {
    "product": {
      "value": [
        {"name": "jacket", "color": "red", "price": 80},
        {"name": "red dress", "price": 100}
      ]
    }
  },
  "n_item": 2,
  "total_item": 5,
  "page_number": 2,
  "total_page": 2
}
```

Although 3 items per page are requested, there are only 2 product items in the response. This is because there are only 5 items in total and we are requesting the last page, which will return the remaining items.

3.3 Facet

Different search queries dynamically return facets. These facets are based on product attributes as options you can filter on. Facets are returned in different formats (term, range, nested) based on the nature of the `facet_type`.

Facet formats:

- Term - String values
- Range - Numerical values with minimum and maximum
- Nested - String values in hierarchical structure

Examples:

- Get facet from all facet type
- Range facet
- Nested facet

The examples provided are only the basic use cases, refer to the Detailed Data Structure section for complete structure.

For more advanced use cases, refer to the corresponding section:

- If a different sort order is required, refer to Facet Order.
- If only specific facet types are required, refer to Facet Type Control.
- If a fixed set of facet values are required, refer to Facet Option Control.

3.3.1 Example: Getting 3 facet values from each facet type from product content

In this example, it is assumed that there are only four facet types (color, product_type, primary_stone_type, primary_metal_type). In practice, there are likely to be more than four facet types.

Request:

```
{
  "facet": {
    "all": true,
    "max": 3
  }
}
```

The `all` flag needs to be 1 because we want all the facet_type. If all the facet values are required, you can specify `max` to be -1.

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_f1",
  "facet": {
    "colors": {
      "value": [
        {"id": "color_id_red", "text": "Red", "count": 23},
        {"id": "color_id_blue", "text": "Blue", "count": 11},
        {"id": "color_id_yellow", "text": "Yellow", "count": 10}
      ]
    },
    "product_type": {
      "value": [
        {"id": "prod_type_id_ring", "text": "Ring", "count": 105},
        {"id": "prod_type_id_necklace", "text": "Necklace", "count": 45},
        {"id": "prod_type_id_watch", "text": "Watch", "count": 5}
      ]
    },
    "primary_stone_type": {
      "value": [
        {"id": "stone_id_diamond", "text": "Diamond", "count": 180},
        {"id": "stone_id_ruby", "text": "Ruby", "count": 55},
        {"id": "stone_id_pearl", "text": "Pearl", "count": 2}
      ]
    },
    "primary_metal_type": {
      "value": [
        {"id": "stone_id_gold", "text": "Gold", "count": 80},
        {"id": "stone_id_silver", "text": "Silver", "count": 45},
        {"id": "stone_id_white_gold", "text": "White Gold", "count": 20}
      ]
    }
  },
  "facet_names": ["colors", "product_type", "primary_stone_type", "primary_metal_type"]
}
```

In the response, the facet is returned in descending order of the *count* value because that is the default `facet.sort` order. If a different sort order is required, refer to Facet Order.

3.3.2 Example: Get 100 facets for a facet type

Request:

```
{
  "facet": {
    "colors": {"max": 100}
  }
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_f1",
  "facet": {
    "colors": {
      "value": [
        {"id": "color_id_red", "text": "Red", "count": 23},
        {"id": "color_id_blue", "text": "Blue", "count": 11},
        {"id": "color_id_yellow", "text": "Yellow", "count": 10},
        {"id": "color_id_green", "text": "Green", "count": 5},
        {"id": "color_id_orange", "text": "Orange", "count": 18},
        {"id": "color_id_purple", "text": "Purple", "count": 2},
        {"id": "color_id_white", "text": "White", "count": 3},
        {"id": "color_id_grey", "text": "Grey", "count": 9},
        {"id": "color_id_black", "text": "Black", "count": 11},
        {"id": "color_id_multi", "text": "Multi-color", "count": 5},
        {"id": "color_id_no_color", "text": "No color", "count": 17},
        ...
      ]
    }
  },
  "facet_names": ["colors"]
}
```

3.3.3 Example: Get price range facet with 3 buckets

Request:

```
{
  "facet": {
    "price": {"max": 3}
  }
}
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_f2",
          "facet": {
            "price": {
              "min": 10,
              "max": 1500,
              "value": [
                {"id": "pR1151f20", "min": 10, "max": 200,
                   "text": "10 - 200", "count": 124},
                {"id": "pR115Qdq", "min": 200, "max": 800,
                   "text": "200 - 800", "count": 89},
                {"id": "pR115gGp", "min": 800, "max": 1500,
                   "text": "800 - 1500", "count": 94}
              ]
            }
          },
          "facet_names": ["price"]
        }
```

3.3.4 Example: Get category nested facet with 5 leaf category

Request:

```
        {
          "facet": {
            "category_tree": {"max": 5}
          }
        }
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_f3",
  "facet": {
    "category_tree": {
      "value": [
        {
          "id": "ct1",
          "text": "Mens",
          "count": 5,
          "sub": [
            {"id": "ct11", "text": "Shirt", "count": 2},
            {"id": "ct12", "text": "Pants", "count": 3}
          ]
        },
        {
          "id": "ct2",
          "text": "Womens",
          "count": 2,
          "sub": [
            {"id": "ct21", "text": "Shirt", "count": 1},
            {"id": "ct22", "text": "Pants", "count": 1}
          ]
        },
        {
          "id": "ct3",
          "text": "Kids",
          "count": 5,
          "sub": [
            {"id": "ct31", "text": "Shirt", "count": 5}
          ]
        }
      ]
    }
  },
  "facet_names": ["category_tree"]
}
```

Note: The `max` is applied to limit the facet_values under `sub`.

The `sub` key indicates the list of sub facets that belongs to the facet. This is useful to represent hierarchical structure such as category tree. In this example, there are 3 **Shirt** categories, which belongs to different parent categories (Mens, Womens, Kids)

3.3.5 Examples: Get the top 3 values for all facets from a dynamic attributes collection

Request:

```
{
  "facet": {
    "dyn_attrs": {"max": 3}
  }
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_f1",
  "facet": {
    "dyn_attrs.color": {
      "number_of_products": 8762,
            "display_name": "Color",
            "value": [
              {"count": 913, "text": "Black", "id": "facet_idQmxhY2s=", "in_content":
"product"},
              {"count": 532, "text": "Yellow", "id": "facet_idWWVsbG93", "in_content":
"product"},
              {"count": 378, "text": "Green", "id": "facet_idR3JlZW4=", "in_content":
"product"}
            ]
          },
          "dyn_attrs.clothing_type": {
            "number_of_products": 3139,
            "display_name": "Clothing Type",
            "value": [
              {"count": 229, "text": "Pants", "id": "facet_idUGFudHM=", "in_content":
"product"},
              {"count": 418, "text": "Coveralls", "id": "facet_idQ292ZXJhbGxz", "in_content":
"product"},
              {"count": 233, "text": "Jackets", "id": "facet_idSmFja2V0cw==", "in_content":
"product"}
            ]
          }
        },
        "facet_names": [
          "dyn_attrs.material",
          "dyn_attrs.color",
          "dyn_attrs.clothing_type"
        ]
      }
```

Note:
The `dyn_attrs` in the request refers to the name of a dynamic attribute enabled for facets.

3.3.6 Example: Get all values for a specific attribute from a dynamic attributes collection

Request:

```
      {
        "facet": {
          "dyn_attrs.clothing_type": {"max": 100}
        }
      }
```

Response:

```
      {
        "ts": 1480977544,
        "rid": "response_id_f1",
        "facet": {
          "dyn_attrs.clothing_type": {
            "number_of_products": 1441,
            "display_name": "Clothing Type",
            "value": [
              {"count": 234, "text": "Pants", "id": "facet_idUGFudHM=", "in_content":
"product"},
              {"count": 441, "text": "Coveralls", "id": "facet_idQ292ZXJhbGxz", "in_content":
"product"},
              {"count": 240, "text": "Jackets", "id": "facet_idSmFja2V0cw==", "in_content":
"product"},
              {"count": 307, "text": "Shirts", "id": "facet_idU2hpcnRz", "in_content":
"product"},
              {"count": 219, "text": "Safety Vests", "id": "facet_idU2FmZXR5IFZlc3Rz",
"in_content": "product"},
              ...
            ]
          }
        },
        "facet_names": [
          "dyn_attrs.color"
        ]
      }
```

Note:
The `dyn_attrs.clothing_type` in the request refers to the name of an individual attribute (`clothing_type`) within a dynamic attribute (`dyn_attrs`) enabled for facets.

3.4 Filter for Search

Filters are specific selections (from Facet or otherwise) that allows you to reduce the number of results returned. Filters must be configured in the Customer Engagement Console (CEC).

Filter ensures that the attribute of the item exactly matches the specified value. Unlike a query, a filter does not consider other attributes.

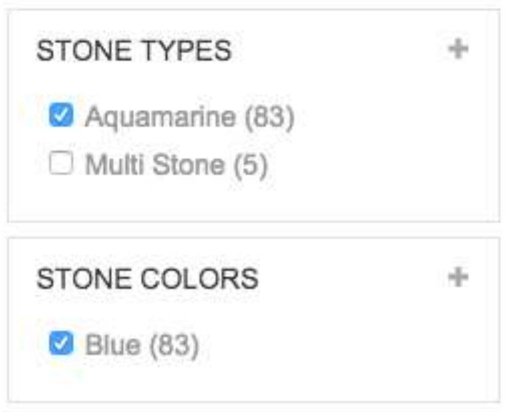
Consider the following two products. Querying with keyphrase “red” gives you both products. This is because the query looks at all the fields of the product. Filter on **color** “red” only gives you product 2. This is because the filter only looks at the **color** field of products.

```
Product 1: {"name": "red flag shirt", "color": "white"}
Product 2: {"name": "shirt", "color": "red"}
```

By default, filters on different facet types are applied with an AND operation while within a facet type are applied with an OR operation.

Filter on different facet types

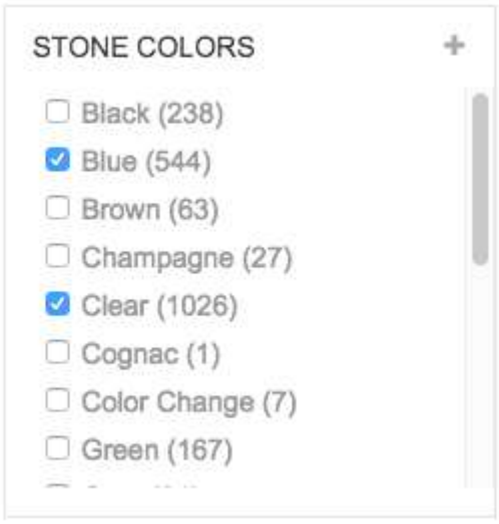
Since the filters are applied on different facet types (stone types and stone colors), it will be applied with an AND operation. This filters on (**Aquamarine** stone types AND **Blue** stone colors).



```
Filter 1:
{"stone_types": "Aquamarine"}

Filter 2:
{"stone_colors": "Blue"}
```

Filter on same facet type

Since the filters are applied on the same facet type (stone colors), it will be applied with an OR operation. This filters on (**Blue** OR **Clear**) stone colors.


STONE COLORS

- [ ] Black (238)
- [x] Blue (544)
- [ ] Brown (63)
- [ ] Champagne (27)
- [x] Clear (1026)
- [ ] Cognac (1)
- [ ] Color Change (7)
- [ ] Green (167)

Filter 1 & 2:

{"stone_colors": ["**Blue**", "**Clear**"]}

The following examples follow:

- Apply a term filter
- Apply a range filter
- Apply a nested filter
- Apply a filter on a dynamic attribute for red products

3.4.1 Examples: Apply diamond filter on primary_stone_type for wedding keyphrase in product

Request:

```
{
  "query": {
    "keyphrase": ["wedding"]
  },
  "filter": {
    "primary_stone_type": {
      "value": ["stone_id_diamond"]
    }
  },
  "content": {}
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_ff1",
  "url": "/searchpage?keyphrase=wedding&primary_stone_type=stone_id_diamond",
  "query2id": {
    "keyphrase": "keyphrase_id_wedding"
  },
  "filter": {
    "primary_stone_type": {
      "value": ["stone_id_diamond"]
    }
  },
  "content": {
    "product": {
      "value": [
        {"name": "wedding ring 1", "price": 2500, "image_url": "",
         "rating": 5, "primary_stone_type": "diamond"},
        {"name": "wedding ring 2", "price": 1000, "rating": 1,
         "primary_stone_type": "diamond"},
        {"name": "wedding jewelry 3", "price": 5000,
         "primary_stone_type": "diamond"},
        …
      ]
    }
  },
  "n_item": 10,
  "total_item": 150,
  "page_number": 1,
  "total_page": 15
}
```

3.4.2 Examples: Apply range between 1000 and 2500 on price for wedding keyphrase in product

Request:

```
      {
        "query": {
          "keyphrase": ["wedding"]
        },
        "filter": {
          "price": {
            "value": [{"min": 1000, "max": 2500}]
          }
        },
        "content": {}
      }
```

Response:

```
      {
        "ts": 1480977544,
        "rid": "response_id_ff2",
        "url": "/searchpage?keyphrase=wedding&price-min=1000&price-max=2500",
        "query2id": {
          "keyphrase": "keyphrase_id_wedding"
        },
        "filter": {
          "price": {
            "value": [{"min": 1000, "max": 2500}]
          }
        },
        "content": {
          "product": {
            "value": [
              {"name": "wedding ring 1", "price": 2500, "image_url": ""},
              {"name": "wedding ring 2", "price": 1000, "rating": 1},
              {"name": "wedding jewelry 4", "price": 1999},
              …
            ]
          }
        },
        "n_item": 10,
        "total_item": 150,
        "page_number": 1,
        "total_page": 15
      }
```

3.4.3 Example: Filter on category Mens > Shirt for red product

This example is a continuation of Getting category nested facet with 5 leaf category

Request:

```
{
  "query": {
    "keyphrase": ["red"]
  },
  "filter": {
    "category_tree": {
      "value": ["ct11"]
    }
  },
  "content": {}
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_37",
  "url": "/searchpage?keyphrase=red&category=ct11",
  "query2id": {
    "keyphrase": "keyphrase_id_red"
  },
  "filter": {
    "category_tree": {
      "value": ["ct11"]
    }
  },
  "content": {
    "product": {
      "value": [
        {"name": "Shirt 1", "price": 25, "color": "red",
          "image_url": "", "category": "shirt",
          "parent_category": "men"},
        {"name": "New Arrival Shirt 2", "color": "red",
          "price": 100, "rating": 1,
          "category": "shirt", "parent_category": "men"},
        {"name": "Red Shirt 4", "price": 19, "category": "shirt",
          "parent_category": "men"},
        …
      ]
    }

  },
  "n_item": 10,
  "total_item": 297,
  "page_number": 1,
  "total_page": 30
}
```

3.4.4 Examples: Filter pants on a dynamic attribute for red products

Request:

```
        {
          "query": {
            "keyphrase": ["red"]
          },
          "filter": {
            "dyn_attrs.clothing_type": {
              "value": ["Pants"]
            }
          },
          "content": {}
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_37",
          "query2id": {
            "keyphrase": "keyphrase_id_red"
          },
          "content": {
            "product": {
              "value": [
                {"name": "Pants 1", "price": 25, "color": "red", "image_url": "", "dyn_attrs":
{"clothing_type": "Pants", ...}},
                {"name": "New Arrival Pants 2", "color": "red", "price": 100, "rating": 1,
"dyn_attrs": {"clothing_type": "Pants", ...}},
                {"name": "Red Pants 4", "price": 19, "dyn_attrs": {"clothing_type": "Pants",
...}}
                …
              ]
            }

          },
          "n_item": 10,
          "total_item": 297,
          "page_number": 1,
          "total_page": 30
        }
```

Note:
The `dyn_attrs` in the request refers to the name of a dynamic attribute enabled for filtering.

3.4.5 Filter for Recommendation

Recommendation results can be filtered if the domain was migrated to use standard specs. There are two types of filters that can be used, namely:

- exact term filtering,
- numeric range filtering.

The filters sent in the request can be placed at the request level or the widget level. We always recommend the former option, which means that the filter will be applied to all widgets in a batch request. Filters can be applied to product or sku level attributes, including locale or store-specific attributes.

3.4.6 Example: Filter on homepage widget for Medium or Large size products

```
{
  "widget": {
    "rfkid": "rfkid_1"
  },
  "filter": {
    "size": ["M", "L"]
  },
  "content": {"product": {}},
  "n_item": 16
}
```

3.4.7 Examples: Filter on PDP widget for Red or Blue products for french variant of the site

```
{
  "widget": {
    "rfkid": "rfkid_31"
  },
  "filter": {
    "color": ["Rouge", "Bleue"]
  },
  "context": {
    "page": {
      "locale_country": "ca",
      "locale_language": "fr"
    }
  }
  "content": {"product": {}},
  "n_item": 16
}
```

3.4.8 Example: Filter on PDP widget for products between 10 and 50 final price

```
{
  "widget": {
    "rfkid": "rfkid_32"
  },
  "filter": {
    "final_price": {"min": 10, "max": 50}
  },
  "content": {"product": {}},
  "n_item": 16
}
```

3.4.9 Examples: Filter on cart widget for products matching a specific fitment (automotive industry)

```
{
  "widget": {
    "rfkid": "rfkid_4"
  },
  "context": {
    "fitment": {
       "ids": ["fit-id1", "fit-id2",...]
    }
  },
  "content": {"product": {}},
  "n_item": 16
}
```

3.5 Suggestion

Suggestions are tips used to guide the end user on their next query. Refer to Appendix for details on different suggestion types.

Examples:

- Get keyphrase suggestion
- Get trending category suggestion
- Get auto complete suggestion from specific content

3.5.1 Examples: Getting 5 keyphrase suggestions for keyphrase shirt

Request:

```
{
  "query": {
    "keyphrase": ["shirt"]
  },
  "suggestion": {
    "keyphrase": {
      "max": 5
    }
  }
}
```

Response:

```
{
  "ts": 1480997544,
  "rid": "response_id_S1",
  "query2id": {
    "keyphrase": "keyphrase_id_shirt"
  },
  "suggestion": {
    "keyphrase": {
      "value": [
        {"id": "keyphrase_id_shirt", "text": "Shirt",
         "in_content": "product"},
        {"id": "keyphrase_id_short", "text": "Short",
         "in_content": "product"},
        {"id": "keyphrase_id_mens_shirt", "text": "Men's Shirt",
         "in_content": "product"},
        {"id": "keyphrase_id_shirt_style_guide",
         "text": "Shirt Style Guide", "in_content": "article"},
        {"id": "keyphrase_id_shirt_on_sale", "text": "shirt on sale"}
      ]
    }
  }
}
```

Keyphrase suggestions return a mix of, but not all, suggestion types. The response also won't identify the type of suggestion.

In the response:

- "Shirt" is autocomplete
- "Short" is type_correction
- "Men's Shirt" and "Shirt Style Guide" are related_search/ query_expansioin (from any content type)

- "shirt on sale" is from recent search history

3.5.2 Example: Getting 3 trending category suggestion

Request:

```
{
  "suggestion": {
    "trending_category": {
      "max": 3
    }
  }
}
```

Response:

```
{
  "ts": 1480997544,
  "rid": "response_id_S2",
  "suggestion": {
    "trending_category": {
      "value": [
        {"id": "category_id_men", "text": "Men", "url": "/cat/men"},
        {"id": "category_id_shirt", "text": "Shirt", "url": "/cat/shirt"},
        {"id": "category_id_backpack", "text": "Backpack", "url": "/cat/backpack"}
      ]
    }
  }
}
```

3.5.3 Examples: Getting 3 autocomplete suggestion for mov in article

Request:

```
{
  "query": {
    "keyphrase": ["mov"]
  },
  "suggestion": {
    "autocomplete": {
      "max": 3,
      "in_content": {
        "value": [{"article": {}}]
      }
    }
  }
}
```

Response:

```
{
  "ts": 1480997544,
  "rid": "response_id_S3",
  "query2id": {
    "keyphrase": "keyphrase_id_mov"
  },
  "autocomplete": "movie",
  "suggestion": {
    "autocomplete": {
      "value": [
        {"id": "keyphrase_id_movie", "text": "Movie", "in_content": "article"},
        {"id": "keyphrase_id_move", "text": "Move", "in_content": "article"},
        {"id": "keyphrase_id_movember", "text": "Movember", "in_content": "article"}
      ]
    }
  }
```

```
        }
```

3.6 Appearance

This section explains the basics of querying for appearance templates and variables to render widgets.

Note:
Appearance is currently only supported for html_block and banner widget types. It is not supported for merchandising widgets like search results, recommendations, and preview search.

A widget's appearance consists of several components. These components are referred to in the APIs as *sections*. These sections are:

- HTML
- CSS
- Javascript (not yet supported)

Each section consists of templates and variables.

A section may consist of several templates, and each template contains content in the format specific to that section. For instance a template for HTML section would consist of content in HTML format.

The content of each of the templates may make use of *variables* that can be substituted at run time to customize the templates. Some examples of common variables are Title, Subtitle, Padding, Font Size, Color etc. These variables are what a merchandiser enters when creating a widget with a specific appearance (or also called style).

Further, templates for any of the sections may be defined specific for a target device like pc, mobile or tablet.

Sitecore Discover API provides an extensive appearance interface to retrieve default, specific or all templates and variables for all sections and devices.

Following are some of the examples.

3.6.1 Example: Get appearance templates for all sections for the default device

Request:

```
{
  "context": {
    "page": {
       "uri": "/"
    }
  },
  "widget": {
    "rfkid": "hs_home_hb"
  },
  "appearance": {
     "templates": {}
  }
}
```

Response:

```
{
  "widget": {
    "rfkid": "hs_home_hb",
    "type": "banner",
    "variation_id": "530651"
  },
          "url": "...",
          "appearance": {
            "templates": {
              "html": {
                "devices": {
                  "pc": {
                    "content": "\n<div class=\"banner\">\n  <div class=\"column\">\n    <a
class=\"image_box\" href=\"/rfk/\"> \n      <div class=\"background\">\n        <div
class=\"content\">\n          <div class=\"title\">NEW ARRIVALS</div> \n          <div
class=\"subtitle\">New fall styles are hitting the stores 11/21</div> \n          <div
class=\"square\">SHOP NOW</div>             \n        </div>\n      </div>\n    </a>\n
</div>\n</div>\n"
                  }
                }
              },
              "css": {
                "devices": {
                  "pc": {
                    "content": "[data-rfkid].rfk2_banner.rfk2_hs_home_hb .banner{-webkit-box-
sizing:border-box;border:none;box-sizing:border-box;margin:0;padding:0;width:100%}
...
[data-rfkid].rfk2_banner.rfk2_hs_home_hb .square{border:1px solid #FFFFFF;display:inline-
block;font-family:'DIN';font-size:16px;padding:26px 32px;text-align:center;text-
rendering:geometricPrecision}"
                  }
                }
              }
            },
            "css_names": [
              "rfk2_banner",
              "rfk2_hs_home_hb"
            ],
            "html_names": [
              "banner",
              "hs_home_hb"
            ]
          },
          "ts": 1618370786846,
          "rid": "response_id:125757321:a7d76cf8a2a6e656b60e4322360db3b5ce4cf9d4",
          "dt": 104
        }
```

3.6.2 Example: Get appearance templates for all sections and all devices, and do not substitute variables in templates and do not add css selectors to CSS

Request:

```
        {
          "context": {
            "page": {
              "uri": "/"
            }
          },
          "widget": {
            "rfkid": "hs_home_hb"
          },
          "appearance": {
            "templates": {
              "devices": [],
              "keep_variables": true,
              "keep_original_css": true
            }
          }
        }
```

Response: {

```
          "widget": {
            "rfkid": "hs_home_hb",
            "type": "banner",
            "variation_id": "530651"
          },
          "url": "...",
          "appearance": {
            "templates": {
              "html": {
                "devices": {
                  "mobile": {
                    "content": "\n<div class=\"banner\">\n  <div class=\"column\">\n    <a
class=\"image_box\" href=\"{{$image_href}}\"> \n      <div class=\"background\">\n        <div
class=\"content\">\n          <div class=\"title\">{{$image_overlay_title,}}</div> \n
<div class=\"subtitle\">{{$image_overlay_subtitle,}}</div> \n          <div
class=\"square\">{{$image_overlay_button,}}</div>             \n        </div>\n      </div>\n
</a>\n  </div>\n</div>\n"
                  },
                  "pc": {
                    "content": "\n<div class=\"banner\">\n  <div class=\"column\">\n    <a
class=\"image_box\" href=\"{{$image_href}}\"> \n      <div class=\"background\">\n        <div
class=\"content\">\n          <div class=\"title\">{{$image_overlay_title,}}</div> \n
<div class=\"subtitle\">{{$image_overlay_subtitle,}}</div> \n          <div
class=\"square\">{{$image_overlay_button,}}</div>             \n        </div>\n      </div>\n
</a>\n  </div>\n</div>\n"
                  },
                  "tablet": {
                    "content": "\n<div class=\"banner\">\n  <div class=\"column\">\n    <a
class=\"image_box\" href=\"{{$image_href}}\"> \n      <div class=\"background\">\n        <div
class=\"content\">\n          <div class=\"title\">{{$image_overlay_title,}}</div> \n
<div class=\"subtitle\">{{$image_overlay_subtitle,}}</div> \n          <div
class=\"square\">{{$image_overlay_button,}}</div>             \n        </div>\n      </div>\n
</a>\n  </div>\n</div>\n"
                  }
                }
              },
              "css": {
                "devices": {
                  "mobile": {
                    "content": "\n.banner {\n  position: relative;\n  -webkit-box-flex: 1;\n
-ms-flex: 1;\n          flex: 1;\n  padding: {{$padding}};\n  margin: {{$margin}};\n  border:
{{$border}};            \n}\n.banner:before{\n  content: \"\";\n  display: block;
        ...
        font-family: {{$font_family_bold}};\n  text-align: center;\n  text-rendering:
geometricPrecision;\n"
                  },
                  "pc": {
                    "content": "\n.banner {\n  width: 100%;\n  padding: {{$padding}};\n
margin: {{$margin}};\n  border: {{$border}};\n  -webkit-box-sizing: border-box;\n  box-sizing:
border-box;  \n}\n\n.column {\n  -webkit-box-orient: vertical;\n  -webkit-box-direction:
normal;\n      -ms-flex-direction: column;\n          flex-direction: column;
        ...
        font-family: {{$font_family_bold}};\n  text-align: center;\n  text-rendering:
geometricPrecision;\n"
                  },
                  "tablet": {
                    "content": "\n.banner {\n  width: 100%;\n  padding: {{$padding}};\n
margin: {{$margin}};
        ...
        font-family: {{$font_family_bold}};\n  text-align: center;\n  text-rendering:
geometricPrecision;\n"
                  }
                }
              }
            },
            "css_names": [
              "rfk2_banner",
              "rfk2_hs_home_hb"
            ],
            "html_names": [
              "banner",
              "hs_home_hb"
            ]
          },
          "ts": 1618370927522,
          "rid": "response_id:125757321:d56fc7ae46816faace76e2e45eaeb7ef899e5925",
          "dt": 38
        }
```

Chapter 4 Detailed Structure of Request.data

This section describes the detailed data structure of Request.data.

4.1 query

4.1.1 Data structure

The following snippet shows the basic structure of a query. See Data structure snippets for interpretation of blue symbols.

```
        ["query": {
          ["<query_type>": {
            "value": [
              [<query_string> ]+
            ]
          } ]*
        }]?

        <query_string>: String(s) OR List of String(s)
```

The `query` object holds the query_strings for each <query_type>. The value of each <query_type> should be a value object. The value of the value object is a list containing either *string* or *list of string(s)*.

For basic use case, the list should contain only one string. If only one string is provided, Sitecore Discover will determine the appropriate boolean operation to applied to the query strings.

If the query does not have any results, Sitecore Discover will, by default, execute another query using the top suggestion for the original query.

For advanced use case, refer to Boolean Operation in Query.

4.1.2 Default value

Key	Value
`query`	`{"keyphrase": [""]}`

4.1.3 Examples

Full representation

```
        "query": {
          "keyphrase": {
            "value": ["red"]
          },
          "category_names": {
            "value": ["men's clothing", "pants"]
          }
        }
```

Short representation:

```
        "query": {
          "keyphrase": "red",
          "category": ["men's clothing", "pants"]
        }
```

Refer to Value Object for details on short representation.

4.2 exact_match

4.2.1 Data structure

The following snippet shows the basic structure of `exact_match`. See Data structure snippets for interpretation of blue symbols.

```
        ["exact_match": bool ]?
```

Exact_match is a flag that allows you to control the search behavior. If it is turned on, Sitecore Discover will only search with the exact keyphrase.

4.2.2 Default value

Key	Value
exact_match	false

4.3 facet

4.3.1 Data structure

The following snippet shows the basic structure of facet. See Data structure snippets for interpretation of blue symbols.

```
[ "facet": {
  [ "all": bool, ]?
  [ "max": int, ]?
  [ "total": bool, ]?
  [ "empty": bool, ]?
  [ "sort": {
    "type": "<sort_type>",
    "order": "<order_type>"
  }, ]?
  [ "<facet_type>": {
    [ "min_count": int, ]?
    [ "max": int, ]?
    [ "total": bool, ]?
    [ "sort": {
      "type": "<sort_type>",
      "order": "<order_type>"
    }, ]?
    [ "in_content": "<content_type>", ]?
    [ "field": [[ "<field_type>" ]+], ]?
    [ "value": [
      [<facet_type_value_or_id> ]+
    ] ]?
  } ]*
}]?
```

`facet` contains the facet_type to be returned, each facet_type is returned in a format corresponding to the nature of the facet_type. There are three different formats (term, range, rested). Refer to Get Started - Facet for the explanations.

Key	Description
`facet.all` boolean	Optional. Get all facet_type
`facet.max` *integer*	Optional. The maximum number of facet_value to return per facet_type. Use -1 to get the maximum allowed set of facet values. Requesting too many facet values will have a latency impact. We recommend you allow the system to return optimal number of facet values,
`facet.total` *boolean*	Optional. Return the total number of values for each facet_type. By default, it is `false`. It is typically useful when you have a lot of values, and you want to get only a few in the initial search

	query using `facet.max` and want to know how many more values are present.
`facet.empty` *boolean*	Optional. Return facets requested even if a given facet has no applicable filters in the response. By default, it is `false`.

`facet.sort`
object
{"type": "text"/ "count",
"order": "asc"/ "desc"}
	Optional. Determine the sort order of the facet value

`facet.<facet_type>.min_count` *integer*	Optional. The minimum count of any returned facet_value.
`facet.<facet_type>.max` *integer*	Optional. Override of `facet.max` for this <facet_type>
`facet.<facet_type>.total` *boolean*	Optional. Override of `facet.total` for this <facet_type>
`facet.<facet_type>.in_content` *string*	Optional. The content_type that this <facet_type> belongs to.

`facet.<facet_type>.sort`
object
{"type": "text"/ "count",
"order": "asc"/ "desc"}
	Optional. Override of `facet.sort` for this <facet_type>

`facet.<facet_type>.field` *list of string*	Optional. Additional field to be returned in facet_value.
`facet.<facet_type>.value` *list of string*	Optional. Fixed facet_value to be returned.
`facet.<facet_type>.keyphrase` *string*	Optional. [Coming Soon]. return facet values that match the given keyphrase.
`facet.<facet_type>.after_value` *string*	Optional. [Coming Soon]. returns facet values after the value that is specified. This is a mechanism to paginate when there are a large set of facet values and not all values can be returned. This would return a list of values after the value specified in `after_value`.

	Note: `sort` option of ascending or descending must be specified when paginating `after_value`.

4.3.2 Default value

Key	Value
`facet.all`	`false`
`facet.max`	`10`
`facet.empty`	`false`
`facet.sort`	`{"type": "text", "order": "asc"}`
`facet.<facet_type>.min_count`	`1`
`facet.<facet_type>.max`	`facet.max`
`facet.<facet_type>.in_content`	`"product"`
`facet.<facet_type>.sort`	`facet.sort`
`facet.<facet_type>.field`	`N/A`
`facet.<facet_type>.value`	`N/A`

Note:
`facet.sort` does not overwrite `request.sort`. They are independent.

4.3.3 Examples

In the following example, we request to:

- Return 2 facet_value per facet_type
- Sort the facet_value by Alphabetical order
- For color, return color_id_10 and color_id_2 if the count is not 0
- For category, give me 15 facet_values (instead of 2). Return the facet_value that has count greater or equal to 5.

```
"facet": {
  "max": 2,
  "sort": {
    {
      "value": [
        {"type": "text", "order": "asc"}
      ]
    }
  },
  "colors": {
    "value": ["color_id_10", "color_id_2"]
  },
  "category_names": {
    "min_count": 5,
    "max": 15
  }
}
```

For basic use cases, refer to Get Started - Facet.

For advanced use cases, refer to one of the following:

- Facet Order
- Facet Type Control
- Facet Option Control

4.4 filter

4.4.1 Data structure

The following snippet shows the basic structure of filter. See Data structure snippets for interpretation of blue symbols.

```
["filter": {
  ["<facet_type>": {
    "value": [["<filter_id>" ]+]
  } ]+
}]?

<filter_id>: <facet_id> OR <range_filter>

<range_filter>: object
{
  "min": float,
  "max": float
}
```

The `filter` object holds the filters to be applied to all <content_type> and <facet_type>. The value of `filter` is a value object. The value of value object is a list that contains either *facet_id* or *range_filter*. <facet_id> is an id returned by a facet query. <range_filter> is an object that contains a "`min`" and "`max`" key that represents a range.

4.4.2 Example

The following example applies:

- 1 filter on category
- 2 filters on color
- 1 range_filter on price

It means "Filter on items that belongs to the category with *idCT15xzaq98bc*, have both color id of "*idCOace567hdjk*" and "*idCOwxg64fhj*" and with price between *0* to *100*."

```
"filter": {
  "category_names": {
    "value": ["idCT15xzaq98bc"]
  },
  "colors": {
    "value": ["idCOace567hdjk", "idCOwxg64fhj"]
  },
  "price": {
    "value": [{"min": 0, "max": 100}]
  }
}
```

4.5 suggestion

4.5.1 Data structure

The following snippet shows the basic structure of suggestion. See Data structure snippets for interpretation of blue symbols.

```
        ["suggestion": {
          ["<suggestion_type>": {
            ["max": int, ]?
            ["in_content": {
              "value": [[<in_content_value> ]*]
            } ]?
          } ]+
        }]?

        <in_content_value>: object
        {
          "type": "<content_type>",
          ["max": int ]?
        }
```

4.5.2 Default value

Key	Value
suggestion.<suggestion_type>.max	5
suggestion.<suggestion_type>.in_content	N/A

There is no default value for `in_content`, there is no limitation on <content_type> by default.

4.5.3 Examples

The following example gets the user's last 10 search query on article content type.

```
        "suggestion": {
          "recent": {
            "max": 10,
            "in_content": {
              "value": [
                {"type": "article"}
              ]
            }
          }
        }
```

For basic use cases, Refer to Get Started - Suggestion.

4.6 widget_id (deprecated)

4.6.1 Data structure

The following snippet shows the basic structure of widget_id.

```
        ["widget_id": {
          "id": "<value>"
        }]?
```

4.6.2 Example

The following example selects widget_id 1.

```
        "widget_id": {
          "id": "1"
        }
```

4.7 widget

4.7.1 Data structure

The following snippet shows the basic structure of widget. See Data structure snippets for interpretation of blue symbols.

```
        ["widget": {
          ["rfkid": "<value>",]?
          ["widget_id": "<value>"]?
          ["all": "<value>"]?
        }]?
```

Key	Value
`rfkid`	A Sitecore Discover id associated with the widget. Only one of the ids (rfkid or widget_id) should be specified. We recommend you use rfkid when specifying a widget.
`widget_id`	A unique id assigned to a widget by Sitecore Discover. If both rfkid and widget_id are specified, then widget_id takes precedence.
`all`	A boolean. If true, all widgets in the page are returned as a batch response. rfkid and widget_id must not be set. If so, it would throw an invalid request exception.

4.7.2 Examples

The following example selects `rfkid` rfkid_3.

```
        "widget": {
          "rfkid": "rfkid_3"
        }
```

The following example selects `widget_id` 3000.

```
        "widget": {
          "widget_id": "3000",
        }
```

The following example selects all widgets in a page defined in the Customer Engagement Console, and will return the response in batch.

```
        "widget": {
          "all": true
        }
```

4.8 content

4.8.1 Data structure

The following snippet shows the basic structure of content. See Data structure snippets for interpretation of blue symbols.

```
        ["content": {
          ["<content_type>": {
            ["max": int, ]?
            ["sort": {<sort_object>}, ]?
            ["get_fields": bool, ]?
            ["field": {
              ["value": [
                ["<content_field>" ]+
              ] ]?
            } ]?
          } ]*
        }]?
```

The content key is required for any content to be returned.

<content_type> can be inquired one at a time or multiple at a time.

max limits the number of results of the content returned per page.

sort determines the sort order of the content_type.

get_fields is a boolean flag. Setting it to true will return a list of all possible content field names. See the content response.

field is a value object. The value object contains *strings* that represent the fields to be returned per result.

4.8.2 Default value

Key	Value
content	{"product": {}}
content.<content_type>.max	n_item
content.<content_type>.sort	sort
content.<content_type>.get_fields	false

The default of content.<content_type>.max is the n_item in request.

The default of content.<content_type>.sort is the sort in request.

4.8.3 Example

The following example request gets

- The name and price of 10 products
- The title of 5 articles

```
        "content": {
          "product": {
            "field": {
              "value": ["name", "price"]
            }
          },
          "article": {
            "max": 5,
            "field": {
              "value": ["title"]
            }
          }
        }
```

For basic use cases, refer to Get Started - Querying Content.

For advanced use cases, refer to one of the following:

- Limiting Returned Content Attribute
- Querying Mixed Content

4.9 appearance

4.9.1 Data structure

The following snippet shows the basic structure of `appearance` . See Data structure snippets for interpretation of blue symbols.

```
        ["appearance":
          ["<templates>": {
            ["section": [string], ]?
            ["devices": [string], ]?
            ["keep_original_css": bool, ]?
            ["keep_variables": bool, ]?
           }]?
          ["<variables>": {
            ["section": [string], ]?
           }]?
          ["<includes>": {
            ["section": [string], ]?
            ["devices": [string], ]?
           }]?
         }]?
```

The `appearance` key is required for any appearance information to be returned.

Appearance request parameters are:

Field	Description
`appearance` object	Request for appearance information By default, it would return the following in the response: html_names css_names
`templates` object (optional)	Request for templates for the given widget. By default, it would return all sections for a specific device as determined by the request (explicitly specified in request in the context, or determined by User-Agent header.
`templates.sections` array of string (optional)	Request for all or specific template sections for the requested widget(s). Possible values: empty array: returns all sections `html`: returns html section `css`: returns css section `js`: returns javascript section
`templates.devices` array of string (optional)	Request for all or specific template sections that apply for that device(s).

	Possible values: empty array: returns sections for all devices `pc`: returns sections for desktop/pc device `mobile`: returns sections for mobile device `tablet`: returns sections for tablet device
`templates.keep_variables` boolean (optional)	If true, variables embedded in the templates are not substituted for its runtime values. Default: `false` By default, all variables are substituted as far as possible. If variables refer to results of the widgets, only those variables are returned as is, so the service may substitute these values based on teh widget's results.
`templates.keep_original_css` boolean (optional)	When set to true, it keeps all CSS section data as specified during style sefinition, for example, it will not add widget div specific selectors to the CSS. By default, appropriate CSS class names are added to ensure CSS applies the appearance style to the specific widget without impacting other parts of the page. Default: `false`
`variables` object (optional)	Request for variables defined as part of the appearance style for the given widget. By default, it would return all variables. Variables are generally not required during the runtime as all variable values are automatically substituted as far as possible when returning the templates.
`variables.sections` array of string (optional)	Request for all or variables that are marked for use in the appropriate sections for the requested widget. Possible values: empty array or `all`: returns variables applicable for any section `html`: returns variables applicable for html section `css`: returns variables applicable for css section `js`: returns variables applicable for js section

`includes` object (optional)	Request for includes (file paths) By default, it would return all sections for specific device as determined by the request (explicitly specified in request in the context, or determined by User-Agent header)
`includes.sections` array of string (optional)	Request for all or specific includes sections for the requested widget. Possible values: empty array or `all`: returns all sections `html`: returns html section `css`: returns css section `js`: returns javascript section
includes.devices array of string (optional)	Request for all or specific includes sections that apply for that device(s). Possible values: empty array or `all`: returns includes sections for all devices `pc`: returns includes sections for desktop/pc device `mobile`: returns includes sections for mobile device `tablet`: returns includes sections for tablet device

4.9.2 Examples

Asking for only html template for the default device.

```
        "appearance":{
          "templates": {
            "section": ["html"],
          }
        }
```

Asking for all templates and variables information for all devices.

```
        "appearance":{
          "templates": {
            "devices": []
          },
          "variables": {
            "devices": []
          }
        }
```

4.10 batch

4.10.1 Data structure

The following snippet shows the basic structure of batch. See Data structure snippets for interpretation of blue symbols.

```
        ["batch": [
          [{<request_key>: <request_value>}]+
        ]]?
```

`batch` is only expected when multiple requests are clubbed together. For example, in case of recommendations, response might be required for similar and bundle items widget in a batch request as opposed to two individual requests.

When you have multiple recommendation widgets on the same page, a batch request returns non-duplicate results between the two widgets.

It's an array of objects where the keys of the object represent the key that needs to be overridden for a specific request in the batch and value represents the overridden value.

4.10.2 Example

The following example overrides filter object and corresponding widget ids for a batch of 2 requests. It means that for the first request, filter on category 1 and color red. For the second request, filter on category 2 and color blue.

```
        "batch": [
           {
             "filter": {"category_id": "1", "colors": "red" },
             "widget": {"rfkid": "rfkid_1"}
           },
           {
             "filter": {"category_id": "2", "colors": "blue" },
             "widget": {"rfkid": "rfkid_4"}
           }
        ]
```

4.11sort

4.11.1 Data structure

The following snippet shows the basic structure of sort. See Data structure snippets for interpretation of blue symbols.

```
        ["sort": {
          ["value": [
            [{"type": "<sort_type>", "order": "<order_type>"} ]+
          ] ]?
        }]?
```

`sort` determines the order of the content results. `sort` is a value object. The value object contains a list of *object*. Each object contains a supported <sort_type>, and the <order_type> either "asc" or "desc".

4.11.2 Default value

Key	Value
`sort`	`{"type": "featured", "order": "desc"}`

4.11.3 Example

The following example sorts the content

- First by price from high to low
- If the price is the same, sort by name alphabetically.

```
        "sort": {
          "value": [
            {"type": "price", "order": "desc"},
            {"type": "name", "order": "asc"}
          ]
        }
```

For additional examples, refer to Content Result Sorting.

4.12 n_item

4.12.1 Data structure

The following snippet shows the basic structure of n_item.

```
["n_item": int]?
```

`n_item` represents the total number of the items to be returned for all <content_type> requested in `content`.

4.12.2 Default value

Key	Value
`n_item`	`10`

4.12.3 Max value

The number of items requested has a direct impact on the latency of the request. We recommend a value of less than or equal to 48. Between 10 and 24 items is a typical number to use. Any number beyond 100 will return 100 results, however, anything above 48 items may degrade the latency for that request, and SLAs are not guaranteed.

4.12.4 Example

For an example, see Get Started – Querying Content.

4.13 page_number

4.13.1 Data structure

The following snippet shows the basic structure of page_number. See Data structure snippets for interpretation of blue symbols.

```
        ["page_number": int]?
```

`page_number` is the page number to be returned. The page contains a maximum of `n_item` items.

Usually, the number of items will be equal to `n_item`, with the following exceptions:

- Not enough items in the last page
- Querying mixed content and some content does not have enough pages

4.13.2 Default value

Key	Value
`page_number`	`1`

4.13.3 Example

For an example, see Get Started – Pagination.

4.14 preview (internal)

4.14.1 Data structure

The following snippet shows the basic structure of preview. See Data structure snippets for interpretation of blue symbols.

```
["preview": {
  "portal_user_id": "<portal_user_id>",
  ["widget_id": "<widget_id>"]?,
  ["variation_id": "<variation_id>"]?,
  ["behaviors": [<behavior_object>]]?,
  ["ignore_variation_behaviors": bool]?,
  ["time": long]?,
  ["preserve_empty_slots": bool]?,
  ["force_ce": bool]?
}]?
```

`preview` contains information to preview certain changes made in the Customer Engagement Console but not yet published.

`portal_user_id` is required, and is the id of the user making changes to the configuration in the Reflektion Portal.

`widget_id`, `variation_id` indicates which widget and variation tuple the preview should be applied to. Note that if the `request.widget.widget_id` doesn't match the `widget_id` for preview, then the standard published behaviors will apply. `widget_id` and `variation_id` are optional.

`variation_id` is optional. If not specified, the active variation is used at the time specified by `preview.time`. If `variation_id` is specified, `widget_id` must also be specified.

`behaviors` is optional, and represents an array of additional behavior objects that should be applied. This feature is not available to customers.

`ignore_variation_behaviors` is optional, the widget variations setup in the portal are ignored, and only the behavior. This feature is not available to customers.

`time` indicates the time that the preview should occur (i.e. preview scheduled variation). `time` should be a long that represents a timestamp in milliseconds (Epoch time).

`preserve_empty_slots` is optional. Default value is `false`. If the flag is set to true, then all the empty slots will be preserved in the search results and returned as 'None'. If the flag is not defined or is set to false, all empty slots are removed and results collapsed.

`force_ce` is optional. Default value is `false`. If the flag is set to `true`, then the API request will make a call to CERT (if `rfkid`/`widget_id` is present), bypassing the CE active flag.

4.14.2 Default value

Key	Value
`preview.variation_id`	`<current_active_variation>`
`preview.time`	`<current_time>`

4.14.3 Examples

The following example shows user 1 previewing variation that is active at 11 May 2017 09:00:00 GMT. All the behaviors that for active variation (including any active inherited behaviors) at this time (May 11, 2017 9AM) will be applied to this request.

```
        "preview": {
          "portal_user_id": "1",
          "time": 1494493200
        }
```

The following example shows user 1 previewing variation 10 on 11 May 2017 at 09:00:00 GMT. All the behaviors that for variation 10 (including any active inherited behaviors) at this time (May 11, 2017 9AM) will be applied to this request. The time should be within the bounds of the variations schedule.

```
        "preview": {
          "portal_user_id": "1",
          "variation_id": "10",
          "time": 1494493200
        }
```

4.15 context

4.15.1 Data structure

The following snippet shows the basic structure of context. See Data structure snippets for interpretation of blue symbols.

```
        ["context": {
          ["<context_type>": {<context_key_values>} ]*
        }]?
```

`context` contains extra information to make search and recommendation results more relevant.

Refer to Appendix B for details on <context_type>:

- User
- Browser
- Geo
- Store
- Fitment
- Page
- Channel

4.15.2 Example

Refer to Context.

4.16request_for

4.16.1 Data structure

The following snippet shows the basic structure of `request_for`. See Data structure snippets for interpretation of blue symbols.

```
        ["request_for":
          [["request_param"]+]
        ]?
```

`request_for` includes `request_param` in response. Returned request param will also contain fields autocorrected by Sitecore Discover. It can have any field listed in Request Data or a wild card `"all"`.

`request_for` in request adds a key `request_for` in response.

4.16.2 Example

```
        "request_for": ["all"]
```

```
        "request_for": ["query", "facets"]
```

4.17 context_values (Coming Soon)

4.17.1 Data structure

The following snippet shows the basic structure of `context_values`. See Data structure snippets for interpretation of blue symbols.

```
        ["context_values": {
           ["<context_key>": {
              ["field": {
                  ["value": [
                     ["<context_field>" ]+
                  ] ]?
              } ]?
           }]*
}]?
```

`context_values` contains any `context_key` with a list of `context_field` to be added to the response pulled from the context applied to the current request. `context_key` can be any context type listed in Appendix B. Context. `context_field` can be any field under the corresponding type in Appendix B. Context. If `context_field` is empty or not present, all the fields listed under the given context type are returned in the response.

Besides those in Appendix B. Context, an additional key called `hard_filter` can be used. `hard_filter` returns all the filters that have been configured for the current page in CEC. It is compulsorily applied to all search results widgets.

4.17.2 Example

The following is an example of a request for `geo.country` and `geo.city` and all fields in filter:

```
        "context_values": {
          "geo": {
             "field": {
               "value": [ "country", "city" ]
             }
          } ,
          "hard_filter": {}
        }
```

Chapter 5 Detailed Data Structure of Response

This section describes the detailed data structure of Response.

5.1 ts

5.1.1 Data structure

The following snippet shows the basic structure of ts:

```
        "ts": int
```

`ts` is the server timestamp in millisecond.

5.2 dt

5.2.1 Data structure

The following snippet shows the basic structure of dt.

```
        "dt": int
```

`dt` is the response processing time in millisecond.

5.3 err

5.3.1 Data structure

The following snippet shows the basic structure of err. See Data structure snippets for interpretation of blue symbols.

```
        ["err": "<error_message>" ]?
```

When the HTTP response status is 4XX or 5XX. The <error_message> contains extra information about the error.

5.4 msg

5.4.1 Data structure

The following snippet shows the basic structure of msg. See Data structure snippets for interpretation of blue symbols.

```
        ["msg": "<message>" ]?
```

This contains additional information about the response.

5.5 rid

5.5.1 Data structure

The following snippet shows the basic structure of rid.

```
        "rid": "<response_id>"
```

`rid` is a unique generated id string that identifies this response. This id is not human readable.

5.6 query2id

5.6.1 Data structure

The following snippet shows the basic structure of query2id. See Data structure snippets for interpretation of blue symbols.

```
[“query2id": {
  ["<query_type>": "<query_string_id>" ]+
} ]?
```

`query2id` is only returned when `query` is sent in the request. The id will be a unique identifier of the query_string sent in each query_type. The id is not human readable.

Example

```
"query2id": {
  "keyphrase": "<keyphrase_id_100>"
}
```

5.7 request_for

5.7.1 Data structure

The following snippet shows the basic structure of request_for. See Data structure snippets for interpretation of blue symbols.

```
        ["request_for": <request_used_for_this_response> ]?
```

`request_for` may be returned when `request_for` is sent in the request. The response may also contain modified keyphrase if you ask for suggestions in the request by including `suggestion` parameter.

Request:

```
        "request_for":["query"]}
```

Response:

```
"request_for": {
        "query": {
            "orig_keyphrase": "chacolate", // Keyphrase with a typo
            "keyphrase": "chocolate" // Autocorrected keyphrase
        }
    },
```

5.8 facet

5.8.1 Data structure

The following snippet shows the basic structure of facet. See Data structure snippets for interpretation of blue symbols.

```
        ["facet": {
          [<facet_type>: {
            "number_of_products": <number_of_products_with_this_facet_type>,
            "total": <number_of_facet_values>,
            "value": [[{<facet_value>} ]+],
            ["display_name": "<alternate_display_name>" ]?
          } ]*
        } ]?

        <facet_value>: object
        {
          "id": "<facet_value_id>",
          "text": "<display_text>,
          "count": <number_of_item_in_facet>,
          "in_content": "<content_type>",
          ["<other_fields>": "<field_value>", ]*
          ["min": int, ]?
          ["max": int, ]?
          ["sub": [[<facet_value> ]*] ]?
        }
```

`facet` is only returned when `facet` is sent in the request. `id` is a unique id correspond to the facet value. The `id` should be used to filter in future requests.

`number_of_products` is the number of products containing an attribute value with this facet type

`total` is the total number of facet values for that facet type.

`<other_fields>` will be returned when additional `field` is requested.

`min` and `max` will only be returned when the <facet_type> supports range (such as price).

`sub` will only be returned when the <facet_type> supports nested structure (such as category tree).

`display_name` can be configured to use a different name besides the returned <facet_type>, contact Reflektion for setup.

5.8.2 Examples

The following is an example of a term facet (discrete values):

- There are 5 items in Red color
- There are 2 items in Blue color
- There are 11 total types of colors, but only 2 are shown

```
        "facet": {
          "colors": {
            "number_of_products": 235,
            "total": 11,
            "value": [
              {"id":"cr1", "text": "Red", "count": 5},
              {"id":"cr2", "text": "Blue", "count": 2}
            ]
          }
        }
```

The following is an example of a range facet:

- The total number of buckets is 2
- The minimum of the price range is 10
- The maximum of the price range is 50
- Suggested buckets are in value:
  - Suggested to group into 2 buckets
  - First bucket from 10 to 20
  - Second bucket from 20 to 50

```
      "facet": {
        "price": {
          "number_of_products": 27,
          "total": 2,
          "min": 10,
          "max": 50,
          "value": [
            {"id": "p1", "text": "10 - 20", "count": 5,
             "min": 10, "max": 20},
            {"id": "p2", "text": "20 - 50", "count": 2,
             "min": 20, "max": 50}
          ]
        }
      }
```

The following is an example of a nested facet (hierarchical structure).

There are four categories in the example:

- Mens > Shirt
- Mens > Pants
- Women > Shirt
- Women > Pants

```
      "facet": {
        "category_tree": {
          "value": [
            {
              "id": "ct1",
              "text": "Mens",
              "count": 5,
              "sub": [
                {"id": "ct11", "text": "Shirt", "count": 2},
                {"id": "ct12", "text": "Pants", "count": 3}
              ]
            },
            {
              "id": "ct2",
              "text": "Women",
              "count": 2,
              "sub": [
                {"id": "ct21", "text": "Shirt", "count": 1},
                {"id": "ct22", "text": "Pants", "count": 1}
```

```
                    ]
                }
            ]
        }
    }
```

5.9 Facet_names

5.9.1 Data structure

The following snippet shows the basic structure of facet_names. See Data structure snippets for interpretation of blue symbols.

```
        ["facet_names": [[{<facet_type>} ]*] ]?
```

`facet_names` is only returned when `facet` is sent in the request.

Each `<facet_type>` corresponds to a facet returned under the facet key. If the order of facets is configured in the system, the order of items in `facet_names` reflects the configured order of the returned facets. If the system is not explicitly configured, the order represents the default recommended order.

5.9.2 Example

The following shows how facet_names determines the order of returned facets. The json keys under `facet` are not relied upon to determine facet order, `facet_names` shows the correct order:

- brand
- price
- size
- color

```
        "facet": {
          "price": …,
          "brand": …,
          "size": …,
          "color": …
        },
        "facet_names": ["brand", "price", "size", "color"]
```

5.10 filter

5.10.1 Data structure

The following snippet shows the basic structure of filter. See Data structure snippets for interpretation of blue symbols.

```
["filter": {
  [<filter_type>: {
    ["display_name": "<display_name>" ]?
    "value": [[{<filter_value>} ]+]
  } ]*
} ]?

<filter_value>: object
{
  "filter_id": "<filter_value_id>",
  "text": "<display_text>,
  ["min": int, ]?
  ["max": int, ]?
}
```

`filter` is only returned when `filter` is sent in the request.

`display_name` is an alternate name to display for front-end purposes. (Contact Reflektion for setup if you would like to use a different name besides the returned <filter_type>.)

`filter_id` is a unique id correspond to the filter value. The `filter_id` should be used to filter in future requests. If `facet` is sent in the request, the `filter_id` in the filter response will be the same as the `id` in the facet response.

`filter_value` is the `min` and `max` will be returned if this filter is a range filter.

5.11 suggestion

5.11.1 Data structure

The following snippet shows the basic structure of suggestion. See Data structure snippets for interpretation of blue symbols.

```
        ["suggestion": {
          [<suggestion_type>: {
            "value": [[{<suggestion_value>} ]+]
          } ]*
        } ]?

        <suggestion_value>: object
        {
          "id": "<suggestion_value_id>",
          "text": "<display_text>,
          "in_content": "<content_type>",
          ["with_filter": {
            "<filter_type>": <filter_value>
          }]?
        }

        <filter_value>: object
        {
          "filter_id": "<filter_value_id>",
          "text": "<display_text>,
          ["min": int, ]?
          ["max": int, ]?
        }
```

`suggestion` is only returned when `suggestion` is sent in the request. `id` is a unique id corresponding to the suggested value. The `id` or `text` may be used to query in future requests.

`with_filter` is only returned if this suggestion should be combined with a specific filter. The `filter_type` is the type of filter this suggestion should be applied with. The `filter_value` is the same as the `filter_value` in the `filter` response.

5.11.2 Example

```
        "suggestion": {
          "query_expansion": {
            "value": [
              {"id": "pkel1Q", "text": "Men's Shirt", "in_content": "product"},
              {"id": "msa2fwS", "text": "Men's Styles", "in_content": "article", "with_filter":
 {"filter_id": "clothing_id123", "text": "Clothing"}}
            ]
          }
        }
```

5.12 autocomplete

Data structure

The following snippet shows the basic structure of autocomplete. See Data structure snippets for interpretation of blue symbols.

```
        ["autocomplete": "<autocomplete_text>" ]?
```

`autocomplete` is only returned when suggestion is sent in the request regardless of the suggestion_type requested. The <autocomplete_text> is the text of the autocomplete suggestion. It is the same as requesting for autocomplete suggestion and take the first `text`.

5.13 content

5.13.1 Data structure

The following snippet shows the basic structure of content. See Data structure snippets for interpretation of blue symbols.

```
        ["content": {
          ["<content_type>": {
            "value": [
              [{<content_result>} ]+
            ],
            "total_item": int,
            "n_item": int,
            ["fields": [<content_field_names>]]?
          } ]+
        } ]?
```

content is only returned when content is sent in the request. Each content_type is returned in a separate key, with a value object. The value object contains two extra key total_item and n_item.

The value of value object is a list of objects, each <content_result> object will contain all the fields of the result.

total_item is the total number of items that matches the query for this content_type.

n_item is the number of items returned in this response for this content_type.

fields is returned if the get_fields flag is true in the request. fields is a list containing all of the possible content fields (note that not all content will have a value for every content field). By default, Sitecore Discover is configured to return a default set of content fields (content_field_names) in the result of content. You can determine all the possible content fields that can be returned by obtaining a list of fields. Contact Reflektion to set up your custom content fields, if desired.

5.13.2 Example

```
        "content": {
          "product": {
            "total_item": 15,
            "n_item": 2,
            "value": [
              {"name": "shirt", "price": 10},
              {"name": "shirt2", "price": 20}
            ]
          },
          "article": {
            "total_item": 5,
            "n_item": 1,
            "value": [
              {"title": "how to match shirts", "img": "http://img.jpg"}
            ]
          }
        }
```

5.14redirect url

5.14.1 Data structure

The following snippet shows the basic structure of redirect_url. See Data structure snippets for interpretation of blue symbols.

```
        ["redirect_url": <url>]?
```

`redirect_url` is only returned when `query.keyphrase` is sent in the request, and the query is determined to match a redirect rule. The API client can decide if it should present search results from the api, or immediately navigate to the url specified by redirect_url.

5.15 widget_title (Deprecated - this support has been removed)

5.15.1 Data structure

The following snippet shows the basic structure of widget_title. See Data structure snippets for interpretation of blue symbols.

```
        ["widget_title": "<widget_title>" ]?
```

`widget_title` is only returned when `widget` is sent in the request, and your titles have been set up. (Contact Reflektion to set up your custom widget titles, if desired.)

5.16widget

5.16.1 Data structure

The following snippet shows the basic structure of the widgetof widget See Data structure snippets for interpretation of blue symbols.

```
        ["widget": {
          ["rfkid": "<value>",]?
          ["widget_id": "<value>"]?
          ["variation_id": "<value>"]?
          ["experiment_id": "<value>"]?
          ["experiment_bucket": "<value>"]?
        }]?
```

`widget` is only returned when `widget` is sent in the request,. `widget` contains all the information as the request. This makes it easier to see which widgets and/or rfk_ids generated the response. In addition, if there is a running experiment, it would contain the experiment details as well.

Key	Value
`rfkid`	A Discover id associated with the widget. Copied from the widget as sent in the request if available.
`widget_id`	A unique id string assigned to a widget by Sitecore Discover. Copied from the widget as sent in the request if available.
`variation_id`	A unique identifier string representing the variation that is applied to this response. A variation is a group of behaviors/rules defined for a given widget in Customer Engagement Console.
`experiment_id`	A unique id string that indicates the split test /experiment entity that this response belongs to. Split tests are defined in Customer Engagement Console. It is only returned when an experiment is running.
`experiment_bucket`	A unique generated id string that indicates the bucket for the running experiment that this response belongs to. It is only returned when an experiment is running. This id is not human readable.
`type`	A Sitecore Discover defined widget type. Some of the values are: "preview_search", "content_grid", "recommendation", "banner", "html_block". Widget type list will continue to grow, and you should not depend on a fixed list of types.

5.17 n_item

5.17.1 Data structure

The following snippet shows the basic structure of n_item. See Data structure snippets for interpretation of blue symbols.

```
        ["n_item": int ]?
```

`n_item` is only returned when `content` is sent in the request. This is the total number of content items returned in this response. It is calculated from the sum of all `content.<content_type>.n_item`

5.18 total_item

The following snippet shows the basic structure of total_item. See Data structure snippets for interpretation of blue symbols.

```
        ["total_item": int ]?
```

`total_item` is only returned when `content` is sent in the request. This is the total number of content items that matches the request. It is calculated from the sum of all `content.<content_type>.total_item`

5.19 page_number

The following snippet shows the basic structure of page_number. See Data structure snippets for interpretation of blue symbols.

```
        ["page_number": int ]?
```

page_number is only returned when `content` is sent in the request. This is the page number of this response.

5.20 total_page

The following snippet shows the basic structure of total_page. See Data structure snippets for interpretation of blue symbols.

```
        ["total_page": int ]?
```

`total_page` is only returned when `content` is sent in the request. This is the total number of pages available for the request.

5.21 url

The following snippet shows the basic structure of url. See Data structure snippets for interpretation of blue symbols.

```
        ["url": "<url_of_this_response>" ]?
```

`url` is only returned when `content` is sent in the request. This is the html encoded url representation of the request. You use the value to query the same request in url content type.

5.22 batch

5.22.1 Data structure

The following snippet shows the basic structure of batch. See Data structure snippets for interpretation of blue symbols.

```
["batch": [
  [{<response_key>: <response_value>}]+
]]?
```

`batch` is only expected when multiple responses are clubbed together. For example, in case of recommendations, response might be sent for similar and bundle items widget in a batch response as opposed to two individual responses.

It's an array of objects where the keys of the object represent the key that needs to be overridden for a specific response in the batch and value represents the overridden value.

5.22.2 Example

The following example overrides content object for a batch of 2 requests.

It means that for the first response, "shirt" and "shirt2" are the `product` contents. For the second response, "shirt3" and "shirt4" are the `product` contents. All other response key and value are the same and are outside of the `batch` key.

```
"batch": [
   {
     "content": {
        "product": {
            "total_item": 15,
            "n_item": 2,
            "value": [
                 {"name": "shirt", "price": 10},
                 {"name": "shirt2", "price": 20}
            ]
        }
   },
   {
     "content": {
        "product": {
            "total_item": 15,
            "n_item": 2,
            "value": [
                 {"name": "shirt3", "price": 40},
                 {"name": "shirt4", "price": 50}
            ]
        }
   }
]
```

5.23 errors

5.23.1 Data structure

The following snippet shows the basic structure of errors. See Data structure snippets for interpretation of blue symbols.

See Appendix C: Error for more details of the error format and various types of errors that can be returned.

```
        ["errors": [
          [
            {
               "message": "<message_string>",
               "type": "<error_type_string>",
               "code": int,
               "severity": "<severity_string>",
        [
                 "details": {
                     "<additional_information_key>":
"<additional_information_value>"
                 }
               ]?
            }
          ]+
        ]]?
```

`errors` object is only expected if any error was encountered during the processing of a request. It is an array of objects, whre each object represents a single type of error enriched with a message, description and a severity level.

5.23.2 Example

The following example shows a scenario where the page uri in the context of the request, `/random`, was not recognized by Sitecore Discover. Assuming this was the only error, we would get back `errors` as an array of a single object.

```
        "errors": [
           {
             "message": "page not found for uri",
             "type": "uri_not_found",
             "code": "1008",
             "severity": "high",
             "details": {
               "uri": "/random"
             }
           }
        ]
```

5.24 context_values

5.24.1 Data structure

The following snippet shows the basic structure of contect_values. See Data structure snippets for interpretation of blue symbols.

```
        ["context_values":
           ["<context_key>": {
               ["<context_field>": <context_field_value> ]+
           }]?
         }]?
```

`context_values` is only returned when context_values is sent in the request and contains at least one key.

5.25 appearance

5.25.1 Data structure

The following snippet shows the basic structure of `appearance` returned in the response. See Data structure snippets for interpretation of blue symbols.

```
[“appearance":
  ["templates": {
    ["<section>": {
      ["devices": {
        ["<device>": {
          ["<content>": string]+
        }]+
      }]?
    }]?
  }]?
  ["<variables>": {
     ["<variable>": {
       ["value": string, ]?
       "valueType": string,
       "expanded": boolean,
       "required": boolean
     }]?
   }]?
   ["<css_names>": [string]+]?
   ["<html_names>": [string]+]?
]?
```

`Appearance` is only returned when appearance is sent in the request. By default, appearance only returns css_names and html_names. To get further information, a request must contain at least one of `templates` or `variables` defined.

5.25.2 Examples

```
"appearance": {
   "templates": {}
   "variables": {}
}
```

```
        "appearance": {
          "css_names": [
            "rfk2_seo",
            "rfk2_hs_home_seo"
          ],
          "html_names": [
            "seo",
            "hs_home_seo"
          ],
          "templates": {
            "html": {
              "devices": {
                "mobile": {
                  "content": "<title>Reflektion</title><meta content=\"Reflektion Demo\"
name=\"description\"><meta content=\"{{$keywords}}\" name=\"keywords\">\n"
                },
                "tablet": {
                  "content": "<title>Reflektion</title><meta content=\"Reflektion Demo\"
name=\"description\"><meta content=\"{{$keywords}}\" name=\"keywords\">\n"
                }
              }
            }
          },
          "variables": {
            "keywords": {
              "valueType": "paragraph",
              "required": false,
              "expanded": false
            },
            "description": {
              "valueType": "paragraph",
              "required": false,
              "expanded": true,
              "value": "Reflektion Demo"
            },
            "title": {
              "valueType": "string",
              "required": false,
              "expanded": true,
              "value": "Reflektion"
            }
          }
        }
```

Chapter 6 Avanced Use Cases

This section describes some advanced use cases.

6.1 Limiting Returned Content Attribute

As mentioned in Querying Content, items in the response will contain a default set of attributes of the items. If a different subset of the attributes is required, a key field should be added.

6.1.1 Example: Search for 2 articles about movies, only return the title

The following is a continuation of the Search for 2 articles about movies example.

Request:

```
        {
          "query": {
            "keyphrase": ["movies"]
          },
          "content": {
            "article": {}
          },
          "n_item": 2
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_q2",
          "url": "/searchpage?keyphrase=movies&content=article",
          "query2id": {
            "keyphrase": "keyphrase_id_movies"
          },
          "content": {
            "article": {
              "value": [
                {"title": "Top 10 movie for 2016", "author": "John Smith", "published_data":
 "2016-12-31"},
                {"title": "Finding memo review", "category_name": "movie review"}
              ]
            }
          },
          "n_item": 2,
          "total_item": 3,
          "page_number": 1,
          "total_page": 2
        }
```

Even though only a subset of the attributes are returned, the search is performed in all attributes. In the response, *"Finding memo review"* is still returned because it belongs to the *"movie review"* category as indicated in the Search for 2 articles about movies example.

6.2 Querying Mixed Content

Multiple content types can be queried in one request. In `content,` include all the interested content types as key with an object as value.

If no `max` is specified for the content_type, it will be a number determined by Sitecore Discover.

6.2.1 Example: Search for the keyphrase *new* in *product* and *article*

In this example, we request 6 items in total: 1 article and any number of product results.

Request:

```
{
  "query": {
    "keyphrase": ["new"]
  },
  "content": {
    "article": {"max": 1},
    "product": {}
  },
  "n_item": 6
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_mx1",
  "url": "/searchpage?keyphrase=new&content=cidaApAs",
  "query2id": {
    "keyphrase": "keyphrase_id_new"
  },
  "content": {
    "product": {
      "value": [
        {"name": "new style shorts", "color": "red", "price": 25, "image_url": "",
"rating": 5},
        {"name": "red hat", "category": "new arrival", "price": 10, "rating": 1},
        {"name": "(NEW) baseball hat", "price": 50, "image_url": ""}
      ]
    },
    "article": {
      "value": [
        {"title": "new year outfit", "description": "trending style for this year"}
      ]
    }
  },
  "n_item": 6,
  "total_item": 65,
  "page_number": 1,
  "total_page": 11
}
```

6.3 Content Result Sorting

The following examples are a continuation of the Search for red products. Return 3 items per page example.

- Sort by Name from A to Z
- Sort by Price from Low to High
- Sort by multiple sort_type

6.3.1 Example: Sort by Name from A to Z

Request:

```
{
  "query": {
    "keyphrase": ["red"]
  },
  "n_item": 3,
  "content": {},
  "sort": {
    "type": "name",
    "order": "asc"
  }
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_150",
  "url": "/searchpage?keyphrase=red&sort=na39dC",
  "query2id": {
    "keyphrase": "keyphrase_id_red"
  },
  "content": {
    "product": {
      "value": [
        {"name": "jacket", "color": "red", "price": 80 },
        {"name": "pants", "price": 50, "category_name": ["red"] },
        {"name": "red dress", "price": 100}
      ]
    }
  },
  "n_item": 3,
  "total_item": 5,
  "page_number": 1,
  "total_page": 2
}
```

6.3.2 Example: Sort by Price from High to Low

Request:

```
{
  "query": {
    "keyphrase": ["red"]
  },
  "n_item": 3,
  "content": {},
  "sort": {
    "type": "price", (or final_price)
```

```
            "order": "desc"
          }
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_151",
          "url": "/searchpage?keyphrase=red&sort=pas31pJ",
          "query2id": {
            "keyphrase": "keyphrase_id_red"
          },
          "content": {
            "product": {
              "value": [
                {"name": "red dress", "price": 100},
                {"name": "jacket", "color": "red", "price": 80 },
                {"name": "pants", "price": 50, "category_name": ["red"] }
              ]
            }
          },
          "n_item": 3,
          "total_item": 5,
          "page_number": 1,
          "total_page": 2
        }
```

6.3.3 Example: Sort by both price and name, first Price from High to Low, if the price is the same sort by name from A to Z

Request:

```
        {
          "query": {
            "keyphrase": ["red"]
          },
          "n_item": 3,
          "content": {},
          "sort": {
            "value": [
              {"type": "price", "order": "desc"},
              {"type": "name", "order": "asc"}
            ]
          }
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_151",
          "url": "/searchpage?keyphrase=red&sort=pas31psdpDa",
          "query2id": {
            "keyphrase": "keyphrase_id_red"
          },
          "content": {
            "product": {
              "value": [
                {"name": "dress", "price": 100, "color": "red"},
                {"name": "red dress", "price": 100},
                {"name": "jacket", "color": "red", "price": 80 }
              ]
            }
          },
          "n_item": 3,
          "total_item": 5,
          "page_number": 1,
          "total_page": 2
        }
```

6.4 Boolean Operation

For advanced use cases, boolean operations are supported in `query` and `filter`.

This section contains information on the generic structure. For specific example, refer to Boolean Operation in Query and Boolean Operation in Filter.

The base data structure for `query` and `filter` is a lit.

In the list, the element can be either a string or a list. Anything in a string is interpreted by Sitecore Discover without explicit boolean control of the user.

Example: blue shirt:

```
        ["blue shirt"]
```

An OR operation applies to all the elements in the base list.

For example: blue OR shirt:

```
        ["blue", "shirt"]
```

An AND operation applies to all the elements in the sub list.

For example: blue AND shirt:

```
        [["blue", "shirt"]]
```

AND and OR operations can be used at the same time.

For example: (blue AND shirt) OR jacket:

```
[["blue", "shirt"], "jacket"]
```

6.5 Boolean Operation in Query (Deprecated)

Refer to Querying Content for basic querying and Boolean Operation for the basic boolean structure.

6.5.1 Example: Search for 3 products with red OR shorts

Request:

```
{
  "query": {
    "keyphrase": ["red", "shorts"]
  },
  "n_item": 3,
  "content": {}
}
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_bo1",
          "url": "/searchpage?keyphrase=red shorts",
          "query2id": {
            "keyphrase": "keyphrase_id_red_shorts"
          },
          "content": {
            "product": {
              "value": [
                {"name": "shorts", "color": "blue", "price": 25, "image_url": "", "rating": 5,
... },
                {"name": "hat", "color": "red", "price": 10, "rating": 1, ... },
                {"name": "red shorts", "price": 50, "image_url": "", ... }
              ]
            }
          },
          "n_item": 3,
          "total_item": 90,
          "page_number": 1,
          "total_page": 30
        }
```

The keyphrase **red** and **shorts** are searched in all attributes. If either one appears in any attribute, the item will be matched.
Note:
The item with name *red shorts* is returned because an OR operation is applied instead of an XOR operation. XOR operation is currently not supported.

6.5.2 Example: Search for 3 products with red AND shorts

Request:

```
        {
          "query": {
            "keyphrase": [["red", "shorts"]]
          },
          "n_item": 3,
          "content": {}
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_bo2",
          "url": "/searchpage?keyphrase=red&keyphrase=shorts",
          "query2id": {
            "keyphrase": "keyphrase_id_red__shorts"
          },
          "content": {
            "product": {
              "value": [
                {"name": "product 1", "color": "red", "price": 25, "gender": "men", "rating":
5, "category_name": ["Men shorts"] },
                {"name": "men red shorts", "price": 10, "rating": 1, ... },
                {"name": "stylish shorts", "color": "red", "price": 50, "gender": "women", ...
}
              ]
            }
          },
          "n_item": 3,
          "total_item": 45,
          "page_number": 1,
          "total_page": 15
        }
```

The keyphrase **red** and **shorts** are searched in all attributes. If both terms appears in any attribute of the same product, the item will be matched. Note that the item with name *product 1* and *stylish shorts* is returned because AND operation doesn't limit the keyphrase to be in the same attribute

6.6 Facet Order

Refer to Get Started - Facet for basic facet use case.

If a different facet sorting order is required, use the `facet.sort` key. The `facet.sort` from the enclosing object applies to all facets. You can overwrite this `facet.sort` on a per facet basis by specifying in the value key.

The following examples are a continuation of the Get 3 facet from each facet type from product content example.

- Sort all facet by text in alphabetical order
- Different sort for different facet

6.6.1 Example: Getting 3 facets from each facet type from product content. Sort all facet by text in ascending order (alphabetical)

Request:

```
{
  "facet": {
    "all": true,
    "max": 3,
    "sort": {
      "type": "text",
      "order": "desc"
    }
  }
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_fo1",
  "facet": {
    "colors": {
      "value": [
        {"id": "color_id_blue", "text": "Blue", "count": 11},
        {"id": "color_id_red", "text": "Red", "count": 23},
        {"id": "color_id_yellow", "text": "Yellow", "count": 10}
      ]
    },
    "product_type": {
      "value": [
        {"id": "prod_type_id_earring", "text": "Earring", "count": 535},
        {"id": "prod_type_id_necklace", "text": "Necklace", "count": 45},
        {"id": "prod_type_id_ring", "text": "Ring", "count": 105}
      ]
    },
    "primary_stone_type": {
      "value": [
        {"id": "stone_id_crystal", "text": "Crystal", "count": 779},
        {"id": "stone_id_diamond", "text": "Diamond", "count": 2374},
        {"id": "stone_id_emerald", "text": "Emerald", "count": 50}
      ]
    },
    "primary_metal_type": {
      "value": [
        {"id": "stone_id_aluminum", "text": "Aluminum", "count": 4},
              {"id": "stone_id_brass", "text": "Brass", "count": 80},
              {"id": "stone_id_gold", "text": "Gold", "count": 45}
            ]
          }
        },
        "facet_names": ["colors", "product_type", "primary_stone_type",
"primary_metal_type"]
      }
```

6.6.2 Example: Getting 3 facets from each facet type from product content. Sort color facet by count in descending order

The following is an example of request to:

- sort color facet by count in descending order
- sort all other facet by text in ascending order (in alphabetical order)
- and to get total values for all facet types.

The `sort` order is applies to all facet_types. If a different `sort` order is required for a specific facet, the sort order needs to be specified in the `value` key with the facet type explicitly stated.

Request:

```
      {
        "facet": {
          "all": true,
          "max": 3,
          "total": true,
          "sort": {
            "type": "text",
            "order": "desc"
          },
          "colors": {
            "sort": {
              "type": "count",
              "order": "desc"
            }
          }
        }
      }
```

Response:

```
      {
        "ts": 1480977544,
        "rid": "response_id_f02",
        "facet": {
          "colors": {
            "total": 11,
            "value": [
              {"id": "color_id_red", "text": "Red", "count": 23},
              {"id": "color_id_blue", "text": "Blue", "count": 11},
              {"id": "color_id_yellow", "text": "Yellow", "count": 10}
            ]
          },
          "product_type": {
            "total": 30,
            "value": [
              {"id": "prod_type_id_earring", "text": "Earring", "count": 535},
              {"id": "prod_type_id_necklace", "text": "Necklace", "count": 45},
              {"id": "prod_type_id_ring", "text": "Ring", "count": 105}
              ]
            },
            "primary_stone_type": {
              "total": 80,
              "value": [
                {"id": "stone_id_crystal", "text": "Crystal", "count": 779},
                {"id": "stone_id_diamond", "text": "Diamond", "count": 2374},
                {"id": "stone_id_emerald", "text": "Emerald", "count": 50}
              ]
            },
            "primary_metal_type": {
              "total": 36,
              "value": [
                {"id": "stone_id_aluminum", "text": "Aluminum", "count": 4},
                {"id": "stone_id_brass", "text": "Brass", "count": 80},
                {"id": "stone_id_gold", "text": "Gold", "count": 45}
              ]
            }
          },
          "facet_names": ["colors", "product_type", "primary_stone_type",
"primary_metal_type"]
        }
```

6.7 Facet Type Control

Refer to Get Started - Facet for basic facet use case.

By default, Sitecore Discover only returns requested facet_type . To request specific facet types only, set the all attribute to **1** and specify the required facet types in value.

The following examples are a continuation of the Getting 3 facet from each facet type from product content example.

- Get only color facet
- Get different number of values for each facet

6.7.1 Example: Get 3 facet from color facet from product content

Request:

```
{
  "facet": {
    "colors": {"max": 3}
  }
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_181",
  "facet": {
    "colors": {
      "value": [
        {"id": "color_id_red", "text": "Red", "count": 23},
        {"id": "color_id_blue", "text": "Blue", "count": 11},
        {"id": "color_id_yellow", "text": "Yellow", "count": 10}
      ]
    }
  },
  "facet_names": ["colors"]
}
```

6.7.2 Example: Get only two facets from product content color and product_type

The following example requests to return 5 facets for color, and for product_type, return 3 facets.

Request:

```
{
  "facet": {
    "colors": {
"max": 5
    },
    "product_type": {
"max": 3
    }
  }
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_182",
  "facet": {
    "colors": {
      "value": [
        {"id": "color_id_red", "text": "Red", "count": 23},
        {"id": "color_id_blue", "text": "Blue", "count": 11},
        {"id": "color_id_yellow", "text": "Yellow", "count": 10},
        {"id": "color_id_black", "text": "Black", "count": 8},
        {"id": "color_id_white", "text": "White", "count": 1}
      ]
    },
    "product_type": {
      "value": [
        {"id": "prod_type_id_ring", "text": "Ring", "count": 105},
        {"id": "prod_type_id_necklace", "text": "Necklace", "count": 45},
        {"id": "prod_type_id_watch", "text": "Watch", "count": 5}
      ]
    }
  },
  "facet_names": ["colors", "product_type"]
}
```

6.8 Facet Option Control

Refer to Get Started - Facet for basic facet use case.

The values of facets returned are determined by the `sort` order and `max`. If a fixed list of facet is required, they can be specified in a value object inside the value object. The fixed list can be either the facet IDs or the facet values. We encourage the use of facet ID.

If a fixed list of facet is provided, the returned facet will be in the same order regardless of `sort`. If the fixed list of facet is less than the `max`, the remaining facets will be dynamic and conform to `sort`.

The following examples are a continuation of the Getting 3 facet from each facet type from product content example:

- Getting fixed facet values that are available
- Getting 3 fixed facet from color facet regardless of item count
- Getting a mix of fixed facet and dynamic facet
- Getting facet with a minimum count of 10
- Getting total number of facet values for all facets
- Getting total number of facet values for one facet only

6.8.1 Example: Getting only white, red, black facet from color facet

Request:

```
{
  "facet": {
    "colors": {
      "max": 3,
      "value": ["color_id_white", "color_id_red", "color_id_black"]
    }
  }
}
```

Alternative request:

```
{
  "facet": {
    "colors": {
      "max": 3,
      "value": ["white", "red", "black"]
    }
  }
}
```

Response:

```
{
  "ts": 1480977544,
  "rid": "response_id_fc1",
  "facet": {
    "colors": {
      "value": [
        {"id": "color_id_white", "text": "White", "count": 1},
        {"id": "color_id_red", "text": "Red", "count": 23},
        {"id": "color_id_black", "text": "Black", "count": 8}
      ]
    }
  },
  "facet_names": ["colors"]
}
```

The colors are returned in the same order as the value provided.

6.8.2 Example: Getting 3 fixed facet (white,black, grey) from color facet regardless of item count

Request:

```
  {
    "facet": {
      "colors": {
        "min_count": 0,
        "max": 3,
        "value": ["color_id_white", "color_id_black", "color_id_grey"]
      }
    }
  }
```

Note:
To get a facet that does not contain any item (where returned count = 0), set the `min_count` attribute to.

Response:

```
  {
    "ts": 1480977544,
    "rid": "response_id_186",
    "facet": {
      "colors": {
        "value": [
          {"id": "color_id_white", "text": "White", "count": 1},
          {"id": "color_id_black", "text": "Black", "count": 8},
          {"id": "color_id_grey", "text": "Grey", "count": 0}
        ]
      }
    },
    "facet_names": ["colors"]
  }
```

6.8.3 Example: Getting 3 fixed facet (white, red, black) from color facet and 2 dynamic color

Request:

```
  {
    "facet": {
      "colors": {
        "max": 5,
        "value": ["color_id_white", "color_id_red", "color_id_black"]
      }
    }
  }
```

Response:

```
  {
    "ts": 1480977544,
    "rid": "response_id_185",
    "facet": {
      "colors": {
        "value": [
          {"id": "color_id_white", "text": "White", "count": 1},
          {"id": "color_id_red", "text": "Red", "count": 23},
          {"id": "color_id_black", "text": "Black", "count": 8},
          {"id": "color_id_blue", "text": "Blue", "count": 11},
          {"id": "color_id_yellow", "text": "Yellow", "count": 10}
        ]
      }
    },
    "facet_names": ["colors"]
  }
```

6.8.4 Example: Getting 3 color facet where the facet have at least 10 item in it

Request:

```
  {
    "facet": {
      "colors": {
        "min_count": 10,
        "max": 3,
        "value": ["color_id_white", "color_id_red", "color_id_black"]
      }
    }
  }
```

Response:

```
  {
    "ts": 1480977544,
    "rid": "response_id_186",
    "facet": {
      "colors": {
        "value": [
          {"id": "color_id_red", "text": "Red", "count": 23},
          {"id": "color_id_blue", "text": "Blue", "count": 11},
          {"id": "color_id_yellow", "text": "Yellow", "count": 10}
        ]
      }
    },
    "facet_names": ["colors"]
  }
```

6.8.5 Example: Getting total number of facet values for all facets

Request:

```
  {
    "facet": {
      "total": true,
      "colors": {
        "max": 1,
      },
      "genders": {
        "max": 2
      }
    }
  }
```

Response:

```
      {
        "ts": 1480977544,
        "rid": "response_id_186",
        "facet": {
          "colors": {
            "total": 11,
            "value": [
              {"id": "color_id_red", "text": "Red", "count": 23}
            ]
          },
          "genders": {
            "total": 4,
            "value": [
              {"id": "gender_id_male", "text": "Men's", "count": 20},
              {"id": "gender_id_female", "text": "Women's", "count": 21}
            ]
          }
        },
        "facet_names": ["colors", "genders"]
      }
```

6.8.6 Example: Getting total number of facet values for one facet only

Request:

```
      {
        "facet": {
          "colors": {
            "total": true,
            "max": 1,
          },
          "genders": {
            "max": 2
          }
        }
      }
```

Response:

```
      {
        "ts": 1480977544,
        "rid": "response_id_186",
        "facet": {
          "colors": {
            "total": 11,
            "value": [
              {"id": "color_id_red", "text": "Red", "count": 23}
            ]
          },
          "genders": {
            "value": [
              {"id": "gender_id_male", "text": "Men's", "count": 20},
              {"id": "gender_id_female", "text": "Women's", "count": 21}
            ]
          }
        },
        "facet_names": ["colors", "genders"]
      }
```

6.9 Boolean Operation in Filter (Deprecated)

Refer to Get Started - Filter for basic filtering and Boolean Operation for the basic boolean structure.

Examples:

- OR operation
- AND operation
- A mix of AND and OR operation

6.9.1 Example: Filter on (red OR blue) color for shirt keyphrase in product

Request:

```
        {
          "query": {
            "keyphrase": ["shirt"]
          },
          "filter": {
            "colors": {
              "value": ["color_id_red", "color_id_blue"]
            }
          },
          "content": {},
          "n_item": 3
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_bo3",
          "url": "/searchpage?keyphrase=shirt&filter=cidrob",
          "query2id": {
            "keyphrase": "keyphrase_id_shirt"
          },
          "filter": {
            "colors": {
              "value": ["color_id_red", "color_id_blue"]
            }
          },
          "content": {
            "product": {
              "value": [
                {"name": "shirt 1", "price": 55, "image_url": "/img/shirt_1.jpg", "rating": 5,
"color": "red"},
                {"name": "shirt 2", "price": 60, "rating": 1, "color": "blue"},
                {"name": "shirt 3", "price": 10, "color": ["red", "blue"]}
              ]
            }
          },
          "n_item": 3,
          "total_item": 10,
          "page_number": 1,
          "total_page": 4
        }
```

6.9.2 Example: Filter on (red AND blue) color for shirt keyphrase in product

Request:

```
        {
          "query": {
            "keyphrase": ["shirt"]
          },
          "filter": {
            "colors": {
              "value": [["color_id_red", "color_id_blue"]]
            }
          },
          "content": {},
          "n_item": 3
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_bo4",
          "url": "????",
          "query2id": {
            "keyphrase": "keyphrase_id_shirt"
          },
          "filter": {
            "colors": {
              "value": [["color_id_red", "color_id_blue"]]
            }
          },
          "content": {
            "product": {
              "value": [
                {"name": "shirt 3", "price": 10, "color": ["red", "blue"]},
                {"name": "shirt 4", "price": 55, "image_url": "/img/shirt_4.jpg", "rating": 5,
"color": ["red", "blue", "white"]},
                {"name": "shirt 5", "price": 60, "rating": 1, "color": ["red", "yellow",
"blue"]}
              ]
            }
          },
          "n_item": 3,
          "total_item": 5,
          "page_number": 1,
          "total_page": 2
        }
```

6.9.3 Example: Filter on ((red AND blue) OR yellow) color for shirt keyphrase in product

Request:

```
        {
          "query": {
            "keyphrase": ["shirt"]
          },
          "filter": {
            "colors": {
              "value": [["color_id_red", "color_id_blue"], "color_id_yellow"]
            }
          },
          "content": {},
          "n_item": 3
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_bo5",
          "url": "????",
          "query2id": {
            "keyphrase": "keyphrase_id_shirt"
          },
          "filter": {
            "colors": {
              "value": [["color_id_red", "color_id_blue"], "color_id_yellow"]
            }
          },
          "content": {
            "product": {
              "value": [
                {"name": "shirt 11", "price": 10, "color": ["red", "yellow"]},
                {"name": "shirt 4", "price": 55, "image_url": "/img/shirt_4.jpg", "rating": 5,
"color": ["red", "blue", "white"]},
                {"name": "shirt 5", "price": 60, "rating": 1, "color": ["red", "yellow",
"blue"]}
              ]
            }
          },
          "n_item": 3,
          "total_item": 8,
          "page_number": 1,
          "total_page": 3
        }
```

6.10 Context

Context will provide more information to Sitecore Discover to make the search result more relevant to the query and the context.

The following example is a continuation of the Search for 3 products with red AND shorts example, where we apply a gender context to the result.

6.10.1 Example: Search for 3 products with red AND shorts with user context (gender information)

Request:

```
        {
          "query": {
            "keyphrase": [["red", "shorts"]]
          },
          "n_item": 3,,
          "content": {},
          "context": {
            "user": {
              "gender": "female"
            }
          }
        }
```

Response:

```
        {
          "ts": 1480977544,
          "rid": "response_id_17",
          "url": "/searchpage?keyphrase=red&keyphrase=shorts&context=Q1fcsl087pn",
          "query2id": {
            "keyphrase": "keyphrase_id_red__shorts"
          },
          "content": {
            "product": {
              "value": [
                {"name": "stylish shorts", "color": "red", "price": 50, "category_names":
["women clothing"]},
                {"name": "red shorts", "price": 40, "gender": "female"},
                {"name": "women's beach shorts", "color": "red", "price": 45}
              ]
            }
          },
          "n_item": 3,
          "total_item": 45,
          "page_number": 1,
          "total_page": 15
        }
```

In the response, the top 3 products returned are now related to gender *female*.

Chapter 7 Appendix A: Types

7.1 Query Types

The following are different query types that can be used to search.

Type	Description
keyphrase	General text search
category	Represents the category that the content should belong to
uri	Url that is returned in any previous response

7.2 Content Types

The following are currently supported contents types. Each of the content type requires a feed or other data source. Additional content types can be supported on a per customer basis.

Type	Description
product	Product catalog
article	Article content

7.3 Suggestion Types

The following are different types/ algorithms for generating the suggestion.

Type	Description
keyphrase	A mix of suggestion from "type_correction", "autocomplete", "query_expansion", "related_search"
category	Categories that contain the searched term
trending_category	Category that is trending now
autocomplete	Completion of keyphrase to a term that we have result
recent	Recent search history

7.4 Facet/ Filter Types

The following are standard facets that are supported across customers (if available). Additional facets are further available depending on the industry the customer belongs to. Customer specific facets can be supported on a per customer basis when provided in the catalog feed.

Type
brand
category_names
price
rating

7.5 Sort Types

The following are typical sorting orders that are supported. Refer to the Customer Engagement Console to see the various possible sort Sort Choices applicable for your account.

Name	Description
featured	Sort by a mix of vendor rules, popular, individualized
name	Sort by name
price	Sort by price
newest	Sort by recently added
rating	Sort by rating
review	Sort by number of reviews

Chapter 8 Appendix B: Context

8.1 Context Types

A context object has the following context types.

Field	Description
user *Context User Object*	Describes the user details on whose behalf the request is being made.
browser *Context Browser Object*	Describes the browser details that the user is using.
geo *Context Geo Object*	Describes geographic information like IP and location information about the user.
store *Context Store Object*	Describes the store id that the user has chosen to see results for.
fitment *Context Fitment Object*	Fitment information that the user has chosen to see results for.
page *Context Page Object*	Describes the page details like page url, device, etc.
campaign *Context Campaign Object*	Describes the campaign details like utm_source etc.
channel *Context Channel Object*	Describes the channel details.

8.2 Context - User

The following are the keys supported in the user context object.

Field	Description
user_id *string*	user_id as identified by the developer. Note: One of the ids (`user_id` or `uuid`) must be specified to allow tracking user activity and assigning users to active experiments if any.
uuid *string*	user id as identified by Sitecore Discover. Note: One of the ids (`user_id` or `uuid`) must be specified to allow tracking user activity and assigning users to active experiments if any.
gender *string*	Either Male/ Female
email *string*	Email of the user
order_id *list of string*	The order id(s) that is associated with this request
new_user *boolean*	This boolean flag indicates if the current user is a new user
groups *list of string*	List of custom groups or segments assigned to user.

8.3 Context - Browser

The following are the keys supported in the browser context object.

Field	**Description**
user_agent *string*	The browser user agent. Must follow industry convention to ensure `device` and `app_type` can be detected automatically.
device *string*	Either • pc • tablet • mobile By default, this is computed automatically from `user_agent`.
app_type *string*	Either • browser • webview • native By default, this is computed automatically from `user_agent`. browser: user_agent starts with "Mozilla" and doesn't contain one of the webview keywords (see below) webview: user_agent starts with "Mozilla" and contains "WebViewApp" or "wv" keywords in the string. native: user_agent doesn't start with "Mozilla".

8.4 Context - Geo

The following are the keys supported in the geographic context object. This is the geographic location of the end user.

Note: Sitecore Discover's AI system takes into account geo information provided in the request and events. If the request contains inconsistent information, the Sitecore Discover system will not be able to utilize geo-specific intelligence.

Field	Description
ip *string*	IP address (optional) The IP address is the IP address of the user that is sent by the browser or native application. Do NOT pass the IP address of your server or your proxy. If IP is not specified, it will be read from X-Forwarded-For header of the request. If there is no X-Forwarded-For header, the IP of the request will be used as default.
country *string*	2 Letter Country Code (ISO Alpha-2) By default, this is computed automatically from the IP address.
state *string*	State or province name (also referred as region) Note: this should be full name, not a state code. Example: "North Dakota", not "ND" By default, this is computed automatically from the IP address.**.**
city *string*	City name Note: this should be a full name, not a code. By default, this is computed automatically from the IP address.
zip *string*	Zip/Postal code of the location By default, this is computed automatically from the IP address.

8.5 Context - Store

The following are the keys supported in the store context object.

Field	Description
id *string*	Store id. This id should be consistent with the id shared as part of product feed. B2B websites use this to identify a restriction id associated with a customer. If restriction is based on the customer id, specify the customer id. If you have a restriction id that is different then customer id, use the specific restriction id.
group_id *string*	Store group id. If your stores are organized in two level hierarchy (for example as zones and stores), you can pass the top level organization identifier as group_id. This id should be consistent with group_id shared as part of product feed. B2B websites, if your restrictions are organized in two-levels, then pass the restriction group id as group_id. If restriction is based on the customer group id, specify the customer group id. If you have a restriction group id that is different then customer group id, use the specific restriction group id.

Sitecore Discover system can be configured for one level or two level store information.

One Level Store Information

If you have only one level of store information, then use `id` to specify the store identifier. Do not use `group_id`.

For B2B sites, if you have availability/restrictions or price organized by a customer or customer' contract id, pass the identifier in `id`. It is common to identify a restriction based on a customer id or customer's contract id, but is not necessary.

Two Level Store Information

If you have store information like price organized at a regional level and at store level, then use `group_id` to specify the store group identifier, and use `id` to specify store identifier.

For B2B sites, if you have availability or pricing organized by contract group/customer groups as well as specific contract or specific customer, then use `id` to specify the contract/customer id, and `group_id` to specify the customer/contract group identifier.

8.6 Context – Fitment

Following are the keys supported in fitment context.

Field	Description

ids
array of string
	An array strings representing customer's fitment ids. Usually, there would be only one fitment ID, as user would have selected only one fitment as part of the interaction on the website.
	
	`ids` should be used when Customer have a separate fitment entity in their system and associate an id for each fitment. Such fitment feed must be provided as part of the product feed to Sitecore Discover.
	
	Ex:
	`"context": {`
	`    "fitment": {`
	`      "ids": ["fitid1", "fitid2"]`
	`    }`
	`  }`

items
array of fitment objects
	An array of fitment objects. This attribute is provided in the rare case when a customer doesn't have fitment ids associated with fitments in their system. `ids` should be used where possible.
	
	Note: Customer should provide only `ids` or `items`, but not both. If both are provided, `ids` are honored, and `items` are ignored.
	
	Usually, there would be only one fitment object, as the user would have selected only one fitment as part of the interaction on the website.
	
	Customers that don't have a separate fitment entity should send fitment attributes part of the context. Note: all fitment attributes must be sent in the object. Partial information would be discarded.
	
	Ex:
	`"context": {`
	`    "fitment": {`
	`      "items": [`
	`        {`
	`          "year": "1990",`
	`          "make": "Honda",`
	`          "model": "H1"`
	`        },`
	`        {`
	`          "year": "2015",`
	`          "make": "Ford",`

	```
	          "model": "F100"
	        }
	      ]
	    }
	  }
	```

8.7 Context - Page

The following are the keys supported in the page context object.

Field	Description
referrer *string*	The page referrer
uri *string*	The url of the current page. Note that the uri should correspond to a page set or single page defined in the Merchandising Control Center.
title *string*	The title of the current page
sku *list of string*	The list of products that are on this page/in the cart
container_id *string*	The id of the page set or single page. We recommend you use `uri` instead of `container_id`.
all_category_ids *string*	The category id (as assigned by Sitecore Discover) of a given category page. Note: Discover assigns a unique id for every category. We recommend you use `uri` instead of `all_category_ids`.
locale_country *string*	The country specified as part of the locale context as per ISO 3166 Alpha 2 country code (e.g. us, ca, ...).
locale_language *string*	The language specified as part of the locale context as per ISO 639-1 standard language codes (e.g. en, fr, ...).

8.8 Context - Campaign

The following are the keys supported in the campaign object.

Field	Description
utm_source *string*	Paid campaign source identifier. This information may be provided in the campaign context or as part of the request url.
utm_campaign *string*	Paid campaign identifier. This information may be provided in the campaign context or as part of the request url.

8.9 Context - Channel

The following are the keys supported in the channel context object.

Field	**Description**
type *string*	One of the following: ● notification_bar ● chatbot ● web_page

8.10 Context - Weather (Deprecated)

The following are the keys supported in the weather context object.

Field	Description
description *string*	One of the following: • Sunny • Cloudy • Overcast • Drizzle • Rain • Snow • Hail • Thunderstorm
temp *string*	Temperature. An integer followed by "C" or "F". (i.e. "13C", "55F")

Chapter 9 Appendix C: Error

A response may contain an errors object, that would contain details about all errors encountered while fulfilling the request. Sitecore Discover attempts to serve the request partially even when there are errors.

Note that errors associated with 500 range of HTTP status may not follow the prescribed format as these could be caused by many layers of load balancers, gateways, proxies, etc.

9.1 Error Format

Following is the description of the individual error object within the errors array.

Name	Description
`message` *string*	Always returned. The error message.
`type` *string*	Always returned. Indicates the kind of error that occurred (string representation).
`code` *int*	Always returned. Indicates the kind of error that occurred (int representation).
`severity` *string*	Optional. Represents the severity level of an error. Valid values: `low` - `medium` - `high` *Note: This is only a hint. Severity is usually context dependent, and you should determine the severity of each error for your use case.*

`details`
object
	Optional. An object with further details about the error. The object is a map of key/value.
	
	Example:
	
	`"details": {`
	`    "rfk_id": "rfkid_9"`
	`}`

9.2 Error Type

Code	Type	Message	Description	Troubleshooting tips
101	domain_id_not_found	domain_id not found	Corresponds with HTTP 404. Not a valid domain_id.	Check the Domain id and try again.
102	bad_request	bad request	Corresponds with HTTP 400. Request was not properly formed.	Check the details in the error, and try again.
103	degraded_response	degraded response	Corresponds with HTTP 200. This is not an error. Response did not fully honor all the configurations in CEC. This usually happens when one or more services the system depends on was not available or was busy.	If it occurs for more than a few minutes, check the Discover service status page, or contact Reflektion support.
1000	bad_context	'context' in request is not properly formed	Corresponds with HTTP 400. It's not a valid json message.	Review how `context` is formatted.
1001	internal_server_error	internal server error while processing request	Corresponds with HTTP 500. Something is broken. This is usually a temporary error, for example in a high load situation or if an endpoint is temporarily having issues.	If it occurs for more than a few minutes, contact Reflektion support.
1003	incorrect_recipe_id	incorrect recipe id for widget	Corresponds with HTTP 500. Recipe is not properly defined for v2 domain.	Review recipe definition in CEC for the given rfk id
1004	missing_store_param	store params not found in context	Corresponds with HTTP 400. Store params are not passed in request.	Domain is configured to require store id/store group id. Review the settings in

				CEC if this was not intended.
1005	missing_required_params	None of the required parameters are found in context: uri | container_id | widget_id | rfk_ids	Corresponds with HTTP 400.	Request must contain at least one of the contexts mentioned. Check your request and try again.
1006	rfkid_not_found	rfk id not found	Corresponds with HTTP 404. No widget could be retrieved from the passed rfk ids	Review if Domain has any of the passed rfk ids in CEC.
1007	widget_not_found	widget_id not found	Corresponds with HTTP 404. No widget could be retrieved from the passed widget id	Review if Domain has the passed widget id in CEC.
1008	uri_not_found	page not found for uri	Corresponds with HTTP 404. No page could be retrieved from the passed uri	Review if Domain Pages are properly configured in CEC.
1009	container_not_found	page not found for container id	Corresponds with HTTP 404. No page could be retrieved from the passed container id	Review if Domain has the passed container id in CEC.
1010	sku_not_found	product not found for sku	Corresponds with HTTP 207. Product could not be retrieved from the passed sku	Review if Domain has the passed sku in CEC.
1011	product_group_not_found	product not found for product group	Corresponds with HTTP 207. Product could not be retrieved from the passed product group	Review if Domain has the passed product group in CEC
1012	category_not_found	category not found	Corresponds with HTTP 207. Category could not be retrieved	Review if Domain has the category id in CEC
1013	widget_not_in_page	widget not in page	Corresponds with HTTP 207. Page corresponding to passed uri does not contain the required rfk id/widget id	Rreview if Domain Pages are properly configured in CEC

1014	error_retrieving_widget_by_rfk_id	internal server error retrieving widget by rfk_id	Corresponds with HTTP 500. Something is broken. This is usually a temporary error, for example in a high load situation or if an endpoint is temporarily having issues.	If it occurs for more than a few minutes, contact support.
1015	error_retrieving_widget_by_widget_id	internal server error retrieving widget by widget_id	Corresponds with HTTP 500. Something is broken. This is usually a temporary error, for example in a high load situation or if an endpoint is temporarily having issues.	If it occurs for more than a few minutes, contact support.

9.3 HTTP Status

HTTP Status	Description
`OK` 200	Request was successful. Note that partial results may be returned when 200 is specified. Always check for errors even when you receive 200 and evaluate if the errors and determine action based on the relevance to the specific request. *In future, 200 will mean there were no errors, and 207 will mean, there are errors for part of the request.*
`OK` 207	A Multi-Status response conveys information about multiple resources in situations where multiple status codes might be appropriate. You should check for errors and determine action based on the relevance to the specific request. Note: Currently, 200 is returned when there are partial errors instead of 207. In future we will be updating the response code to be 207. Ensure you check 200 and 207 for errors.
`Bad Request` 400	The request was invalid or cannot be otherwise served. An accompanying error message will explain further.
`Unauthorized` 401	Missing or incorrect authentication credentials.
`Not Acceptable` 406	Returned when an invalid format is specified in the request.
`Request Timeout` 408	Client uploading too much data and timed out
`Too Many Request` 429	Reached the rate limit. Try again later.
`Internal Error` 500	Error on Sitecore Discover side. Something is broken. This is usually a temporary error, for example in a high load situation or if an endpoint is temporarily having issues. Try again later.
`Bad Gateway` 502	Sitecore Discover Gateway is misconfigured, authentication token could be validated. Clear your cache and cookies, and try again.
`Server Busy`	Reflektion server is busy. Try again later.

503	
`Operation Timeout` 504	Reflektion server took too long to respond. Try again later.

Chapter 10 Appendix D: Internal or Load Test Requests

There are many cases when customers don't want requests from certain users or test traffic affecting or influencing analytics or datasets. Customers can identify such requests in two ways

1. By registering specific IP addresses with Reflektion to be tagged as internal/load test requests and events. Contact Reflektion Support.
2. By providing an indication using rfk_flags when making a request. Ensure that these flags are only sent upon identification of internal or load test traffic, accidentally flagging real requests will lead to data loss.

10.1 Flag Specification

The flag can be sent by one of the following means. Note that the flags sent by multiple means have a precedence order as listed in the table below.

Mechanism	Key	Description / Purpose
URL Query Parameter	rfk_flags	Specify rfk_flags as a query parameter when making a request. Example: `GET {API_URL}?`**`data`**`={<request>}&`**rfk_flags=cust_internal**

Request Body	rfk_flags	Specify rfk_flags as part of the request.
		
		Example Request:
		`{`
		`  "context": {`
		`    "page": {"uri": "/search"}`
		`  },`
		`  "widget": {`
		`     "rfkid": "rfkid_7"`
		`  },`
		`  "content": {`
		`     "product": {}`
		`  },`
		`  "rfk_flags": "cust_internal"`
		`}`
		
		Curl Example:
		
		curl '{API_URL}'
		-H 'accept-encoding: gzip, deflate, br' \
		-H 'content-type: application/json' \
		-H 'accept: application/json'
		--data-binary
		$'{"data":{"context":{"page":{"uri":"/search"}},"widget":{"rfkid":"rfkid_7"},"content":{"product":{}},**"rfk_flags":"cust_internal"**}}' --compressed

HTTP Header	x-rfk-flags	Specify x-rfk-flags in the header. Curl Example: curl '{API_URL}' **-H 'x-rfk-flags: cust_internal'** \ -H 'accept-encoding: gzip, deflate, br' \ -H 'content-type: application/json' \ -H 'accept: application/json' --data-binary $'{"data":{"context":{"page":{"uri":"/search"}},"widget":{"rfkid":"rfkid_7"},"content":{"product":{}}}}' --compressed

Cookie	__rfk_flags	Specify __rfk_flags in a cookie. Curl Example: curl '{API_URL}' **--cookie "__rfk_flags=cust_internal"** \\ -H 'accept-encoding: gzip, deflate, br' \\ -H 'content-type: application/json' \\ -H 'accept: application/json' \\ --data-binary $'{"data":{"context":{"page":{"uri":"/search"}},"widget":{"rfkid":"rfkid_7"},"content":{"product":{}}}}' --compressed

Supported customer flag values are:

Flag	Flag value	Description / Purpose
Customer internal	“cust_internal”	Let the Discover system know that the traffic is from internal users. These requests will be ignored from analytics. Note that it will still count towards your usage of the service.
Customer loadtest	“cust_loadtest”	Let the Discover system know the traffic is from load testing. These requests will be ignored from analytics. Note that it will still count towards your usage of the service.

10.2 Sample JS Implementation using a cookie

Customers using JS implementation can set RFK flag cookie by adding the following script before the beacon:

```
        function setRFKCookie(name, value) {
            value = encodeURIComponent(value);
            value += ';path=/;';
            if (name)
              document.cookie = name + "=" + value;
        }

        If ( user is identified as internal or load test user ) {
        setRFKCookie('__rfk_flags', '<value>')
        }
```

where value can be `cust_internal` or `cust_loadtest` or both like `cust_internal|cust_loadtest`.

Call `setRFKCookie` only when the user is known to be an internal user or it is a load test.