Search and Retrieve API

The Search and Retrieve API provides a means for looking up content from an application based on search criteria including related content and returning the relevant fields for a specific use case. The application can be an iOS or Android mobile app, a web application or a web site requesting the content. The Search and Retrieve API can be used to search for any content, but it is especially useful for headless content because it allows for referenced content to be included and it also offers field filtering. It is very fast thanks to its use of the Apache Solr search engine and because it retrieves content via the XperienCentral cache. The search queries are concise and written either in JSON, which is usually used when querying from code, or in YAML which is easier to read and write when developing or testing queries.

In This Topic

Requirements and Installation

The Search and Retrieve API functionality requires XperienCentral versions R32 and higher or versions R27 and higher in combination with the Headless Add-on versions 2.2.5 and higher. The setting `search_retrieve_enabled` in the `headless_search_retrieve` section on the General tab of the Set up Tool must be enabled. A free tool that is useful for sending these types of requests is Postman.

Back to top

Configuration

In addition to the options to enable the Search and Retrieve API, the `headless_search_retrieve` section on the General tab of the Setup Tool contains two configuration options:

`max_items_in_result`	Limits the total number of content items (pages and media items) returned in the response, regardless of those specified in the `from` and `to` parameters. (See Search Parameters for details on those parameters.) When the limit has been reached, no more references defined in the `includes` section will be "expanded".
`default_search_result_amount`	The default number of results that are returned when no `from` and/or `to` parameters have been provided in the query.

Clustered environments

`contentindex_index_readonly_nodes`	*The following applies to XperienCentral versions R37 and higher.* When using the Search and Retrieve API on a clustered environment, this setting should be enabled. When you enable this setting in an existing environment, you need to manually rebuild the content index. To do so, delete the `<webmanager-root>/work/contentindex` directory and then restart XperienCentral. The content index will then be regenerated. You need to perform this step even if the `application_settings.contentindex_queue_empty_reindex` setting on the General tab of the Setup Tool is enabled.

Back to top

Basic Usage

To send an HTTP POST request to https://yoursite.com/web/searchretrieve/v1/yaml, the search query would be:

YAML

```
search:
  text: The text to search for in your content
```

JSON

```
{
  "search": {
    "text": "The text to search for in your content"
  }
}
```

This query will return JSON containing the contents of all content items matching the text in all languages. For example:

```
{
  "results" : [ {
    "_searchRetrieve" : {
      "contentItemID" : "M123",
      "language" : "nl_NL",
      "success" : true
    },
    ... JSON contents of this item ...
  }, {
    "_searchRetrieve" : {
      "contentItemID" : "P12345",
      "language" : "en_US",
      "success" : true
    },
    ... JSON contents of this item ...
  } ]
}
```

To retrieve the JSON content, all presentations must be headless (JSON). This means the page presentation as well as all the required element presentations. To retrieve results 11 through 20 of content items in English with the keywords "news" and "sports", the query would be:

YAML

```
search:
  keywordsAnd:
  - news
  - sports
  languages:
  - en_US
  from: 11
  to: 20
```

JSON

```
{
  "search": {
    "keywordsAnd": [
      "news",
      "sports"
    ],
    "languages": [
      "en_US"
    ],
    "from": 11,
    "to": 20
  }
}
```

To retrieve the Dutch content item versions for the content items with ID M1 and P26111, the query would be:

YAML

```
search:
  ids:
  - M1
  - P26111
  languages:
  - nl
```

JSON

```
{
  "search": {
    "ids": [
      "M1",
      "P26111"
    ],
    "languages": [
      "nl"
    ]
  }
}
```

Back to top

Releases and Work in Progress

The first version of the Search and Retrieve API was released in XperienCentral version R32. It contains the functionalities described here and is intended to be used in a development environment but should not be used in a production environment. The following functionality will be added in later versions in order to make the API production ready:

- Rate limitations to protect the server from Denial of Service attacks (DDoS) and other security enhancements
- The inclusion of referenced items in the response - *available since 2.2.7*
- Field filtering to only retrieve relevant fields in the response - *available since 2.2.7*
- Support for more types of search criteria including custom fields - *available since 2.2.6*
- Limited support for non-JSON content items

If you have access to the GX Software Jira issue tracking system, see https://jira.gxsoftware.com/jira/browse/XA-636 and its sub-tasks for all the technical details. The XperienCentral Headless add-on, which contains this functionality, can be upgraded without upgrading XperienCentral.

Back to top

Search Requests

URL

A search query is constructed using an HTTP POST request to a URL as follows:

```
https://yoursite.com/web/searchretrieve/v1/yaml
```

Both HTTPS and HTTP are accepted as well as other context paths. To construct a search query in JSON format, use a URL that ends in `/json`. For example:

```
https://yoursite.com/web/searchretrieve/v1/json
```

Search Query

YAML

The examples here are written in YAML as well as JSON, since both formats are supported. In general, YAML is easier to read and write and can be written quickly which makes it useful for doing search queries by hand. JSON is the more logical choice when serializing an object for the search query in your code.

The following YAML:

YAML

```
search:
  ids:
  - M123
  - P12345
  languages:
  - nl
```

translates to this JSON:

JSON

```
{
  "search": {
    "ids": [
      "M123",
      "P12345"
    ],
    "languages": [
      "nl"
    ]
  }
}
```

Notice that the content of an object (map) is on a separate level in the YAML example and that its own indent level and list items are denoted by hyphens ("-"). You can easily convert between these two formats at json2yaml.com.

IDs and ID-based Lookup

There are various different IDs used in XperienCentral for different purposes. Content Item IDs are used to request specific content items via this API, often in combination with a specific language which will return the current active version in that language. Version IDs are never used in the API. A content item is either a page or a media item. Since the numerical ID used for pages and media items might overlap, this internal ID is prefixed by either a "P" (for pages) or an "M" (for media items) in this API. If the ID of a content item is already known from a previous search request, it can easily be retrieved using the ID's parameter as shown in the example above.

Languages and "Display-on" Pages

Query results can be limited to either one or a number of explicit languages using the `languages` parameter. See Search Parameters below for details on the format. A media item is rendered via its "display-on" page. The language of the media item determines the language version of the display-on page, therefore you must ensure that a display-on page is available for each supported language.

Back to top

Responses

Successful Responses

A successful response contains a results field that contains a list of results . A successful result always starts with the following field:

```
    "_searchRetrieve" : {
      "contentItemID" : "M123",
      "language" : "nl_NL",
      "success" : true
    }
```

As you can see, the response contains the Content Item ID, language and a field indicates whether it was successful. The JSON that comes thereafter is determined by the JSON presentation of the content item. See also the examples above for sample responses.

Response Item Errors

If there is an error for a specific item, the result for that item resembles the following:

```
  "_searchRetrieve" : {
    "error" : "A message describing the error",
    "success" : false
  }
```

Failed Search Request

Bad query

A search might fail due to a malformed search query. In that case, the response will contain an error message. Because an internal parse error is passed on, it looks similar to the following:

```
{
  "error" : "Error parsing search query: Unrecognized field \"idsxxx\" (class nl.gx.product.
wmaheadlesssearchretrieveapi.api.data.SearchDefinition), not marked as ignorable (9 known properties: \"
keywordCategories\", \"keywordsNot\", \"from\", \"text\", \"keywordsOr\", \"keywordsAnd\", \"ids\", \"
languages\", \"to\"])\n at [Source: java.io.StringReader@4e4183f0; line: 3, column: 4] (through reference
chain: nl.gx.product.wmaheadlesssearchretrieveapi.api.data.SearchAndRetrieveQuery[\"search\"]->nl.gx.product.
wmaheadlesssearchretrieveapi.api.data.SearchDefinition[\"idsxxx\"])"
}
```

In this example, the field name `idsxxx` was used in place of one of the valid ones listed (including IDs). The HTTP response code for invalid search query errors is "400, Bad Request".

Unexpected (internal) errors

Errors which are not due to an invalid search query will return the following general error message:

```
{
  "error" : "Internal Error"
}
```

These errors will have the response code "500, Internal Server Error".

Back to top

Search Parameters

The following are the currently supported parameters for querying the Search & Retrieve API.

Parameter	Description	YAML Example	JSON Example

`ids`	Search for a content item using the item's content ID. Note that this is not the content item version ID. When searching for a page, the ID must be prefixed with a "P" and when searching for a media item, the ID should be prefixed with an "M". For example, searching for a page would with the content ID 26111, you would use P26111 and searching for a media with the content ID of 1, you would use M1.
	
	This parameter can be used together with the `languages` parameter in order to retrieve the current or active version for those languages. When no languages are specified, the current version for all languages defined in XperienCentral are returned.
		`search:`
		`  ids:`
		`  - P26111`
		`  - M1`
			`{`
			`"search":`
			`{`
			`  "ids":`
			`[`
			`"P26111",`
			`  "M1"`
			`]}`
			`}`

`languages`	This parameter specifies the language you want the results to be returned in. This parameter supports both the short country code metatag value (ISO-639) as well as the full metatag value (ISO-639 and ISO-3166 separated by an underscore "_"). When the full metatag value is provided, only the country code will be used due to a limitation within the content index. The `languages` parameter can be used in combination with both ID-based queries and parameterized queries. If this parameter is omitted, the active version for each available language will be returned. If there is no active version available, no version will be returned.	`search:`
		`  languages:`
		`  - en_US`
		`  - nl`
			`{`
			`"search":`
			`{`
			`"languages": [`
			`"en_US",`
			`  "nl"`
			`]}`
			`}`

`text`	This parameter searches for any text in the title or the body of the documents in the content index.	`search:`
		`  text:`
		`lorem`
			`{`
			`"search":`
			`{`
			`"text":`
			`"lorem"`
			`  }`
			`}`

`keywords`
 	
`(simple configuration)`
	The simple configuration for the keywords parameter makes it possible to search for content containing a specific keyword. When searching for multiple keywords or for results excluding certain keywords, use the advanced configuration (below).	`search:`
		
		`keywords:`
		`term 1`
			`{`
			`"search":`
			`{`
			`"keywords": "term 1"`
			`  }`
			`}`

`keywords`
 	
`(advanced configuration)`
	Allows more complex querying for articles with or without certain keywords. The `and`, `not` and `or` parameters which support lists of keywords should be used in the respective query fields. Values should always be provided in a list (YAML) or array (JSON) format.	`search:`
		
		`keywords:`
		`  and:`
		`  -`
		`term 1`
		`  -`
		`term 2`
		`  not:`
		`  -`
		`term 3`
			`{`
			
			`"search":`
			`{`
			
			`"keywords": {`
			
			`"and": [`
			
			`"term 1",`
			
			`"term 2"`
			`  ],`
			
			`"not": [`
			
			`"term 3"`
			`  ]`
			`  }`
			`  }`
			`}`

`types`	Allows filtering on specific content types. The following parameters are allowed:
	
	• `page`
	• `download`
	• `article`
	• `image`
	• `media page`
	• `multimedia`
	• any custom content item identifier, for example `helloworldmediaitem`
	• any modular content type identifier, prefixed with `wmammodularcontent_`, for example `wmammodularcontent_news`
	• `mediaitem` - This type is a special case. If this parameter is provided, all types of media items will be returned. This parameter can not be used in combination with the other types. If other types are provided together with `mediaitem`, an exception will be thrown.
		`search:`
		`  types:`
		`  - page`
		`  - article`
		`  - download`
		`  - helloworldmediaitem`
			`{`
			
			`"search":`
			`{`
			
			`"types":`
			`[`
			
			`"page",`
			
			`"article",`
			
			`"download",`
			
			`"helloworldmediaitem"`
			`  ]`
			`  }`
			`}`

`keywordCategories`	Adds a list of term categories to the query. The result has to contain at least one term that is in one of the provided categories.	`search:`
		`  keywordCategories:`
		`  - term category 1`
		`  - term category 2`
			`{`
			`"search":`
			`{`
			
			`"keywordCategories": [`
			
			`"term category 1",`
			
			`"term category 2"`
			`  ]}`
			`}`

`sortBy`	Specifies which field should be used to sort the results. The following options are supported:
	
	• `lastModifiedDate`
	• `publicationDate`
	• `score` (default)
		`search:`
		
		`sortBy: publicationDate`
			`{`
			
			`"search":`
			`{`
			
			`"sortBy": "publicationDate"`
			`  }`
			`}`

`sortOrder`	Can be either `desc` (default) or `asc`.	`search:`
		
		`sortOrder: asc`
			`{`
			
			`"search":`
			`{`
			
			`"sortOrder": "asc"`
			`  }`
			`}`

`publicationDate`	Supports filtering the results by publication date. A `from` and/or a `to` parameter can be provided. If only a `from` parameter is provided, the results will contain documents that are published between the specified date and now. If only a `to` parameter is specified, all results with a publication date up to the provided date will be retrieved. Please note that dates should be provided in a ISO-8601 format.
	
	When using Javascript it's really easy to retrieve an ISO string for a Date object. The Date object has a function toISOString() that will return a proper ISO String, using UTC as the timezone. The provided time on the creation of the Date object is converted accordingly.
		`search:`
		
		`publicationDate:`
		
		`from: 2021-03-19T14:58+02:00`
		`  to: 2021-05-19T14:58Z`
			`{`
			
			`"search":`
			`{`
			
			`"publicationDate":`
			`{`
			
			`"from": "2021-03-19T14:58+02:00",`
			
			`"to": "2021-05-19T14:58Z"`
			`  }`
			`  }`
			`}`

`<custom fields>`
 	
`(simple configuration)`
	Custom fields can be used to query fields that are not explicitly exposed via the Search and Retrieve API. Examples of these are fields that are added to the content index via annotating methods in custom media items or fields in a Modular Content template that are configured to be indexed. The `custom` field option supports the same query parameter types as the keyword parameter (both a simple and an advanced query type). To query a custom field, use the identifier of the field as it's stored in the Content Index and provide the value to search for. See the column to the right for examples.
	
	Modular Content fields
	
	Fields in a Modular Content Template can be configured to be indexed in the content index automatically. The identifiers that should be used to query for these fields depend on the exact configuration of that field. In all cases, the identifier should be prefixed with `mcf_`. If the Search Index value in the configuration of the field is set to one of the "Unique ..." values, the identifier should also include the identifier of the Modular Template as follows:
	
	`modular_<identifier-of-the-template>0<identifier-of-the-field>`
	
	When the Search Index value is set to "Combined ...", the template identifier can be omitted as follows:`modular_<identifier-of-the-field>`
	
	Modular date fields
	
	If a Modular Content item has a field of type Date it is possible to search for those content items using a date range, similar to searching for an item's publication date. Please refer to the publicationDate parameter for more information. Searching for a date range can not be combined with other parameters within the same field.
	
	Discovering and Debugging Fields in the Content Index
	
	The Solr Maintenance Reusable allows the user to discover which fields are available within the content index. When this plugin is installed, your role requires the "Developer options" permission for the Solr Maintenance panel in the Authorization panel. Open the Solr Maintenance panel and navigate to **Developer Options > Explore the content index** to explore the content index. Set the radio button "Show all facet info" to "yes" to show a list of all available facets per result.
	
	Combining Multiple Custom Fields
	
	It's possible to query for multiple custom fields at the same time by combining both simple and advanced query parameters, as well as "regular" and Modular Content Fields.
		`search:`
		`  pagever`
		`sion_navi`
		`gationtit`
		`le:`
		`Navigatie`
		`titel`
		
		`search:`
		
		`modular_s`
		`tring4:`
		`value 1`
		
		`modular_n`
		`ews0strin`
		`g:`
		`  and:`
		`  -`
		`value 1`
		`  -`
		`value 2`
		
		`modular_d`
		`ate:`
		
		`from:`
		`2021-03-`
		`19T14:`
		`58+02:00`
		`  to:`
		`2021-05-`
		`19T14:58Z`
			`{`
			
			`"search":`
			`{`
			
			`"pageversion_navigationtitle":`
			`"Navigatie titel"`
			`  }`
			`}`
			
			`{`
			
			`"search":`
			`{`
			
			`"modular_string4":`
			`"value 1",`
			
			`"modular_news0string": {`
			
			`"and": [`
			
			`"value 1",`
			
			`"value 2"`
			`  ]`
			`  },`
			
			`"modular_date": {`
			
			`"from":`
			`"2021-03-19T14:58+02:00",`
			
			`"to" :`
			`"2021-05-19T14:58Z"`
			`  }`
			`}`
			`}`

<custom fields>
 	
(advanced configuration)
	Allows more complex querying for articles with fields (not) containing or one more specific values. Supports the `and`, `not` and `or` parameters, which in turn support lists of values which should be used in the respective query fields. Values should always be provided in a list (YAML) or array (JSON) format.	search:
		  pageversion_navigationtitle:
		Navigatie titel
		
		modular_string4:
		  and:
		  -
		value 1
		  -
		value 2
		
		modular_news0string:
		  not:
		  -
		value 3
			{
			
			"search":
			{
			
			"pageversion_navigationtitle":
			"Navigatie titel",
			
			"modular_string4":
			{
			
			"and": [
			
			"value 1",
			
			"value 2"
			]
			},
			
			"modular_news0string": {
			
			"not": [
			
			"value 3"
			]
			}
			}
			}

`from`	Allows the selection of a subset of the results starting at the value of this parameter. The first result in a result set has an index of 1, therefore the `from` parameter should always be 1 or higher. Also note that this parameter is inclusive, meaning that if `from` is 3, for example, the result set will begin with the third result. When `from` is omitted, the index value 1 is used.	search:
		  from: 3
			{
			"search":
			{
			
			"from": 3
			}
			}

`to`	Allows the selection of a subset of the results ending at the value of this parameter. Like the `from` parameter, `to` is inclusive, meaning that when it is set to 1, only the first result will be returned. For example, if `from` is set to 3 and `to` is set to 5, the results 3, 4 and 5 will be returned. The minimum value for this parameter is 1. When `to` is omitted, a maximum number of 100 results will be returned starting at the `from` parameter (if specified), otherwise starting from index value 1.	search:
		  to: 3
			{
			"search":
			{
			"to": 5
			}
			}

Back to top

Includes (Following References)

Other content items referenced in the search results can be included in the response by adding an `include` section to the search request. This can be applied to any string in the result which contains a reference to another content item in the form of a content item ID ("P12345" or "M123" for example).

Referring to Fields in Nested JSON Objects

The `parentPage` field in the standard "Headless Page" presentation is a good example of references to nested JSON objects:

```
{
  "results" : [ { ...
    "parentPage" : {
      "contentItemID" : "P93983",
      "language" : "nl_NL", ...
    }, ...
  } ]
}
```

In order to retrieve the content of that parent item, we can extend the search query with the include parameter by specifying the path of the field we want to include:

YAML

```
search: ...
include:
- field: parentPage/contentItemID
```

JSON

```
{
  "search": { ... },
  "include": [
    {
      "field": "parentPage/contentItemID"
    }
  ]
}
```

The `path` syntax refers to the nested JSON objects within each search result JSON object where the field `parentPage` contains a JSON object with the field `contentItemID` which contains the actual reference. The full contents of the referenced content item ("P93983" in this example) will be included in the response:

```
{
  "results" : [ { ...
    "parentPage" : {
      "contentItemID" : "P93983",
      "language" : "nl_NL", ...
    }, ...
  } ],
  "includes" : {
    "P93983" : {
      "_searchRetrieve" : { ... },
      "contents" : [ { ... } ],
    }
}
```

Accessing Included Content

An included content item can be accessed as shown below. For convenience, the `includes` object will also be available in the response when it's empty:

```
response.includes[contentItemID]
```

Building a breadcrumb path would look something like this:

```
function breadcrumb(response, contentItemID) {
    let includedItem = response.includes[contentItemID];
    return !includedItem ? "" : breadcrumb(response, includedItem.parentPage.contentItemID) + "/" +
includedItem.title;
}
```

Multiple Levels

A parent page often itself has its own parent page. By default, only one level of references is included, which means that the reference to that page is not replaced by the contents of the item. If you want to include the parent content item and its parent and so on all the way up to the homepage, you can include add a `levels` parameter like the following:

YAML

```
search: ...
include:
- field: parentPage/contentItemID
  levels: 100
```

JSON

```
{
  "search": { ... },
  "include": [
    {
      "field": "parentPage/contentItemID",
      "levels": 100
    }
  ]
}
```

One `level` means that all included fields have been followed once for each search result. In more complex cases where there are more fields included, the second level can therefore mean that one content item is included based on an include field and thereafter another content item is included based on another include field. For example, the subpages of the parent page would be included if the `levels` parameter for the subpages is at least 2. In this case the `levels` parameter for the parent page does not need to be higher than 1 because that reference is followed first.

References Contained in Arrays (Lists)

The `subPages` field in the standard "Headless Page" presentation is a good example of references contained in arrays:

```
{
  "results" : [ { ...
    "subPages" : [ {
      "contentItemID" : "P94012",
      "language" : "nl_NL",
      "title" : "Page, label 1",
      "url" : "/web/headless-root/terms/page-label-1.htm?channel=json"
    }, {
      "contentItemID" : "P94031", ...
    }, {
      "contentItemID" : "P94046", ...
    } ],
    "title" : "Terms", ...
  } ]
}
```

The interesting difference compared to `parentPage` is that the `subPages` field contains an array of (JSON) objects, one for each sub page, where the `contentItemID` field contains the reference. The referenced content items can be included in the result using this search request:

YAML

```
search: ...
include:
- field: subPages/contentItemID
```

JSON

```
{
  "search": { ... },
  "include": [
    {
      "field": "subPages/contentItemID"
    }
  ]
}
```

If you compare this to the `parentPage` example, you see that there is no difference except of course for the field name. The structure remains the same. The reason is that each object referred to using the "path" notation may be included in a list. The content IDs can also be in a list. Any combination of these cases and any number of nested lists are allowed.

Configuration

The `max_items_in_result` setting applies to `includes`.

Errors and Warnings in the Response

If the search request completely fails, the response will only contain a field named "error" containing an error message in place of the field `results` containing the search results. This can happen, for example if the search query contains unknown fields. If the search can be completed but problems are encountered during the processing, the search results will be returned in the `results` field but one or more warnings will also be returned in an array field *named* `warnings`. This can happen, for example if you include a field which does not contain a proper reference to another content item. The `warnings` array is always present in the response but is empty if there are no warnings.

Back to top

Filtering the Fields in the Response

When requesting data from an API you often do not need all the information that is returned by the API. In order to limit the size of the response, it is possible to specify the exact fields which should be returned by the query.

Discovering the Available Fields

The easiest way to discover which fields are available to filter on is to call the API with your query without a filter. Let's assume that a call to the API returns the following response:

Full response


```
{
  "results" : [ {
    "_searchRetrieve" : {
      "contentItemID" : "M6067",
      "language" : "nl_NL",
      "success" : true
    },
    "contents" : [ {
      "area" : "main",
      "elements" : [ ]
    } ],
    "language" : "nl_NL",
    "links" : [ ],
    "metadata" : {
      "Date" : {
        "Date" : "2021-08-27T00:00+02:00"
      },
      "contenttype" : "wmammodularcontent_date",
      "copyright" : "",
      "expiration_date" : "",
      "external_id" : "",
      "id" : 6067,
      "lastmodified_date" : "2021-08-24T17:03+02:00",
      "lead" : "Some small introduction text",
      "leadimage" : "/upload_mm/3/f/a/cid6067_index3.jpg",
      "publication_date" : "2021-08-24T14:35+02:00",
      "tags" : [ ],
      "title" : "Article title",
      "url" : "/web/myweb/date.htm?channel=json"
    },
    "navtitle" : "",
    "subtype" : "wmammodularcontent_datemediaitem",
    "title" : "Article title",
    "type" : "mediaitem",
    "url" : "/web/myweb/article.htm?channel=json"
  } ],
 ...
}
```


This response contains all fields that are available for this specific content item. Let's assume that only the title and URL of the article are required. The filter for retrievnig only these properties would look like the following:

YAML

```
search:
  ...
filter:
- title
- url
```

JSON

```
{
   "search": {
    ...
   },
   "filter": [
      "title",
      "url"
   ]
}
```

The above example filter returns the following JSON:

Response

```
{
  "results" : [ {
    "_searchRetrieve" : {
      "contentItemID" : "M6067",
      "language" : "nl_NL",
      "success" : true
    },
    "title" : "Article title",
    "url" : "/web/myweb/article.htm?channel=json"
  } ],
  "includes" : { },
  "warnings" : [ ]
}
```

Please note that the `_searchRetrieve` JSON object is always present in the response and can not be filtered out.

Nested Fields

It's also possible to filter more specifically on fields that contain nested fields like the metadata object in the example response above. If you wanted to create an overview of articles that contain not only a title of the article and a link to it but also the lead image and a short description of the article, the filter could be expanded as follows:

YAML

```
search:
  ...
filter:
- metadata:
  - lead
  - leadImage
- title
- url
```

JSON

```
{
   "search": {
      ...
   },
   "filter": [
      {
         "metadata": [
            "lead",
            "leadImage"
         ]
      },
      "title",
      "url"
   ]
}
```

This will return:

Response

```
{
  "results" : [ {
    "_searchRetrieve" : {
      "contentItemID" : "M6067",
      "language" : "nl_NL",
      "success" : true
    },
    "metadata" : {
      "lead" : "Some description of the article",
      "leadimage" : "/upload_mm/3/f/a/cid6067_index3.jpg"
    },
    "title" : "Date",
    "url" : "/web/myweb/date.htm?channel=json"
  } ],
  "includes" : { },
  "warnings" : [ ]
}
```

Differences Between Content Types

Not all content types contain the exact same fields which leads to differences in the exact JSON objects returned by the API. It is possible to create a filter that takes this into account. If a filter contains a field that is not present in a specific content item in the result, then that field is simply ignored. Let's take the response below as an example:

Response

```
{
  "results" : [ {
    "_searchRetrieve" : {
      "contentItemID" : "P26111",
      "language" : "nl_NL",
      "success" : true
    },
    "contents" : [ {
      "area" : "main",
      "elements" : [ {
        "html" : "<p class=\"anyipsum-output\">Bacon ipsum dolor amet t-bone leberkas  alcatra ham, short ribs
bacon cow brisket.  Tail andouille pancetta, pig  chislic beef ham hock ham strip steak.  Kevin doner tongue,
```

```
ham flank  pancetta chislic pork loin bacon tenderloin porchetta rump tri-tip.  Cupim hamburger salami prosciutto, shoulder ham andouille rump  frankfurter fatback chuck picanha ground round.  Corned beef bacon alcatra jowl meatloaf, kielbasa ground round short ribs pancetta  turducken.  Filet mignon fatback cow beef beef ribs, bacon jerky chicken  picanha kielbasa tongue ribeye landjaeger.  Alcatra hamburger beef ribs  tongue salami.</p><p data-wm-forced-paragraph=\"true\">Short ribs fatback pork loin, porchetta chislic  kielbasa pastrami landjaeger buffalo cow strip steak.  Pork belly  andouille ham hock short loin swine.  Ham hock shank pig, tail cow  chicken ham frankfurter.  Pancetta chislic ham hock turkey, flank beef  tongue tri-tip meatball biltong t-bone pork swine beef ribs.  Drumstick  kielbasa frankfurter sirloin turkey, buffalo boudin shoulder short ribs  short loin bacon salami.</p><p></p>",
        "type" : "text"
      }, {
        "alignment" : "CLEAR",
        "alternativeText" : "",
        "focuspoint-x" : -1,
        "focuspoint-y" : -1,
        "id" : 117842,
        "link" : { },
        "original_size" : "498x672",
        "source" : "/upload/653b33df-b487-45ef-b25d-ddf3a9c29fd3_index3.jpg",
        "subText" : "",
        "type" : "image"
      }, {
        "html" : "<p></p><p>Leberkas frankfurter beef ribs pork loin.  T-bone pork belly boudin corned beef tail cupim salami capicola swine.  Flank landjaeger bresaola meatloaf drumstick, sausage salami jerky  fatback chicken andouille kielbasa shank doner burgdoggen.  Corned beef  jerky flank, cupim doner bacon turducken.  Doner bacon jerky turducken  cow cupim.  Ham fatback pork belly, ham hock boudin shankle pastrami  tail.</p><p>Bresaola andouille biltong tongue, ground round pork chop  leberkas shankle sirloin picanha ham hock spare ribs.  Pig andouille cow  capicola tri-tip strip steak flank fatback picanha beef pork brisket.  Flank jerky porchetta tongue salami shank short ribs pancetta landjaeger  buffalo pork chop hamburger andouille rump.  Pork picanha brisket  drumstick, boudin pork chop prosciutto corned beef doner shoulder pig  pork loin.  Meatball tail beef ribs chislic, brisket doner rump cupim  shank chuck.</p><p>Boudin chuck pork, flank meatball prosciutto  landjaeger shank jowl beef kevin turducken.  T-bone ground round ribeye  kevin, venison landjaeger andouille chislic biltong pork spare ribs  fatback.  Hamburger beef chicken cow corned beef filet mignon.  Shoulder  turducken fatback pork belly porchetta doner pork chop.</p>",
        "type" : "text"
      } ]
    } ],
    "language" : "nl_NL",
    "languages" : [ {
      "id" : 43,
      "locale" : "nl_NL",
      "title" : "MyWeb",
      "url" : "/web/myweb.htm?channel=json",
      "value" : "dutch"
    }, {
      "id" : 42,
      "locale" : "en_US",
      "title" : "MyWeb",
      "url" : "/web/myweb.htm?channel=json",
      "value" : "english"
    } ],
    "links" : [ ],
    "navtitle" : "",
    "title" : "MyWeb",
    "type" : "page",
    "url" : "/web/myweb.htm?channel=json"
  }, {
    "_searchRetrieve" : {
      "contentItemID" : "M6067",
      "language" : "nl_NL",
      "success" : true
    },
    "contents" : [ {
      "area" : "main",
      "elements" : [ ]
    } ],
    "language" : "nl_NL",
    "links" : [ ],
    "metadata" : {
      "Date" : {
        "Date" : "2021-08-27T00:00+02:00"
```

```
      },
      "contenttype" : "wmammodularcontent_date",
      "copyright" : "",
      "expiration_date" : "",
      "external_id" : "",
      "id" : 6067,
      "lastmodified_date" : "2021-08-24T17:03+02:00",
      "lead" : "",
      "publication_date" : "2021-08-24T14:35+02:00",
      "tags" : [ ],
      "title" : "Date",
      "url" : "/web/myweb/date.htm?channel=json"
    },
    "navtitle" : "",
    "subtype" : "wmammodularcontent_datemediaitem",
    "title" : "Date",
    "type" : "mediaitem",
    "url" : "/web/myweb/date.htm?channel=json"
  } ],
  "includes" : { },
  "warnings" : [ ]
}
```


The response above contains two different content types: a page and a media item, specifically a `wmammodularcontent_datemediaitem`. The media item contains a `metadata` object whereas the page does not. The page contains a `languages` object and the media item does not. Assuming that the response should contain both these objects, the filter can be constructed as follows:

YAML

```
search:
  ...
filter:
- metadata
- languages
```

Javascript

```
{
   "search": {
      ...
   },
   "filter": [
      "metadata",
      "languages"
   ]
}
```

This filter results in the following response:

Filtered response


```
{
  "results" : [ {
    "_searchRetrieve" : {
      "contentItemID" : "P26111",
      "language" : "nl_NL",
      "success" : true
    },
    "languages" : [ {
      "id" : 43,
      "locale" : "nl_NL",
      "title" : "MyWeb",
      "url" : "/web/myweb.htm?channel=json",
      "value" : "dutch"
    }, {
      "id" : 42,
      "locale" : "en_US",
      "title" : "MyWeb",
      "url" : "/web/myweb.htm?channel=json",
      "value" : "english"
    } ]
  }, {
    "_searchRetrieve" : {
      "contentItemID" : "M6067",
      "language" : "nl_NL",
      "success" : true
    },
    "metadata" : {
      "Date" : {
        "Date" : "2021-08-27T00:00+02:00"
      },
      "contenttype" : "wmammodularcontent_date",
      "copyright" : "",
      "expiration_date" : "",
      "external_id" : "",
      "id" : 6067,
      "lastmodified_date" : "2021-08-24T17:03+02:00",
      "lead" : "",
      "publication_date" : "2021-08-24T14:35+02:00",
      "tags" : [ ],
      "title" : "Date",
      "url" : "/web/myweb/date.htm?channel=json"
    }
  } ],
  "includes" : { },
  "warnings" : [ ]
}
```


Back to top