Use the KnowledgeOwl API

Last Modified on 02/26/2026 12:08 pm EST

An introduction to using our API.

Introducing the KnowledgeOwl API

This feature is available on select plans.

The KnowledgeOwl REST API provides a large collection of endpoints, allowing you to perform many of the same tasks that can be done through the GUI. This can be helpful in a variety of scenarios, including:

- Automating repeated tasks
- Bulk operations, such as extracting custom data reports.
- Integrating with other tools. For example, you can use the API with Webhooks by Zapier to integrate KnowledgeOwl with other software.

This section of the documentation provides information on how to query the API, including authentication and filtering. It is specific to the KnowledgeOwl API, and assumes you have some familiarity with REST APIs. If you are new to working with APIs, you may want to take a look at Working with APIs first.

Refer to Endpoint reference for details of available endpoints.

Authentication

The API uses basic authentication. This means all API requests need to include a username and password. For the KnowledgeOwl API, the username must be an API key. The password can be any dummy value, such as x. Refer to API keys for information on creating, editing, and deleting your API keys.

Our API is an HTTPS-only API. Any non-secure requests will return an error.

API keys

Authors with **Full Admin permissions** can create and manage API keys, regardless of their author role.

To create and manage your API keys:

1. Go to **Account > API**.

Create a new key

To generate a new API key:

1. Go to **Account > API**. The **API Keys** page opens.
2. Select **+ Add New API Key**. The API Key Creation modal opens.
3. Enter a **Purpose for this key**. The purpose should identify any integrations, processes, and so on that would help someone know what this key is used for and whether it is still used.
4. Select the permissions you want this API key to have in the **Allowed actions for this key** section.

   Only use what you need
   Limit permissions to only those actions needed for the task. To pull information from KnowledgeOwl, only **GET** permissions are needed.

5. Once you've finished configuring your API key, select **Add** to create your key.

KnowledgeOwl adds the key to the bottom of the list. It is now ready to use.

Edit an existing API key

You may want to change the purpose listed for an API key or add or remove allowed methods.

To update an existing API key:

1. Go to **Account > API**. The **API Keys** page opens.
2. Select the gear icon in the **Actions** column next to the API key you want to edit:

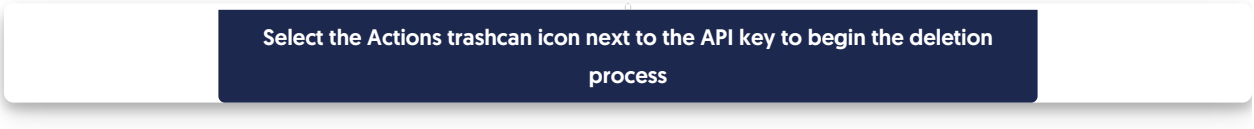
   The **Update API Key** modal opens.
3. Make any changes to the key's purpose and allowed actions.
4. Select **Update** to save your changes.

Delete a key

To delete an API key that is no longer needed:

1. Go to **Account > API**. The **API Keys** page opens.

2. Select the red trashcan icon to the right of the API key you want to delete.



   The **Delete API Key** confirmation modal opens.

3. Deleting an API key cannot be undone. If you're sure you want to delete the API key, select **OK** to confirm deletion.

4. The key is now deleted and all calls made using it will fail.

Find your knowledge base ID or project ID

Many API calls require you to pass in a knowledge base ID, known as the `project_id` in the API documentation.

To find your knowledge base ID:

1. Go to **Articles**.

2. Your knowledge base/project ID is in the URL on this page, the string of letters and numbers that appears after `/articles/id/`.
   For example, if my URL on the articles page is `https://app.knowledgeowl.com/kb/articles/id/11abc2d3e45fg678h9012345`, my project ID is `11abc2d3e45fg678h9012345`.

Pagination and limiting results

Limit results

You can limit the number of objects returned per page in your query results. To do this, add `"limit": <number>` to your query parameters. For example, to get all articles in your knowledge base, limited to 20 objects per page, add `"limit": 20` to your query parameters.

Accessing pages

If your query returns a large amount of data, or you set a limit on results, KnowledgeOwl paginates the results and returns the first page. If your data is paginated, the response will include the following:

```
"page_stats": {
 "total_records": <number>
 "total_pages": <number>
}
```

To access later pages, you have to make multiple calls, requesting each page in turn. Add `"page": <number>`, along with any other query parameters. For example, to access the fourth page of data, include `"page": 4` in your

query parameters.

Usage limits

The KnowledgeOwl API is not intended for extremely heavy use. If you anticipate making more than 1500 requests per five minutes, please contact us to discuss your requirements.

Input and output formats

KnowledgeOwl's API accepts the following input types:

- **Standard REST HTTP headers -** `?field1=value&field2=value&array1[]=value1&array1[]=value2`
- **Application/JSON -** `'{"field1": "value", "field2": "value", "array1": ["value1", "value2"]}'`

KnowledgeOwl's API can return the following output types as specified by the call endpoint:

- JSON (default) — https://app.knowledgeowl.com/api/head/{{object}}.json
- JSONP — https://app.knowledgeowl.com/api/head/{{object}}.jsonp?callback={{functionName}}
- HTML — https://app.knowledgeowl.com/api/head/{{object}}.html
- PSON — https://app.knowledgeowl.com/api/head/{{object}}.pson

API date formats

For PUT or POST calls, dates can be formatted in one of two ways:

- Unix timestamps: Also known as epoch timestamps. See Unix Time Stamp if you're unfamiliar with using this format. Example: 1701745258.
- Y-m-d H:i:s format: four-digit year, two-digit month and day, military time format for GMT or its current equivalent. Example: "2023-10-05 18:48:00".

Query operators

The KnowledgeOwl API supports a selection of query operators, allowing you to add logic to the body of your API calls. This means you can do things like search for a particular name or term, or filter your results. This section lists all the available operators, with examples of how to use them.

String and array comparisons

```
//Example article object
{
 "id": "9999",
 "name": "Article 1",
 "parents": [
  "1234",
  "4321"
 ]
}
```

$in - in array

```
//Use $in to find the example article based off of the "name" field, which is a string
//This filter matches any article that has a "name" of "Article 1", OR "Article 2"
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"name": {"$in": ["Article 1", "Article 2"]}}'
```

```
//Use $in to find the example article based off of the "parents" field, which is an array
//This filter matches any article that contains the value "1234" within the "parents" array
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"parents": {"$in": ["1234"]}}'
```

$nin - not in array

```
//Use $nin to exclude the example article based off of the "name" field, which is a string
//This filter excludes any articles that have a "name" of "Article 1" OR "Article 2"
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"name": {"$nin": ["Article 1", "Article 2"]}}'
```

```
//Use $nin to exclude the example article based off of the "parents" field, which is an array
//This filter excludes any articles that have the value "1234" within the "parents" array
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"parents": {"$nin": ["1234"]}}'
```

$ne - not equal to

```
//Use $ne to find all articles that do not have a "status" of "deleted"
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"status": {"$ne": "deleted"}}'
```

$regex - regular expression string comparison

```
//Use regex to find the example article based off of the "name" field, which is a string
//This filter matches any article that contains the string "article" in the "name" field
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"name": {"$regex": "article", "$options": "i"}}'

//The "$options" value of "i" modifies the regular expression to be case insensitive
//Available regex modifiers are "i", "m", "x", and "s"
```

Date and numeric value comparisons

Dates are returned via the API in human readable format based on the Timezone and Date Format settings in your knowledge base.
When filtering API objects based off of dates, the API expects unix timestamps to be passed in.

```
//Example article object
{
 "id": "1234",
 "index": 2,
 "date_created": "11/07/2015 11:06 am GMT", //Equal to 1446894360 unix timestamp
}
```

$gt - greater than

```
//Use $gt to include the example article based off of the "index" field, which is numeric
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"index": {"$gt": 1}}'
```

```
//Use $gt to include the example article based off of the "date_created" field, which is a timestamp
//1446807960 is equal to 11/06/2015 11:06 am GMT
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"date_created": {"$gt": "1446807960"}}'
```

$gte - greater than or equal to

```
//Use $gte to include the example article based off of the "index" field, which is numeric
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"index": {"$gte": 2}}'
```

```
//Use $gte to include the example article based off of the "date_created" field
//1446894360 is equal to 11/07/2015 11:06 am GMT
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"date_created": {"$gte": "1446894360"}}'
```

$lt - less than

```
//Use $lt to include the example article based off of the "index" field, which is numeric
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"index": {"$lt": 3}}'
```

```
//Use $lt to include the example article based off of the "date_created" field
//1446980760 is equal to 11/08/2015 11:06 am GMT
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"date_created": {"$lt": "1446980760"}}'
```

$lte - less than or equal to

```
//Use $lte to include the example article based off of the "index" field, which is numeric
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"index": {"$lte": 2}}'
```

```
//Use $lte to include the example article based off of the "date_created" field
//1446894360 is equal to 11/07/2015 11:06 am GMT
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"date_created": {"$lte": "1446894360"}}'
```

Logic filtering

```
//Example article object
{
  "id": "9999",
  "name": "Article 1",
  "index": 10,
  "status": "deleted",
}
```

$and - must match ALL specified filters

```
//Use $and to include the example article based off of multiple filters
//Article must have an "index" that is greater than 1 AND less than 20
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"$and": [{"index": {"$gt": 9}}, {"index": {"$lt": 20}}]}'
```

$or - must match ONE of the specified filters

```
//Use $or to include the example article based off of multiple filters
//Article can either have an "index" that is greater than 20 OR less than 11
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"$or": [{"index": {"$gt": 20}}, {"index": {"$lt": 11}}]}'
```

$nor - must NOT match ANY of the specified filters

```
//Use $nor to include the example article based off of multiple filters
//Article must NOT have an "index" that is greater than 11 NOR less than 9
curl -u {{API Key}}:X
  -H "Content-type: application/json"
  -X GET "https://app.knowledgeowl.com/api/head/article.json"
  -d '{"$nor": [{"index": {"$gt": 11}}, {"index": {"$lt": 9}}]}'
```

API calls in snippets

Sometimes you might want to extend KnowledgeOwl's built in functionality by utilizing client side API calls in your knowledge base.

Previously, this involved constructing an AJAX call that contains a KnowledgeOwl API key for authentication. However, by exposing your API key on the client side (even when restricted to GET access only), you are opening yourself up to unintended reader behavior.

KnowledgeOwl API Merge Code

To prevent this issue, we have added the ability to construct an API merge code within a snippet's content. On page render, the merge code will be replaced with a unique, single use URL that does not contain your API key or any account specific information.

Constructing the Merge Code

Let's take a look at how to construct this merge code and then what we can do with it. Below is the template for the merge code.

```
[ko_api(API Object|{JSON API filter})]
```

If we break the above template down into its parts, we get the following 3 required pieces.

1. **The outer wrapper: [ko_api( )]**
   This wrapper and everything within it will be replaced server side with a unique URL at the time of page rendering.

2. **The API Object**
   The first part of the inner required information denotes which API object you are going to be querying. For example, if you want query for categories, you would use category followed by the pipe symbol | .

3. **JSON API Filter**
   The second part of the inner required information needs to be a JSON formatted string containing a valid API filter. Let's say we want to query for the 5 newest categories in our knowledge base that aren't deleted. We can construct our JSON string like so:

   ```
   //project_id = Knowledge base ID
   {"project_id": "123456", "status": "active", "limit": 5, "sort": {"date_created": 1}}
   ```

When we put the parts from above together, we get the following fully constructed merge code:

```
[ko_api(category|{"project_id": "123456", "status": "active", "limit": 5, "sort": {"date_created": 1}})]
```

Knowledge Base Variables

The API merge code is replaced server side so you will not be able to use Javascript variables within it. However, we have created the following variables that you can use to reference information about the current page and the current reader that is logged in:

Variable Name	Variable Value	Example JSON
%cur_kb_id%	The ID of the knowledge base that is currently being viewed.	"project_id": "%cur_kb_id%"
%cur_cat_id%	If viewing a category: returns the ID of the category currently being viewed; If viewing an article: returns the ID of the category in which the article is contained. If the article is in a subcategory, this is the category *immediately above* this article in the hierarchy, not it's ultimate top-level parent.	"category": "%cur_cat_id%"
%cur_top_cat_id%	The ID of the top most parent category that the current article or category is in.	"category": "%cur_top_cat_id%"
%cur_parent_cat_ids%	Array of all parent category IDs that the current article or category is in.	"category": {"$in": "%cur_parent_cat_ids%"}
%cur_art_id%	The ID of the article that is currently being viewed	"id": "%cur_art_id%"
%cur_art_tags%	Array of tag IDs that are in use on the currently viewed article	"tags": {"$in": "%cur_art_tags%"}
%cur_art_permalink%	The permalink of the currently viewed article	"url_hash": "%cur_art_permalink%"
%cur_reader_id%	The ID of the currently logged in reader; will filter results by content that reader has access to; will not work for authors who are also readers	"reader_id": "%cur_reader_id%"

Variable Name	Variable Value	Example JSON
%cur_reader_groups%	Array of reader groups IDs that the currently logged in reader is assigned to	"reader_roles": "%cur_reader_groups%" *NB: The merge code itself surrounded by " is all you need - an $in comparison is already included for you in the merge code*
%cur_reader_username%	The username of the currently logged in reader	"username": "%cur_reader_username%"
%cur_search_term%	The "phrase" parameter in the URL	"phrase": "%cur_search_term%"

Using the Merge Code

Now that we have our merge code, let's look at how we can use it within our snippet content to get the information requested. Below is a script that console logs the information returned from our API call.

```
<script>
 $(function() {
  $.get('[ko_api(category|{"project_id": "123456", "status": "active", "limit": 5, "sort": {"date_created": 1}})]',
  function(apiData) {
       //do something with the returned data
       console.log(apiData);
  }).fail(function(error) {
    //uh oh something went wrong. Alert the end-user or otherwise handle the error
  });
 });
</script>
```

As you can see, the merge code is used in place of the AJAX URL, but the rest of the jQuery code remains exactly the same. When the above code is rendered to the page, the merge code is replaced with a safe, valid URL, and results in something like the following.

```
<script>
 $(function() {
  $.get('/help/ko-api/mid/9999aaaaadsfsdfsdf',
  function(apiData) {
       //do something with the returned data
       console.log(apiData);
  }).fail(function(error) {
    //uh oh something went wrong. Alert the end-user or otherwise handle the error
  });
 });
</script>
```

Now let's use some of the knowledge base variables listed above to get all of the other articles that are in the currently viewed article's category.

```
<script>
  $(function() {
    //get all published or needs review articles in the current category except for the one currently being viewed
    $.get('[ko_api(article|{"project_id": "%cur_kb_id%", "status": {"$in": ["published", "review"]}, "category": "%cur_cat_id
%", "url_hash": {"$ne": "%cur_art_permalink%"}, "sort": {"index": 1}})]',
    function(apiData) {
        //do something with the returned data
        console.log(apiData);
    }).fail(function(error) {
     //uh oh something went wrong. Alert the end-user or otherwise handle the error
    });
  });
</script>
```

Working with article status

If you're pulling a list of articles via API snippet, the odds are pretty good that you're going to be using the status field. While most of our other API endpoints have a status field that is "active" or "deleted", the publishing status on articles has two statuses that could be considered active: Published ("published" in the API) and Needs Review ("review" in the API).

If you'd like to filter your article API call to get status, instead of using "status": "active" here, you'd want to use an in operator and look for the status to be in one of those two: `"status": {"$in": ["published", "review"]}` **. You can see an example of this in action in the final code block in the section before this one.**

API calls with paged results

Sometimes your API call may have multiple pages of results. In this case, we will return the next API call URL as part of the returned data. The URL will be located in the "page_stats" array like so:

```
page_stats: {
  total_records: 203
  total_pages: 3
  next_page: 2
  next_page_url: /help/ko-api/mid/9999aaaaadsfsdfsdf
}
```

Here's a template to get you started with paged API snippet calls:

```
<script>
 $(function(){
  //first page of results API call
  var firstUrl = '[ko_api(article|{"project_id": "%cur_kb_id%","_fields": ["name"], "limit": 75})]';

  //function to get multiple pages of results from API
  var getArticles = function(curUrl) {
   $.get(curUrl, function(data) {
    console.log(data);
    $.each(data['data'], function(index, value){
     //do something with api objects
    });
    //now fetch the next page of results if there is one
    //using the URL returned from the previous API call
    if(data['page_stats']['next_page_url'])
     getArticles(data['page_stats']['next_page_url']);
   }).fail(function(error) {
//you failed!
    console.log(error);
   });
  }

  //get the first page of results;
  getArticles(firstUrl);
 });
</script>
```

Requirements for use

API merge codes can ONLY be used for GET calls. Attempts to POST, PUT, or DELETE will return an error.

You must have at least one active API key in your account with ONLY GET permission. If you do not have an available API key that meets this requirement, the merge code URL will return an error.

The JSON string containing the API filter must contain a valid knowledge base ID in the format of {"project_id": "1234"}.

DO NOT include your API key in the merge code JSON. If you include an API key in the JSON, the merge code URL will return an error.

API calls in snippets do not show in article Preview mode. You'll need to publish the article to view the results of the API snippet.