REST Interface

Agiloft supports REST-style invocations that correspond to CRUD operations: Create, Read, Update, Delete.

For CRUD operations, two invocation styles are supported:

- HTTP methods
- GET/POST methods

Invocations that map directly to HTTP methods are listed in the second column below. The fallback GET/POST methods listed in the third column are generally used for organizations that don't support all HTTP methods.

Operations for both invocation styles are detailed in the table below. REST API calls are documented further in their individual, linked pages. Standard API calls are also included in your KB's Swagger API documentation. REST API calls specifically related to outbound webhooks are documented on the Webhooks page.

Operation	Supported HTTP Methods	REST Endpoint	Returns
Create	GET/POST	/ewws/EWCreate	ID of the newly created record
Read	GET/POST	/ewws/EWRead	Encoded record information
Update	GET/POST/PUT	/ewws/EWUpdate	Encoded record information after update
Delete	GET/POST /DELETE	/ewws/EWDelete	Does not return anything
Select	GET/POST	/ewws/EWSelect	A list of record identifiers and a length of that list Supports limited SQL select functionality. Only available via GET/POST
Login	GET/POST	/ewws/EWLogin	A session token, expiration time, and authentication scheme
Logout	GET/POST	/ewws/EWLogout	Does not return anything
Search	GET/POST	/ewws/EWSearch	Supports saved search and ad hoc queries
Attach	PUT	/ewws/EWAttach	Adds attachment
RemoveAttached	GET/POST	/ewws /EWRemoveAttachment	Does not return anything
RetrieveAttached	GET/POST	/ewws/EWRetrieve	Retrieves attachment
AskQuestion	GET/POST	/ewws/EWQuestion	Returns answer to text question using Genius™ from Cognizer
Lock	GET/PUT/DELETE	/ewws/EWLock	Provides ability to check, lock, and unlock the lock status of a record

AttachInfo	GET/POST	/ewws/EWAttachInfo	Returns info about the attachments of a record

OpenAPI

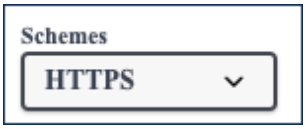
You can access detailed REST documentation with Swagger OpenAPI directly in your KB at **Setup > System > View REST documentation**. This code repository includes entries for the tables and fields specific to your system, and offers you the ability to test an API call with specified parameters once you obtain authorization. Testing API calls should always be performed in test KBs, rather than a production environment. If you have no other choice but to use the production environment, use extreme caution.

You can download a comprehensive Open API JSON file from the right-hand side of the page by clicking **Download Open API JSON**. This file is useful for importing into Postman.

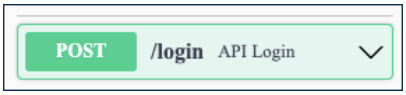
Authorization

You need to go through an authorization process before you can test API calls. Follow the steps below to complete this process:

1. In your test KB, navigate to **Setup > System > View REST documentation**.
2. Under Schemes, select HTTPS.

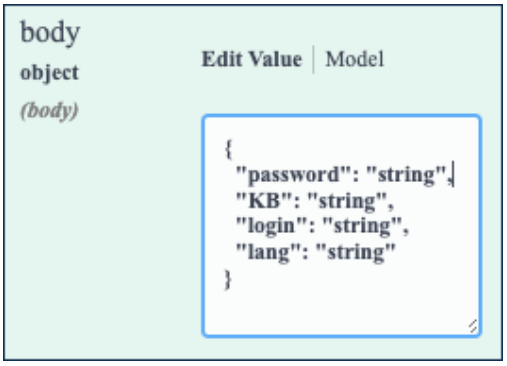


3. Open the General section and click POST /login.

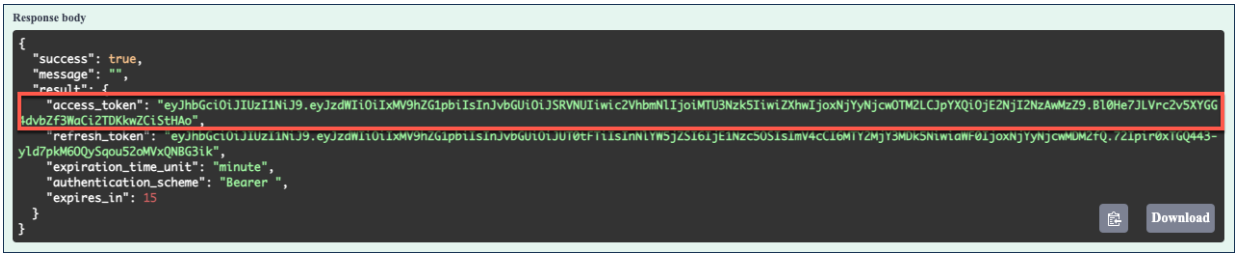


4. Click Try It Out.
5. Replace the "string" values in the parameters.
    a. For password and login, enter the user that you are going to use for your REST calls, which is commonly just a user created specifically for this purpose. This user must be in a REST-enabled Group.
    b. For KB, enter the name of the KB as it appears in the top right-hand corner of your KB next to the Help icon.

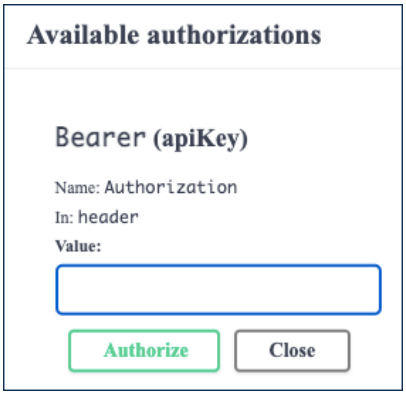
c. For language, enter the language code your KB uses. Language codes are generally two lowercase letters. For example, English would be "en".



6. Click Execute.
7. Scroll down slightly to the Server response section and copy the access token. Do not include the quotation marks.



8. Scroll back up to the top of the page and click Authorize.
9. Paste the access token into the Value field in the window that appears.



10. Click Authorize to complete the authorization process.

Now, you can begin to test API calls in the KB you just authorized. You only need to authorize once, but will need to log in again after the default 15 minute expiration timer.

Testing an API Call

Now that you are authorized, you can test commands in the KB using various methods of POST, GET, DELETE, and PUT.

Below are example steps for testing a call. This examples uses PUT to update a record in the Person table.

1. Click the table you'd like to test calls against.
2. Click PUT /contacts/{id}.



3. Click Try it Out.
4. In the body field, you'll see a list of the parameters for a Person record. Find the fields you'd like to update, and replace the "string" value with the new data. ID is a required field because it indicates which record will receive the update. You can delete any fields you do not want to update with this call.
5. Click Execute.

URL Conventions

The following conventions apply to how URLs are constructed for different operations.

KB names and table names are case sensitive. To find the correct styling for your table, go to **Setup > Tables**, select your table, click Edit, and look for the Logical Table Name. Use the same text and format of the Logical Table Name when referencing that particular table.

REST

Use the code block below for REST. However, you should omit `/{id}` when using Create (POST); creating a new record does not require a reference to an existing one.

```
/ewws/REST/{kbName}/{table}[/{id}]?$login={login}&password={password}&lang={lang}
&...
```

GET/POST

For the fallback GET/POST interface, use the following:

```
/ewws/{operation}?$KB={kbName}&$table={table}&$login={login}&password={password}
&lang={lang}&...
```

The parameters of the POST request can be inserted into the body of the request to conceal the user credentials.

Name/value pairs

The URL string should contain the parameter name/value pairs, as per operation specification:

Example

```
GET {server name}/ewws/REST/Demo/Company
/123?$login=user&$password=123&$lang=en
```

Or

```
POST {server name}/ewws
/EWRead?$KB=Demo&$table=Company&id=123&$login=user&$password=123&$lang=en
```

Return Values

Return values come in an encoded form suitable for applying the JavaScript `eval()` operations. Extended characters (like ö) are returned in a UTF-encoded format.

Return values can be accessed from local variables. Fields with empty values are not returned.

To avoid interfering with variables that may already exist in the client script or document, all table column names in the variables that result from the `eval()` call are prefixed with `EWREST_`. As such, what is returned is escaped using JavaScript rules. The content type of the field is irrelevant for the escaping.

```
EWREST_company_name='Agiloft';
EWREST__1794_full_name=' agiloft.com Admin';
EWREST_website_url=' https://www.agiloft.com'
EWREST_date_updated=' 21 8 06 15:18:43 PM';
EWREST_id=' 21';
```

You may want to consider using a JSON decorator to receive a JSON formatted stream instead, since JSON has more readily available parsers. Here is its syntax in a REST call:

```
https://<hostname>/ewws/EWRead/.
json?$KB=KB&$table=<table>&$login=admin&$password=<pwd>$lang=en&id=<id>
```

In this case the return result would look like:

```
{"success":true,"message":"","result":{...,"company_name":"Agiloft","
_1794_full_name":"Agiloft System","id":21}}
```

Delays

Each call via the REST interface has a delay inserted after the operation has completed.

The delay is set to one second by default and is configurable via the global variable Web Services Delay (WSDelay).

Delays are important because:

- An operation on a record may invoke rules and other functions, which need to be allowed enough time and resources to complete.
- Client applications could mistakenly overuse web services, resulting in a flood of requests.

Security

Use the Login and Logout operations to secure sessions initiated by REST.

- **Login**: Use this operation to authenticate KB credentials and return a token, which can then be used by following requests to avoid including login credentials in request URLs. Input the KB login and password as parameters, and the operation returns a token, expiration time, and authentication scheme to the client. The token can then be used in an Authorization request header, prefixed by the authentication scheme, instead of including the login and password parameters in following requests. By default, the Bearer authentication scheme is used, and the token expiration time is 15 minutes. You can adjust the expiration time by creating a token_expires_in global variable and setting the number of minutes, up to 60.
- **Logout**: Use this operation to terminate a session created by the Login operation. This terminates the session associated with the token passed in the Authorization header.

Examples

These examples show the process of creating the token, placing it in the request header, and then terminating the token session. With the token in the header, you can use functions like Search without passing in a login and password in the URL.

Usage	Headers	Example Text	Response
Login		POST https://server/ewws/EWLogin?	HTTP/1.1 200 C

request		$login=user&$password=passwd& $KB=Demo&$lang=en	{"**access_toker** eyJhbGciOiJIUz refresh _token":"…"," expiration_time_ minute","expires _in":5," **authentication_** Bearer "}
Functional request	**Authorization**: Bearer eyJhbGciOiJIUzI1NiJ9.…	GET https://server/ewws/EWSearch? $KB=Demo&$table=body&$lang=en& field=id&field=text&field=body&query=...	
Refresh token	**Authorization**: Bearer eyJhbGciOiJIUzI1NiJ9.…	POST https://server/ewws /EWLogin?$KB=Demo&$lang=en&refresh_token=xYZsk ...	HTTP/1.1 200 C {"**access_toker** jkShbGciOiJIUz refresh _token":"…"," expiration_time_ minute","expires _in":5," **authentication_** Bearer "}
Logout request	**Authorization**: Bearer eyJhbGciOiJIUzI1NiJ9.…	GET https://server/ewws/EWLogout	HTTP/1.1 200 C

Data Encoding

The following conventions are in place to encode aspects of a typical Agiloft knowledgebase:

Simple fields

Simple fields can be filled directly by setting the value for them.

```
... &first_name=John&last_name=Doe&...
```

Choice fields

Choice fields are encoded directly with their text values as seen in the GUI.

```
...&country=USA&...
```

⚠ For ad hoc queries in the Select call, choice values should be addressed via the ID values obtained from GetChoiceLineId.

Multi-choice fields

Multi-choice fields are encoded as multiple key/value pairs.

```
... &contactMethod=phone&contactMethod=email&...
```

Date, date-time and time fields

Date, date-time and time fields can be encoded with any of 3,275 formats currently supported. The system evaluates the possible formats sequentially and stops when parsing if one of the formats succeeds.

Please refer to the following document to see the list of supported date-time formats: datetime.txt

Elapsed time fields

These can be encoded as "days:hours:minutes:seconds" e.g "0:1:35:15"6

Linked field relationships

If the linked field allows independent values, they can be assigned to the columns in the main table:

```
...&company_name=Agiloft&...
```

To create a link based on the values of imported columns, you can use Query By Example, expressed with a colon ':' qualifier.

Example values have to be provided for one or more of the imported columns in either of the following ways:

- ```
  ...&company_name=:Agiloft&...
  ```

  Or

- ```
  ...&company_name=Company:Agiloft&...
  ```

The name of the original table that contained the field is required if the link type is "single field from multiple tables", but may be omitted in the case of a single donor table.

If the value contains the colon ':' symbol or the question mark '?' symbols, they have to be escaped with backward-slash "\" in the following way:

```
...&company_url=Company:http\://www.example.com&...
```

For more complex queries, possibly including sub-selects, aggregative functions and columns from the donor table that are not imported into the target, you can use a SQL query, expressed with a question mark '?' qualifier. The "where" clause follows the qualifier for the query that will be run against the donor table. The columns in the query have to be referred with their database names rather then logical ones.

Example

To look for Employee records where the Company currency is EUR, and not in the linked set, use the following search:

```
https://localhost:8080/ewws
/EWSearch?$login=admin&$password=qwerty&$lang=en&$table=contacts.
employees&$KB=Demo&query=_1576_company_name0=Company?currency~='EUR'
```

Multiple values for the linked field

These are encoded as multiple key/value pairs:

```
...&company_name=Company:Agiloft& company_name=Company:SaaSWizard&...
```

File and image fields

An example of this field is an Attached File field.

The REST interface accepts files in POST requests when used with **enctype="multipart/form-data"** encoding.

The name of the form field should match the name of the file or image field. Additionally a field *fieldName$overwrite* can be specified with any value to instruct the REST interface to override the current data in the file or image field, rather than add.

Application clients can use REST - Attach operation instead with PUT HTTP method.

Decorators

REST calls allow use of three decorators:

- **Asynchronous decorator:** These can be applied to EWCreate, EWUpdate and EWDelete calls to do "fire-and-forget" type of call. They should used if the results normally returned by an operation are not important to the caller e.g. a scenario when a lot of records have to be created in the backend.

  Use

  ```
  /ewws/async/EWCreate?...
  or
  /ewws/EWCreate/.async?...
  ```

- **Redirect decorator**: These can be applied to all calls to have a HTTP redirect issued depending on the success of the operation and are useful when integrating with web sites. Please note the page to which redirect is performed will NOT receive any return data. These calls require two parameters to be specified: *$exiturl* and *$errorurl* - for redirecting in case of successful operation and in case of error respectively. Both parameters should be absolute URLs and URL encoded if necessary.

  Use

  ```
  /ewws/redirect/EWCreate?...&$exiturl=http%3A%2F%2Fwww.google.
  com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
  or
  /ewws/EWCreate/.redirect?...&$exiturl=http%3A%2F%2Fwww.google.
  com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
  ```

- **JSON decorator**: produces a JSON formatted stream. If you apply JSON formatting, you can optionally include the parameter err_code_resp=1 to receive specific response codes, rather than the default behavior of receiving code 200 in response.

  Use

  ```
  /ewws/EWRead/.json?...
  ```

Please note that decorators can be chained. In the case of chaining they are applied from left to right.

Use

```
/ewws/redirect/EWCreate/.async?...&$exiturl=http%3A%2F%2Fwww.google.
com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
/ewws/async/EWCreate/.redirect?...&$exiturl=http%3A%2F%2Fwww.google.
com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
/ewws/redirect/async/EWCreate?...&$exiturl=http%3A%2F%2Fwww.google.
com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
```

```
/ewws/async/redirect/EWCreate?...&$exiturl=http%3A%2F%2Fwww.google.
com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
```