Zendesk® REST API v1 (deprecated)

Zendesk Inc.

Notice

Copyright and trademark notice

Zendesk® REST API (deprecated)

Zendesk - 989 Market St, Ste 300 - San Francisco - CA 94103 - USA

www.zendesk.com

Contents

Chapter 1

Zendesk® REST API v1 (deprecated)

This document covers the first version of the Zendesk® REST API, which is deprecated as of November 2012. Visit our *Developers site* for our API v2, which includes our current *REST API*.

Introduction

The Zendesk REST API allows developers to hook into Zendesk and connect it to third-party applications. Whether you're writing a plugin for an application or planning on hooking some internal application into Zendesk, the API can do it for you.

The REST API is implemented as plain XML or JSON over HTTP using all four REST commands – GET, POST, PUT and DELETE. Every resource, like Ticket, User or Tag, has their own URL and are manipulated in isolation. The API closely follows the REST principles and it's easy to use.

You can explore the GET part of the API through any browser. We recommend Firefox. Most URLs in Zendesk can be viewed in XML form by appending the URL with .xml such that /users/4 becomes /users/4.xml if you want to see the XML version of the ticket.

API Throttle

To ensure continuous quality of service, API usage can be subject to throttling. The throttle will be applied once an API consumer reaches a certain threshold in terms of a maximum of requests per minute. Most clients will never hit this threshold, but those that do, will get met by a HTTP 503 response code and a text body of “Number of allowed API requests per minute exceeded.”

We encourage all API developers to anticipate this error, and take appropriate measures like e.g. using a cached value from a previous call, or passing on a message to the end user that gets subjected to this behaviour (if any).

API Changes

Zendesk may introduce changes to the API without notice if the changes do not break existing functionality. If we make changes that will impact an existing API, such as removing an attribute, we will announce these changes in our *Announcements* forum. You may want to subscribe to that forum to monitor any API changes that might affect you.

Authentication

Use of the API is always through an existing user in Zendesk. There's no special API user. You get to see and work with what the user you are logging in to the API is allowed to. You're required to add user credentials via HTTP Basic Authentication. Security is provided via SSL.

Authentication and Acting on Behalf of Another User

The API supports a header that allows you to execute API calls on behalf of another user. This is in the same fashion, that Zendesk allows you to "assume" another user when logged into the backend. This can be done using the X-On-Behalf-Of header when issuing the API calls. If, for example, you would like to create a request as "Joe Enduser" but do not know his password, you can do this by setting the header like so:

The X-On-Behalf-Of header

```
X-On-Behalf-Of: joe.enduser@theendusers.com
```

You would then do the actual authentication as an agent user, and set the above header to make the actions on behalf of the end user.

REST API Reading

We recommend using *curl* for testing out the various features that the REST API offers. A nice online option to curl is *hurl*.

The REST API has two modes of actions for reading – show and list. Show returns a single record and list returns a collection. Usually, there's just a single show action for each resource, but several lists. You can easily explore REST API reading through a browser, as all these actions are done through GET.

Examples

GET a list of users – XML

```
curl -u agentemail:password http://helpdesk.zendesk.com/users.xml
```

This command issues a HTTP GET request to /users.xml (replace "helpdesk" in the URL with your help desk subdomain). It will return HTTP status code 200, and a XML document listing the users in your help desk – end users, agents and administrators. If nothing is found, a HTTP 404 "not found" response will be returned. This is equivalent to typing in the URL "http://helpdesk.zendesk.com/users.xml" in your browser, when logged on as the authenticated user.

GET a list of users – JSON

```
curl -u agentemail:password http://helpdesk.zendesk.com/users.json
```

This is equivalent to the previous example, but returns the result as JSON instead of XML. Furthermore, you can specify a callback by appending "?callback=your_call_back_function" to the GET URL. See the Widgets section for JSON examples.

Get details for a specific user

```
curl -u agentemail:password \
  http://helpdesk.zendesk.com/users/304.xml
```

If a user with ID 304 exists, an XML response is generated along with the status code “200”. The XML response contains the user data registered for the particular user. If nothing is found, the response HTTP 404 “not found” is returned.

Get a list of requests for a user using X-On-Behalf-Of

```
curl -u agentemail:password -H "X-On-Behalf-Of: enduseremail" \
  http://helpdesk.zendesk.com/requests.xml
```

Uses the agent credentials for authentication and executes the action as if done by the end user.

API writing

When you’re creating and updating REST resources, you’ll be sending XML into Zendesk. Remember to add the header “Content-type: application/xml”, such that Zendesk knows that it’s not regular HTML form-encoded data coming in. Then just include the XML of the resource in the body of your request, and you’re done.

A succesful creation responds with the status code “201”. The URL of the new resource is located in the Location header (letting you know where to update your new resource in the future). Also included is the complete XML for the resource in the response. This is handy, as you can usually create a new resource with less than all its regular attributes, including attributes such as created-at.

Updating resources is done through the PUT command and against the URL of the resource you want to update. The response to a successful update is “200”.

Examples

POST (create) example – create new ticket

```
curl -u agentemail:password -H "Content-Type: application/xml" \
  -d "<ticket><description>Test</description><requester_id>54</
requester_id></ticket>" \
  -X POST http://helpdesk.zendesk.com/tickets.xml
```

This creates a new ticket, requested by user ID 54 and with the authenticated user as submitter.

It may be easier to have the XML fragment in a file rather than type it into the command line, you can do this using curl also:

POST (create) example – create new ticket from file

```
curl -u agentemail:password -H "Content-Type: application/xml" \
  -d @sample.xml -X POST http://helpdesk.zendesk.com/tickets.xml
```

Where the contents of sample.xml is:

```
<ticket> <description>Test</description> <requester_id>54</requester_id>
 </ticket>
```

PUT (update) example – change ticket priority

```
curl -u agentemail:password -H "Content-Type: application/xml" \
  -d "<ticket><priority_id>4</priority_id></ticket>" \
  -X PUT http://helpdesk.zendesk.com/tickets/5.xml
```

Sets the priority of the ticket to 4 (Urgent).

PUT (update) example – add comment to ticket

```
curl -u agentemail:password -H "Content-Type: application/xml" \
  -d "<comment><value>Some comment</value></comment>" \
  -X PUT http://helpdesk.zendesk.com/tickets/5.xml
```

Adds a comment to ticket ID 5, submitted by the authenticated user.

Deleting through the REST API

Finally, you can delete resources (if you're allowed to) using the DELETE command.

DELETE example – removing an entry

```
curl -u agentemail:password \
  -X DELETE http://helpdesk.zendesk.com/entries/5.xml
```

This example deletes the forum topic entry with ID 5

Organizations

An end-user can be a member of an organization.

Show

```
GET /organizations/#{id}.xml
```

Returns a single organization.

Response

```
Status: 200

<organization>
  <id>2</id>
  <name>Support</name>
  <is-shared>true</name>
  <default>ajax-corp.com</default>
</organization>
```

List All

```
GET /organizations.xml
```

Response

```
Status: 200

<organizations>
  <organization>
    ...
  </organization>
  <organization>
    ...
  </organization>
</organizations>
```

Create

`POST /organizations.xml`

Creates a new organization.

Request

```
<organization>
  <name>Ajax Corp</name>
</organization>
```

Response

```
Status: 201
Location: {new-organization-id}.xml
```

Adds a organization with one agent as member.

Update

`PUT /organizations/#{id}.xml`

Updates an existing organization with new details from the submitted XML.

Request

```
<organization>
  <name>Ajax Mega Corp</name>
</organization>
```

Response

```
Status: 200
```

Destroy

`DELETE /organizations/#{id}.xml`

Destroys the organization at the referenced URL.

Response

```
Status: 200
```

Members of the organization

`GET /organizations/#{id}/users.xml`

Get a list of the members of an organization. This endpoint supports pagination of the users. For more information about the format of the users list, see the *Users API*.

Groups

An agent is a member of one or more groups.

Show

```
GET /groups/#{id}.xml
```

Returns a single group.

Response

```
Status: 200

<group>
  <id>2</id>
  <is-active>true</is-active>
  <name>Support</name>
</group>
```

List all

```
GET /groups.xml
```

Return a list of all groups, with group members nested within each group.

Response

```
Status: 200

<groups type="array" count="19">
  <group>
    <created-at type="datetime" nil="true"></created-at>
    <id type="integer">1</id>
    <is-active type="boolean">true</is-active>
    <name>1. Level Support</name>
    <updated-at type="datetime">2009-05-13T00:07:08-07:00</updated-at>
  </group>
  <group>
    ...
  </group>
  ...
</groups>
```

Create

```
POST /groups.xml
```

Creates a new group.

Request

```
<group>
  <name>Hardware</name>
  <agents type='array'>
    <agent>43</agent>
  </agents>
</group>
```

Response

```
Status: 201
Location: {new-group-id}.xml
```

Adds a group with one agent as member.

Update

```
PUT /groups/#{id}.xml
```

Updates an existing group with new details from the submitted XML.

Request

```
<group>
  <name>Hardware and ICP</name>
  <agents type='array'>
    <agent>43</agent>
    <agent>67</agent>
  </agents>
</group>
```

Note that if you do not submit an agents array, no agents get unassigned from the group. If you submit an empty array, all agents get removed from the group:

```
<group>
  <name>The now empty group</name>
  <agents type='array'></agents>
</group>
```

Response

```
Status: 200
```

Updates the group and sets two users to be a member of this group. Omit the `<agents>` part if you do not wish to change the group's membership status.

Destroy

```
DELETE /groups/#{id}.xml
```

Destroys the group at the referenced URL.

The group is not actually deleted from the system, but is set inactive and can no longer be assigned.

Response

```
Status: 200
```

Members of a group

```
GET /groups/#{id}/users.xml
```

Get a list of the members of a group. This endpoint supports pagination of the users. For more information about the format of the users list see the *Users API*.

Tickets

A Zendesk ticket has several properties with values that are fixed.

Ticket status

Status	Status ID
New	0
Open	1
Pending	2
Solved	3
Closed	4

Ticket type

Type	Type ID
No type set	0
Question	1
Incident	2
Problem	3
Task	4

Ticket priorities

Priority	Priority ID
No priority set	0
Low	1
Normal	2
High	3
Urgent	4

Ticket via types

Via types identifies the ticket submission type. Tickets submitted via the REST API have via type "Web service" per default.

Via type	Via type ID
Web form	0
Mail	4
Web service (API)	5
Get Satisfaction	16
Dropbox	17
Ticket merge	19
Recovered from suspended tickets	21
Twitter favorite	23
Forum topic	24
Twitter direct message	26

Via type	Via type ID
Closed ticket	27
Chat	29
Twitter public mention	30
Voicemail	33

Show

```
GET /tickets/#{id}.xml
```

Returns a single ticket.

Response

```
Status: 200

<ticket>
  <assigned-at>2007-06-03T22:15:44Z</assigned-at>
  <assignee-id>4</assignee-id>
  <assignee-updated-at/>
  <created-at>2007-06-03T20:15:44Z</created-at>
  <subject></subject>
  <description>My printer is not working</description>
  <external-id/>
  <group-id>2</group-id>
  <id>303</id>
  <linked-id/>
  <priority-id>3</priority-id>
  <submitter-id>3</submitter-id>
  <status-id>1</status-id>
  <status-updated-at>2007-06-03T22:15:44Z</status-updated-at>
  <requester-id>3</requester-id>
  <requester-updated-at>2007-06-03T22:15:44Z</requester-updated-at>
  <ticket-type-id>2</ticket-type-id>
  <updated-at>2007-06-03T20:15:45Z</updated-at>
  <via-id>0</via-id>
  <current-tags>printer hp</current-tags>
  <score>28</score>
  <comments type="array">
    <comment>
      <author-id>3</author-id>
      <created-at>2007-06-03T20:15:45Z</created-at>
      <is-public>true</is-public>
      <value>This is a comment</value>
      <via-id>0</via-id>
      <via-reference-id/>
    </comment>
    <comment>
      <author-id>5</author-id>
      <created-at>2007-06-04T10:07:02Z</created-at>
      <is-public>true</is-public>
      <value>Make sure it is plugged in</value>
      <via-id>0</via-id>
      <via-reference-id/>
    </comment>
  </comments>
  <ticket-field-entries type="array">
```

```
    <ticket-field-entry>
      <ticket-field-id>139</ticket-field-id>
      <value>Please contact me by phone during work hours</value>
    </ticket-field-entry>
    <ticket-field-entry>
      <ticket-field-id>141</ticket-field-id>
      <value>true</value>
    </ticket-field-entry>
  </ticket-field-entries>
   <permissions>
      <can-update-ticket>true</can-update-ticket>
      <can-delete-ticket>true</can-delete-ticket>
      <can-make-public-comments>true</can-make-public-comments>
      <can-merge-ticket>true</can-merge-ticket>
      <can-edit-ticket-tags>true</can-edit-ticket-tags>
   </permissions>
</ticket>
```

The first comment for a ticket is always equivalent to the ticket description.

If you have any custom fields in your Zendesk, they will show up in the <ticket-field-entries> part of the document.

A ticket's permissions reflect the permissions of the agent assigned to the ticket. If a different agent is assigned, the permissions may change.

List

```
GET /rules/#{view-id}.xml
```

Listing of tickets are handled through views. A view defines conditions for tickets in a collection. An example of a view is "Unassigned tickets".

The list is paginated using offsets. If 15 elements are returned (the default page limit), use ?page=2 to check for the next 15 and so on. If the view is set up to show 15 tickets on the web portal, then doing a GET on that view will return 15 records on each page; if the view is set up to show 30 tickets on the web portal, then doing a GET on that view will return 30 records on each page.

Response

```
Status: 200

<tickets count="156">
  <ticket>
    ...
  </ticket>
  <ticket>
    ...
  </ticket>
</tickets>
```

Create

```
POST /tickets.xml
```

Creates a new ticket.

Note: You can't have an assignee and no group on a ticket – an Agent needs to belong to a Group, to be assignable. If you don't pass a group to a ticket, but only the agent via the API, Zendesk will

simply pick the first group the agent is a member of. Also, if you only have one group in the system, we automatically assign to that group.

Request

```
<ticket>
  <description>My printer is not working</description>
  <requester-id>3</requester-id>
  <priority-id>4</priority-id>
  <set-tags>pr facility</set-tags>
  <ticket-field-entries type="array">
    <ticket-field-entry>
      <ticket-field-id>139</ticket-field-id>
      <value>Please contact me by phone during work hours</value>
    </ticket-field-entry>
    <ticket-field-entry>
      <ticket-field-id>141</ticket-field-id>
      <value>true</value>
    </ticket-field-entry>
  </ticket-field-entries>
</ticket>
```

Custom fields are updated through a custom field ID. Submitter of the entry is set to the authenticated user.

Response

```
Status: 201
Location: {new-ticket-id}.xml
```

Create a new ticket with requester

```
POST /tickets.xml
```

Creates a new ticket AND creates a new user as the tickets requester, IF the user does not already exist (based on the requester email). If the requester exists, no user is created and the ticket is created with the existing user as requester.

Request

```
<ticket>
  <description>My printer is not always working</description>
  <priority-id>4</priority-id>
  <requester-name>Mike Newson</requester-name>
  <requester-email>mike@nowhere.com</requester-email>
</ticket>
```

Submitter of the ticket is set to the authenticated user.

Response

```
Status: 201
Location: {new-ticket-id}.xml
```

You can omit requester-name, if you want to set an existing user as requester through an email address.

Create a new ticket from a Twitter status

```
POST /tickets.xml
```

Creates a new ticket where the subject and description are the contents of the Twitter status message.

Requires the Twitter status message ID and a Twitter account which has been configured with the Zendesk account. The list of configured Twitter accounts is available under /account/twitter_accounts.xml.

List configured Twitter accounts

```
<monitored-twitter-handles type="array" count="2">
  <monitored-twitter-handle>
    <id type="integer">2</id>
    <twitter-user-id type="integer">151356708</twitter-user-id>
    <screen-name>@zendesk</screen-name>
  </monitored-twitter-handle>
  <monitored-twitter-handle>
    <id type="integer">3</id>
    <twitter-user-id type="integer">429822864</twitter-user-id>
    <screen-name>@zendeskops</screen-name>
  </monitored-twitter-handle>
</monitored-twitter-handles>
```

If a user profile already exists in Zendesk with the same Twitter account as the Twitter status author, the new ticket is associated with this user profile. A new user profile will be created if no user exists with the same Twitter account.

Request

```
<ticket>
  <twitter-status-message-id>8605426295771136</twitter-status-message-id>
  <monitored-twitter-handle-id>4</monitored-twitter-handle-id>
</ticket>
```

Submitter of the ticket is set to the authenticated user.

Response

```
Status: 201
Location: {new-ticket-id}.xml
```

The monitored-twitter-handle-id must be the id of a Twitter account which has been added to your Zendesk account. This Twitter account is used to respond to the requester from Zendesk.

Update

```
PUT /tickets/#{id}.xml
```

Updates an existing ticket with new details from the submitted XML. In this example, the assignee, tags and custom field with ID 514 are all updated.

Request

```
<ticket>
  <assignee-id>5</assignee-id>
  <additional-tags>presales</additional-tags>
  <ticket-field-entries type="array">
    <ticket-field-entry>
      <ticket-field-id>139</ticket-field-id>
      <value>Please contact me by phone during work hours</value>
    </ticket-field-entry>
    <ticket-field-entry>
      <ticket-field-id>141</ticket-field-id>
```

```
      <value>true</value>
    </ticket-field-entry>
  </ticket-field-entries>
</ticket>
```

You can remove tags with remove-tags and set tags with set-tags.

Note: The ID of a custom field is displayed on the page where you edit the custom field. Look for it in right hand sidebar.

Response

```
Status: 200
```

Add comment

`PUT /tickets/#{id}.xml`

Adds a comment to an existing ticket.

Request

```
<comment>
  <is-public>false</is-public>
  <value>Joe, you know the drill</value>
</comment>
```

Response

```
Status: 200
```

Comment is added to ticket. You can combine this action with a ticket update.

Destroy

`DELETE /tickets/#{id}.xml`

Destroys the ticket with the referenced ID.

Response

```
Status: 200
```

REST API: Tickets for end users

For authentication on behalf of an end user, as an agent, please refer to the section on *REST authentication*.

If you want to check tickets as an end user, or update one, this is perfectly possible. End users do not see as much information as agents, so the API is similar in approach to the above examples, with a few differences elaborated here. Most notably, you must use **requests** rather than **tickets** in the URL, but the result will still be XML formatted in the same way as tickets. The following sections give concrete examples.

Show

`GET /requests/#{id}.xml`

Returns a single request.

Response

```
Status: 200

<ticket>
  <created-at>2008-07-07T14:16:16Z</created-at>
  <description>All is broken!</description>
  <nice-id>575</nice-id>
  <organization-id nil="true"></organization-id>
  <via-id>0</via-id>
  <comments type="array">
    <comment>
      <author-id>328</author-id>
      <created-at>2008-07-07T14:16:16Z</created-at>
      <value>All is broken!</value>
    </comment>
  </comments>
</ticket>
```

The first comment for a request is always equivalent to the request description.

List

```
GET /requests.xml
```

This lists all the open requests for the end user. The list is paginated using offsets. If 15 elements are returned (the page limit), use ?page=2 to check for the next 15 and so on.

Response

```
Status: 200

<tickets>
  <ticket>
    ...
  </ticket>
  <ticket>
    ...
  </ticket>
</tickets>
```

Create

```
POST /requests.xml
```

Creates a new ticket. Be aware that the attribute you set when creating a ticket as an end user differs from what you're used to when submitting tickets.

POST (create) example

Create new request as end user:

```
curl -u username:password -H "Content-Type: application/xml" \
-d "<ticket><subject>Hello</subject><description>My message</
description></ticket>" \
-X POST http://helpdesk.zendesk.com/requests.xml
```

Thus the request structure is as follows:

Request

```
<ticket>
  <subject>Hello</subject>
  <description>My message</description>
</ticket>
```

Submitter and requester of the entry is set to the authenticated end user.

Response

```
Status: 201
Location: {new-ticket-id}.xml
```

Update

```
PUT /requests/#{id}.xml
```

Updates an existing request with another comment from the submitted XML.

Request

```
<comment>
  <value>All is good again</value>
</comment>
```

Response

```
Status: 200
```

Destroy

```
DELETE /requests/#{id}.xml
```

Destroys the request with the referenced ID. Only admins can destroy tickets.

Response

```
Status: 200
```

Attachments

You can upload and associate attachments to comments via the REST API. Please note, that in order to do so, you submit the attachments first, and then subsequently link them to the comment when creating that.

Create attachment

```
curl -u username:password -H "Content-Type: application/binary" \
 --data-binary @somefile.txt \
 -X POST http://helpdesk.zendesk.com/uploads.xml?filename=somefile.txt
```

You will get a response containing a token, which is used to identify this file later on. Example:

```
<uploads token="abc123">
  <attachments>
    <attachment>789</attachment>
 </attachments>
</uploads>
```

The number (789 in this case) is the id of the attachment. You can add another attachment to the same upload "batch" by supplying the just returned token:

```
curl -u username:password -H "Content-Type: application/binary" \
 --data-binary @otherfile.txt \
 -X POST http://helpdesk.zendesk.com/uploads.xml?
filename=otherfile.txt&token=abc123
```

Which results in:

```
<uploads token="abc123">
  <attachments>
    <attachment>789</attachment>
    <attachment>790</attachment>
 </attachments>
</uploads>
```

Once you've uploaded the files, you can create a ticket with these attachments like so:

```
curl -u username:password -H "Content-Type: application/xml" \
-d "<ticket><description>Hi</description><uploads>abc123</uploads></
ticket>" \
-X POST http://helpdesk.zendesk.com/tickets.xml
```

Or, create a comment for an existing ticket (in this case ticket 123):

```
curl -u username:password -H "Content-Type: application/xml" \
-d "<comment><value>Some comment</value><uploads>abc123</uploads></
comment>" \
-X PUT  http://helpdesk.zendesk.com/tickets/123.xml
```

Users

Users in your help desk are assigned one of the following roles:

Role	value
End user	0
Administrator	2
Agent	4

In the Enterprise version of Zendesk, agent users can also be assigned to an Enterprise agent role. See Enterprise agent roles below.

Administrators always have Agent privileges, in addition to Administrator privileges.

Additionally, you can define ticket access restrictions for end-users and agents:

Restricted to	Value	Applicable for end-users	Applicable for agents
All tickets	0		X
Tickets in member groups	1		X
Tickets in member organization	2	X	X
Assigned tickets	3		X

Restricted to	Value	Applicable for end-users	Applicable for agents
Tickets requested by user	4	X	

Show

```
GET /users/#{id}.xml
```

```
GET /users/current.xml
```

Returns a single user. Using /users/current in lieu of a user ID returns the currently authenticated user's XML.

Response

```
Status: 200

<user>
  <created-at>2007-05-15T18:07:57Z</created-at>
  <email>aljohson@yourcompany.dk</email>
  <id>88</id>
  <is-active>true</is-active>
  <is-verified>true</is-verified>
  <name>Al Johnson</name>
  <roles>2</roles>
  <restriction-id>1</restriction-id>
  <time-format>0</time-format>
  <locale-id type="integer">1</locale-id>
  <time-zone>(GMT +01:00) Brussels, Copenhagen, Madrid, Paris</time-zone>
  <updated-at>2007-07-06T12:46:12Z</updated-at>
  <organization-id>2143</organization-id>
  <current-tags>tag_a tab_b tag_c</current-tags>
  <groups>
    <group>
      <id>13</id>
      <is-active>true</is-active>
      <name>Support</name>
    </group>
    <group>
      <id>3</id>
      <is-active>true</is-active>
      <name>Sales</name>
    </group>
  </groups>
</user>
```

This user is an agent, and is a member of 2 *groups* with access to tickets in member groups only.

List all

```
GET /users.xml
```

```
GET /organizations/#{organization_id}/users.xml
```

```
GET /groups/#{group_id}/users.xml
```

Returns all users in your help desk. The list is paginated using offsets. If 100 elements are returned (the page limit), use ?page=2 to check for the next 100 and so on. You can narrow down the list to

only show the users in an organization by requesting /organizations/#{organization_id}/users.xml. Likewise, you can get the users in a group by requesting /groups/#{group_id}/users.xml.

Response

```
Status: 200

<users>
  <user>
  ...
  </user>
  <user>
  ...
  </user>
</users>
```

List by search term

```
Get /users.xml?query=John&role=2
```

Returns a collection of users that has a name matching the query passed in through the URL. The list is paginated using offsets. If 100 elements are returned (the page limit), use ?page=2 to check for the next 100 and so on.

Response

```
Status: 200

<users>
  <user>
    ...
    <name>John Smith</name>
    <role>2</role>
  </user>
  <user>
    ...
    <name>Alexander Johnson</name>
    <role>2</role>
  </user>
</users>
```

Create

```
POST /users.xml
```

Creates a new user. The XML for the new user is returned on a successful request with the timestamps recorded and user id. If the account doesn't allow for more users to be created, a "507 Insufficient Storage" response will be returned.

Request

```
<user>
  <email>aljohson@yourcompany.dk</email>
  <name>Al Johnson</name>
  <remote-photo-url>http://www.example.com/directory/image.png</remote-
photo-url>
  <roles>4</roles>
  <restriction-id>1</restriction-id>
  <current-tags>tag_a tab_b tag_c</current-tags>
  <groups type='array'>
```

```
    <group>2</group>
    <group>3</group>
  </groups>
</user>
```

Note that, if a remote photo url is provided, it must be a direct URL which is not behind authentication. The API does not follow redirects.

Response

```
Status: 201 Location: {new-user-id}.xml
```

Update

```
PUT /users/#{id}.xml
```

Updates an existing user with new details from the submitted XML.

Request

```
<user>
  <name>Albert Johnson</name>
  <remote-photo-url>http://www.example.com/directory/image.png</remote-
photo-url>
  <current-tags>tag_a tab_b tag_c</current-tags>
  <groups type='array'>
    <group>2</group>
  </groups>
</user>
```

Note that if you do not submit a groups array, the user does not get unassigned from his groups. If you submit an empty group array, the user gets unassigned from all groups:

```
<user>
  <name>The man with no groups</name>
  <groups type='array'></groups>
</user>
```

Response

```
Status: 200
```

Updates the user and sets the user to be a member of group 2 only. Omit the part, if you do not wish to change the users' group status.

You're able to set a users preferred language via the API, either during CREATE or UPDATE. You're able to achieve this by simply setting <locale-id>1</locale-id> where the value is the ID of the corresponding locale. You can find a listing of your currently enabled languages with **GET / account/translations.xml** and a full list of the existent languages with **GET /i18n/translations.xml**. This will only work if you have multiple languages selected, and the locale-id being referenced is turned on in your account (Account -> Personalize your zendesk -> Additional Languages). This is only applicable to End-Users, and for accounts on the Plus+ plan.

Destroy

```
DELETE /users/#{id}.xml
```

Destroys the user at the referenced URL.

The user is not actually deleted from the system, but is set inactive and can no longer be assigned or submit tickets and entries.

Response

```
Status: 200
```

List user identities

```
GET /users/#{user_id}/user_identities.xml
```

Returns all user identities of the given user.

Response

```
<records>
  <record>
    ...
    <value>2222222</value>
    <identity-type>twitter_handle</identity-type>
    <screen-name>the_handle</screen-name>
    ...
  </record>
  <record>
    ...
    <value>aljohson@yourcompany.dk</value>
    <identity-type>email</identity-type>
    ...
  </record>
  ...
</records>
```

Add an email address to a user

```
POST /users/#{user_id}/user_identities.xml
```

Adds an email address for the user specified in the URL.

Request

```
<email>address@example.com</email>
```

Response

```
Status: 201
```

Add a Twitter handle to a user

```
POST /users/#{user_id}/user_identities.xml
```

Adds a Twitter handle for the user specified in the URL.

Request

```
<twitter>a_twitter_handle</twitter>
```

Response

```
Status: 201
```

Make a given user identity the primary for that user

```
POST /users/#{user_id}/user_identities/#{id}/make_primary
```

Response

```
Status: 200
```

Delete a given user identity

```
DELETE /users/#{user_id}/user_identities/#{id}
```

Response

```
Status: 200
```

Enterprise agent roles

In the Enterprise version of Zendesk, you can assign your agents additional pre-defined agent roles or create your own custom agent roles. Unlike the standard end-user, administrator, and agents roles, the Enterprise roles do not have fixed values that are used for all Zendesk accounts. Instead, the Enterprise agent roles have values that are unique to your account. To assign an agent to an Enterprise role using the API, you need to first discover your unique enterprise agent role values using the following API command.

`GET /roles.xml` (or `GET /roles.json`)

XML Response

```
<permission-sets type="array" count="4">
   <permission-set>
      <description>
      Advisors manage the workflow and configure the help desk.
      They create or manage automations, macros, triggers, views,
      and SLA targets. They also set up channels and extensions.
      Advisors don't solve tickets, they can only make private comments.
      </description>
      <id type="integer">8882</id>
      <name>Advisor</name>
   </permission-set>
   <permission-set>
      <description>
      Light agents can view, be CC'd on, and add private comments to
      ticket accessible to them. They can also view reports of all
      tickets within the help desk. Light agents cannot be assigned
      to or edit tickets. You can add an unlimited number of light
      agents at no charge.
      </description>
      <id type="integer">8887</id>
      <name>Light Agent</name>
   </permission-set>
   <permission-set>
      <description>
      A Staff agent's primary role is to solve tickets. They can edit
      tickets within their groups, view reports, and add or edit personal
      views and macros.
      </description>
      <id type="integer">8872</id>
```

```
      <name>Staff</name>
   </permission-set>
   <permission-set>
      <description>
      Team leaders have greater access to the help desk than staff agents.
      They can read and edit all tickets, moderate forums, and create and
      edit end-users, groups, and organizations.
      </description>
      <id type="integer">8877</id>
      <name>Team Leader</name>
   </permission-set>
</permission-sets>
```

The response example above lists all the predefined agent roles. If you've created custom agent roles, these will be listed as well.

```
<permission-set>
   <description>
   This is the description for my custom agent role.
   </description>
   <id type="integer">8890</id>
   <name>My custom agent role</name>
</permission-set>
```

Once you've discovered the role IDs for your Enterprise agent roles, you can assign these roles to your agents via the API.

Note: The Enterprise agent roles can only be assigned to agents (users with their role set to 4).

`PUT /users/#{id}.xml`

Updates an agent's profile to include the Enterprise agent role specified in the submitted XML.

Request

```
<user><custom_role_id>8877</custom_role_id></user>
```

Response

```
Status: 200
```

You can see the change to the agent's role by viewing their user profile and then selecting **Role & Groups**.

Tags

Tags can be attached to tickets and topics. Tags cannot be created or updated alone, but are always submitted in conjunction with create/update of an asset type.

List

`GET /tags.(xml|json)`

Returns the 100 most used tags. The list is updated once a day.

Response

```
Status: 200
```

```
<tags>
  <tag>
    <count type="NilClass">10</count>
    <id>63</id>
    <name>printer</name>
  </tag>
  <tag>
    <count type="NilClass">8</count>
    <id>62</id>
    <name>XP</name>
  </tag>
  ...
</tags>
```

List assets associated with given tags

```
GET /tags/#{tag}.(xml|json)?for=(ticket|entry)
```

Returns all assets associated with the given tags. Delimit tags with “+”. The list is paginated using offsets. If 15 elements are returned (the page limit), use ?page=2 to check for the next 15 and so on.

Response

```
Status: 200

<records>
  <ticket>
    ...
  </ticket>
  <ticket>
    ...
  </ticket>
  ...
</records>
```

Forums

Show

```
GET /forums/#{id}.xml
```

Returns a single forum.

Response

```
Status: 200

<forum>
  <id>6</id>
  <name>Announcements</name>
  <description>New features, fixes, and other important announcements.</
description>
  <entries-count>10</entries-count>
  <posts-count>12</posts-count>
  <is-locked>true</is-locked>
  <is-public>true</is-public>
</forum>
```

List

```
GET /forums.xml
```

Returns all forums.

Response

```
Status: 200

<forums>
  <forum>
    ...
  </forum>
  <forum>
    ...
  </forum>
</forums>
```

Create

```
POST /forums.xml
```

Creates a new forum.

Request

```
<forum>
<name>Knowleddge Base</name>
<description>Knowledge Base</description>
<is-locked type="boolean">false</is-locked>
<visibility-restriction-id type="integer">1</visibility-restriction-id>
</forum>

{:id => 1, :label => "Everybody"},
{:id => 2, :label => "Logged-in users"},
{:id => 3, :label => "Agents only"}
```

Response

```
Status: 201
Location: {new-forum-id}.xml
```

Update

```
PUT /forums/#{id}.xml
```

Updates an existing forum with details from the submitted XML.

Request

```
<forum>
  <is_public>false</is_public>
</forum>
```

Response

```
Status: 200
```

Forum set not to be public, i.e. not visible to end-users.

Destroy

```
DELETE /forums/#{id}.xml
```

Destroys the forum with the referenced ID. All related entries (topics) and posts are destroyed as well.

Response

```
Status: 200
```

Entries

"Entries" is the common URL location for forum topics.

Show

```
GET /entries/#{id}.xml
```

Returns a single entry.

Response

```
Status: 200
<entry>
  <id>12</id>
  <forum-id>6</forum-id>
  <title>This is the title of the topic</title>
  <body>This is the body of the topic</body>
  <created-at>2007-01-03T00:10:08Z</created-at>
  <updated-at>2007-06-04T10:07:02Z</updated-at>
  <hits>3</hits>
  <posts-count>1</posts-count>
  <is-locked>false</is-locked>
  <is-pinned>true</is-pinned>
  <is-public>false</is-public>
  <submitter-id>3</submitter-id>
</entry>
```

List

```
GET /forums/#{forum_id}/entries.xml
```

Returns all entries (topics) for a given forum. The list is paginated using offsets. If 25 elements are returned (the page limit), use ?page=2 to get the next 25 entries, and so on.

Response

```
Status: 200

<entries count="87">
  <entry>
    ...
  </entry>
  <entry>
    ...
  </entry>
```

```
</entries>
```

Create

```
POST /entries.xml
```

Creates a new entry.

Request

```
<entry>
  <forum-id>6</forum-id>
  <title>This is the title of the entry</title>
  <body>This is the text of the first post for the entry</body>
  <current-tags>printer ink</current-tags>
  <is-locked>false</is-locked>
  <is-pinned>true</is-pinned>
</entry>
```

Submitter of the entry is set to the authenticated user.

Response

```
Status: 201
Location: {new-entry-id}.xml
```

Update

```
PUT /entries/#{id}.xml
```

Updates an existing entry with new details from the submitted XML.

Request

```
<entry>
  <is-pinned>false</is-pinned>
  <additional-tags>printing</additional-tags>
</entry>
```

Response

```
Status: 200
```

Entry set not to be pinned and “printing” tag added. You can remove tags with remove-tags and set tags with set-tags.

Destroy

```
DELETE /entries/#{id}.xml
```

Destroys the entry with the referenced ID. All related posts are destroyed as well.

Response

```
Status: 200
```

List posts

```
GET /entries/#{id}/posts.xml
```

Lets you iterate all the posts for a given entry. You can supply a page parameter to get next page, i.e.: /entries/#{id}/posts.xml?page=#{page}.

Response

```
Status: 200

<?xml version="1.0" encoding="UTF-8"?>
<posts type="array" count="7">
 <post>
   <account-id type="integer">1</account-id>
   <body>This was an awesome article, thanks so much</body>
   <created-at type="datetime">2011-02-03T11:41:12-08:00</created-at>
   <entry-id type="integer">43</entry-id>
   <forum-id type="integer">12</forum-id>
   <id type="integer">594718</id>
   <is-informative type="boolean">false</is-informative>
   <updated-at type="datetime">2011-02-03T11:41:12-08:00</updated-at>
   <user-id type="integer">2640</user-id>
 </post>

 ...
 <post>
   <account-id type="integer">1</account-id>
   <body>You make me happy.</body>
   <created-at type="datetime">2011-02-03T08:25:51-08:00</created-at>
   <entry-id type="integer">43</entry-id>
   <forum-id type="integer">12</forum-id>
   <id type="integer">598</id>
   <is-informative type="boolean">false</is-informative>
   <updated-at type="datetime">2011-02-03T08:25:51-08:00</updated-at>
   <user-id type="integer">2647</user-id>
 </post>
</posts>
```

Add post

`POST /posts`

Add a post to an existing entry.

Request

```
<post>
  <entry-id>43</entry-id>
  <body>This is a comment to an entry</body>
</post>
```

Response

```
Status: 200
```

Update post

`PUT /posts/#{post_id}`

Edits an existing post.

Request

```
<post>
```

```
  <body>This is a edited comment to an entry</body>
</post>
```

Response

```
Status: 200
```

Destroy post

```
DELETE /posts/#{id}.xml
```

Destroys the post with the referenced ID.

Response

```
Status: 200
```

Note: You can decide if the creation of a topic, or a comment within a topic, triggers an email notification to subscribers. This is handy when bulk importing a large amount of topics and comments via the API. When making a POST to entries.xml or posts.xml, just include `<silence-notification>true</silence-notification>` to stop any notifications being generated.

Search

Your zendesk contains a unified search interface for users, groups, organizations, tickets and topics, and offers a wide range of search parameters.

You can use all available search parameters in your REST search. Examples:

- type:user company.com (search for users from a specific domain)
- printer fire assignee:alexander
- assignee:none
- requester:me@zendesk.com
- submitter:“Joe Doe”
- group:support
- group:none
- status:solved
- priority>normal
- tags:printer

Note: Your help desk is indexed every 15 minutes. You might see a delay in the contents of the index.

More search information.

Sorting

You can sort and order your search results. Supported values are:

- sort:asc
- sort:desc
- order_by:priority
- order_by:status
- order_by:ticket_type
- order_by:updated_at
- order_by:created_at

Note: You have to use both terms to get correct results. Example: “order_by:created_at sort:desc” or ”order_by:updated_at sort:asc”.

List

```
GET /search.xml?query=#{query term}
```

Returns all assets matching the query. The list is paginated using offsets. If 15 elements are returned (the page limit), use ?page=2 to check for the next 15 and so on.

Response

```
Status: 200

<records count="342">
  <record>
    ...
  </record>
  <record>
    ...
  </record>
</records>
```

Ticket fields

Ticket fields consist of standard fields that came with your Zendesk, such as Subject and Description, or custom fields that you have defined in the Ticket Fields configuration of your Zendesk.

List all

```
GET /ticket_fields.xml
```

Returns a list of all ticket fields in your Zendesk. Fields are returned in the order that you specify in your Ticket Fields configuration in Zendesk. Clients should cache this resource for the duration of their API usage and map the value of `id` for each ticket field to the values returned under the `ticket_field_entries` attributes on the Tickets resource.

Response

```
Status: 200

<records type="array" count="14">
  <record>
    <account-id type="integer">17721</account-id>
    <created-at type="datetime">2009-10-18T20:56:52-07:00</created-at>
    <default-value nil="true"></default-value>
    <description>Please enter the details of your request. A member of our
 support staff will respond as soon as possible.</description>
    <id type="integer">120913</id>
    <is-active type="boolean">true</is-active>
    <is-editable-in-portal type="boolean">true</is-editable-in-portal>
    <is-required type="boolean">false</is-required>
    <is-required-in-portal type="boolean">true</is-required-in-portal>
    <is-visible-in-portal type="boolean">true</is-visible-in-portal>
    <position type="integer">0</position>
```

```
    <regexp-for-validation nil="true"></regexp-for-validation>
    <sub-type-id type="integer" nil="true"></sub-type-id>
    <title>Description</title>
    <title-in-portal>Description</title-in-portal>
    <type>FieldDescription</type>
    <updated-at type="datetime">2010-07-09T15:04:37-07:00</updated-at>
  </record>
  <record>
    <account-id type="integer">17721</account-id>
    <created-at type="datetime">2009-10-18T20:56:52-07:00</created-at>
    <default-value nil="true"></default-value>
    <description></description>
    <id type="integer">120912</id>
    <is-active type="boolean">true</is-active>
    <is-editable-in-portal type="boolean">true</is-editable-in-portal>
    <is-required type="boolean">false</is-required>
    <is-required-in-portal type="boolean">true</is-required-in-portal>
    <is-visible-in-portal type="boolean">true</is-visible-in-portal>
    <position type="integer">1</position>
    <regexp-for-validation nil="true"></regexp-for-validation>
    <sub-type-id type="integer" nil="true"></sub-type-id>
    <title>Subject</title>
    <title-in-portal>Subject</title-in-portal>
    <type>FieldSubject</type>
    <updated-at type="datetime">2010-06-24T22:23:18-07:00</updated-at>
  </record>
  ...
  <record>
    <account-id type="integer">17721</account-id>
    <created-at type="datetime">2010-04-15T11:41:50-07:00</created-at>
    <default-value nil="true"></default-value>
    <description></description>
    <id type="integer">224471</id>
    <is-active type="boolean">true</is-active>
    <is-editable-in-portal type="boolean">false</is-editable-in-portal>
    <is-required type="boolean">false</is-required>
    <is-required-in-portal type="boolean">false</is-required-in-portal>
    <is-visible-in-portal type="boolean">false</is-visible-in-portal>
    <position type="integer">13</position>
    <regexp-for-validation>[a-z]</regexp-for-validation>
    <sub-type-id type="integer" nil="true"></sub-type-id>
    <title>Custom Regex Field</title>
    <title-in-portal>Custom Regex Field</title-in-portal>
    <type>FieldRegexp</type>
    <updated-at type="datetime">2010-07-09T15:04:37-07:00</updated-at>
  </record>
</records>
```

Macros

Macros are predefined ticket field and ticket comment values which can be applied to a ticket for sending consistent responses to ticket requesters. Currently macros can only be listed and applied to tickets via the API.

List all

```
GET /macros.xml
```

Returns an ordered list of available macros. Response consists of macro records with ID, title and availability type (everyone, group or personal).

```
Status: 200

<macros>
  <macro>
    <id type="integer">110</id>
    <title>Mark as Incident</title>
    <availability-type>everyone</availability-type>
  </macro>
  <macro>
    <id type="integer">106</id>
    <title>Assign</title>
    <availability-type>personal</availability-type>
  </macro>
  <macro>
    <id type="integer">68</id>
    <title>Close and redirect to topics</title>
    <availability-type>personal</availability-type>
  </macro>
</macros>
```

Nested macros

For Zendesk accounts which use many macros, Zendesk allows administrators to nest macros for easier categorization and access by support agents. Nesting of macros is represented by the presence of a delimiter (the string "::") in the macro name. API clients are expected to use this delimiter to display available macros in a hierarchical fashion.

For example, if there are three macros named "Respond::SLA Platinum", "Respond::SLA Gold", "Respond::SLA Silver" in the API response, it means that there is a top level group of macros named "Respond", which contains the actual macros, called "SLA Platinum", "SLA Gold" and "SLA Silver".

This nesting can be multiple levels deep. Using the above example, another macro named "Respond::SLA Platinum::Billing" which implies a three-level hierarchy.

Evaluate

```
POST /macros/#{macro-id}/apply.xml?ticket_id=#{ticket-id}
```

Apply a macro to a ticket by ID, specified in the ticket_id query parameter. For applying a macro to a new ticket, use ticket_id=0.

The response will contain a collection of key value pairs corresponding to ticket attributes. The format of these key value pairs follows the request formats used in the Ticket API. API clients should apply these attributes to the ticket 'form' being displayed to the user.

Response

```
Status: 200

<ticket>
  <comment>
    <value>Foo</value>
    <is-public>false</is-public>
  </comment>
  <current-tags>foobar2</current-tags>
```

```
  <ticket-type-id>2</ticket-type-id>
  <group-id>4</group-id>
  <assignee-id>4</assignee-id>
</ticket>
```