Creating and updating tickets

ON THIS PAGE

This article explains how to set ticket properties and specify actions when creating and updating a ticket. For reference information, see the Tickets resource in the Ticketing API reference.

Creating a follow-up ticket

Once a ticket is closed (as distinct from solved), it can't be reopened. However, you can create a new ticket that references the closed ticket. To create the follow-up ticket, include a `via_followup_source_id` parameter that specifies the closed ticket. Example:

```
curl https://{subdomain}.zendesk.com/api/v2/tickets.json \
  -d
'{"ticket": {"via_followup_source_id": 103, "comment": {"body": "My printer is
still too hot!"}}}'
```

```
      \
3     -v -u {email_address}/token:{api_token} -X POST -H
    "Content-Type: application/json"
```

The parameter only works with closed tickets. It has no effect with other tickets.

By default, the first comment author is the authenticated user making the API request. To set a different author, include an `author_id` property in the comment object.

Follow-up tickets inherit the tags from the parent ticket upon creation, with one exception. If you create a follow-up ticket and include tags in the request, those tags will replace the parent ticket's tags.

Creating a ticket asynchronously

Ticket creation can sometimes take a while because of complex business rules. Asynchronous creation allows you to get the response back quickly and queues a background job to do the actual work.

To make an asynchronous request, simply add the parameter `async` in your request:

```
POST api/v2/tickets.json?async=true
```

As soon as the request is received, it returns a `202 Accepted`. The response includes the new ticket `id` with a `job_status` JSON object. However, you may not be able to fetch the ticket until the job is done.

Note: Some ticket id numbers may get skipped if ticket creation fails.

Setting collaborators

Note: Email CCs and followers only work if you are using CCs and Followers. See Setting email CCs and Setting followers. You can still set collaborators with the `collaborators` and `additional_collaborators` ticket properties described in this section. They'll automatically make end users CCs and agents followers.

When creating or updating a ticket, you can set collaborators on a ticket using one of three properties: `collaborator_ids`, `collaborators` or `additional_collaborators`. The `collaborators` property provides more flexibility, as outlined below.

An email notification is sent to the collaborators when the ticket is created or updated.

Setting `collaborators` or `collaborator_ids` when updating a ticket replaces existing collaborators. Make sure to include existing collaborators in the update request if you want to retain them on the ticket. If you only want to add more collaborators, use `additional_collaborators` instead.

The `collaborator_ids` property is a basic option that takes an array of user ids.

Example

```
{
  "ticket": {
    "collaborator_ids": [562624, 624562, 243642]
  }
}
```

The `collaborators` property is a more flexible option. It takes an array consisting of a combination of user ids, email addresses, or objects containing a `name` and `email` property. For example:

- 562562562
- "someone@example.com"
- `{ "name": "Someone Special", "email": "someone@example.com" }`

Use an object to create the user on the fly with the appropriate name in Zendesk Support.

Example

```
{
  "ticket": {
    "collaborators": [ 562, "someone@example.com", { "name": "Someone Else", "email": "else@example.com" } ]
  }
}
```

The `additional_collaborators` property is used to add additional collaborators. It preserves the existing collaborators, and adds new collaborators to the ticket. It takes an array consisting of a combination of user ids, email addresses, or objects containing a `name` and `email` property.

Example

```
{
  "ticket": {
    "additional_collaborators": [ 562, "someone@example.com", { "name": "Someone Else", "email": "else@example.com" } ]
  }
}
```

Even if you use the `collaborators` property, the JSON response will specify the new or updated collaborators as an array of user ids in the `collaborator_ids` property. The response doesn't have a `collaborators` property.

Setting followers

When creating or updating a ticket, you can set followers on a ticket using the `followers` property.

An email notification is sent to the followers when the ticket is created or updated.

The `followers` property takes an array of objects representing agents. Each object must have an identifier, either user ID (`user_id`) or email address (`user_email`). If a `user_email` or `user_id` is given which doesn't exist in the account, that user object is ignored. Additionally, the `action` key can be set to either "put" or "delete" to indicate whether the user should be added to or removed from the followers list. If the `action` key is not given, the default value is "put".

Example

```
{
  "ticket": {
    "followers": [
      { "user_id": "562624", "action": "put" },
      { "user_id": "624562" },
      { "user_id": "243642", "action": "delete" },
      { "user_id": "562", "user_email": "existing-user1@example.com", "action": "delete" },
      { "user_email": "existing-user2@example.com", "action": "put" },
    ]
  }
}
```

Setting email CCs

When creating or updating a ticket, you can set email CCs on a ticket using the `email_ccs` property.

A single email notification is sent to the requester and CCs when the ticket is created or updated, depending on trigger settings.

The `email_ccs` property takes an array of objects representing users. Each object must have an identifier, either user ID (`user_id`) or email address (`user_email`). If an email address is specified and the user doesn't exist in the account, a new user is created. Additionally, the key `action` can be set to either "put" or "delete" to indicate whether the user should be added to or removed from the email CCs list. If the `action` key is not given, the default value is "put". Optionally, if a new user is created, you can specify `user_name` to set the new user's name. If the user is implicitly created and a `user_name` is not given, the user name is derived from the local part of the email address. If an email address not associated with a user is included (for example, with the intention to implicitly create the user), but the value of the `action` key is "delete", that email address is not be created or added as an email CC.

A ticket can only have 48 email CCs. If more than 48 email CCs are included in the request, a "400 Bad Request" will be returned. If existing email CCs and the additional email CCs added through the request add up to more than 48, the email CCs will be truncated to 48 and no error will be reported.

Email CCs will not be updated if an internal note is also added to the ticket in the same update.

Example

```
{
  "ticket": {
    "email_ccs": [
      { "user_id": "562624", "action": "put" },
      { "user_id": "243642", "action": "delete" },
      { "user_email": "else@example.com", "user_name": "Someone Else", "action": "put"}
    ]
  }
}
```

Setting metadata

When you create or update a ticket, an Audit gets generated if the ticket properties have changed. On each such audit, you can add up to 1 kilobyte of custom metadata. You can use this to build your own integrations or apps. **Note**: If your update does not change the ticket, this will not create an Audit and will not save your metadata.

Example

```
{
  "ticket": {
    "metadata": { "time_spent": "4m12s", "account": "integrations" },
    "comment":  { "body": "Please press play on tape now" },
    "status":   "pending"
  }
}
```

Note that metadata can only be set as part of other regular ticket updates as they are associated to, rather than just the ticket. Zendesk Support also adds metadata on each ticket update, and the resulting audit JSON structure looks like this:

```
{
  "audit": {
    "id":         35436,
    "ticket_id":  47,
    "created_at": "2012-04-20T22:55:29Z",
    "author_id":  35436,
    "metadata": {
      "custom": {
```

```
        "time_spent": "4m12s",
        "account": "integrations"
      },
      "system": {
        "ip_address": "184.106.40.75",
        "location": "United States",
        "longitude": -97,
        "latitude": 38,
        "client": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3)"
      }
    },
    "via": {
      "channel": "web"
    },
    "events": [
      {
        "id":          1564245,
        "type":        "Comment"
        "body":        "Please press play on tape now",
        "public":      true,
        "attachments": []
      },
      ...
    ]
  }
}
```

Attaching files

When creating and updating tickets, you may attach a file to a comment by including a token received from uploading the file.

For details and examples, see Attaching ticket attachments with the API.

Example

```
{
  "ticket": {
    "comment":  { "body": "Please press play on tape now", "uploads":  [ "vz7ll9ud8oofowy"] }
  }
}
```

Creating a ticket with a new requester

You can specify a requester when creating a ticket. A name, email, and locale id can be set for the requester. A name and email are required. For locale ids, see the Locales API.

Example

```
{
  "ticket": {
    "subject":   "Hello",
    "comment":   { "body": "Some question" },
    "requester": { "locale_id": 8, "name": "Pablo", "email":
"pablito@example.org" }
  }
}
```

If the requester's identity doesn't exist in Zendesk, a new user profile is created on the user's behalf. The user needs to verify their identity to view their requests.

If a user exists in Zendesk with the specified email address, then Zendesk uses that user and makes no updates to existing users during the ticket creation process. In this approach, only the email attribute is required.

Setting custom field values

To set the value of one or more custom fields in a ticket, specify an array of objects consisting of `id` and `value` properties. For text or numeric fields, specify text or numbers for the `value` property:

```
"custom_fields": [{"id": 25356371, "value": "I bought it at Buy More."}, ...]
```

For date fields, specify only the date (formatted as a ISO 8601 string) and omit the time:

```
"custom_fields": [{"id": 25356634, "value": "2020-02-24"}, ...]
```

For fields with values defined by tags, such as drop-down lists, specify the option's tag name for the `value` property, not the option text displayed in the list. For example, if the option's display text is "HD 3000 color printer" and its tag is hd_3000, set the custom field's value as follows:

```
"custom_fields": [{"id": 21938362, "value": "hd_3000"}]
```

Multi-select field values are also defined by tags. However, the value property is a comma-separated list. For example, if the values of the tags are hd_3000 and hd_5555, the custom field's values is as follows:

```
"custom_fields": [{"id": 21938362, "value": ["hd_3000", "hd_5555"]}]
```

For lookup relationship fields, you can the target record id or the unique external id.

For example, to set a lookup field's value using the target record's id (`67890`), use the following structure:

```
"custom_fields": [{"id": 12345, "value": "67890"}]
```

Alternatively, to set a lookup field's value using an external id (`object_identifier`), use the following structure:

```
"custom_fields": [{"id": 12345, "value": "external_id:object_identifier"}]
```

For more information, see Ticket Fields and Setting ticket lookup field values..

Example

```
{
  "ticket": {
    "subject": "Hello",
    "comment": { "body": "Some question" },
    "custom_fields": [{ "id": 34, "value": "I need help!" }]
  }
}
```

Protecting against ticket update collisions

As with any database application, it's good practice to guard against record update collisions. If multiple updates to the same ticket are performed almost simultaneously, the updates might collide and result in data loss. For example, take a ticket with two tags, "red" and "blue". One update attempts to append "green" to the list of tags. A separate update attempts to append "yellow". If the two updates are made at almost the same time, the following sequence might unfold:

1. Update A reads ["red", "blue"]
2. Update B reads ["red", "blue"]
3. Update A writes ["red", "blue", "green"]
4. Update B writes ["red", "blue", "yellow"]

The data "green" is lost because Update B was allowed to proceed with an out-of-date array.

Zendesk implements optimistic locking across all API endpoints involved in saving tickets. This concurrency control mechanism is designed to prevent race conditions, ensuring that one ticket update does not

inadvertently overwrite another. Any collisions that occur on unprotected endpoints return a 409 Conflict value.

As an extra layer of protection, you can make safe updates to prevent updates such as Update B from proceeding. In a safe update, you include the last known ticket update with your request to check for more recent updates. If the system detects a ticket update more recent than the last known update, it prevents your update from proceeding and sends a response that lets you know that the ticket wasn't updated. As a developer, you should account for this result and get a fresh copy of the ticket data and try the update again. See Checking if the ticket was not updated below.

To enable safe updates, include the `safe_update` and `updated_stamp` properties in the ticket object in your PUT request to check for more recent updates. Example:

```
{
  "ticket": {
    "subject": "Updated Subject",
    "updated_stamp": "2021-08-24T16:53:51Z",
    "safe_update": true,
    ...
  }
}
```

The value of `updated_stamp` is the last known time the ticket was updated at the moment you make your update request. The last known time is the time currently specified by the ticket's `updated_at` property. Make a preliminary request to Show Ticket to get this value. For placeholders in webhook workflows, the last known update time is `ticket.updated_at_with_timestamp`.

When updating many tickets at once, you can also specify `safe_update` and `updated_stamp` properties for each ticket object. This protects against ticket update collisions and returns a message to let you know if one occurs.

Checking if the ticket was not updated

When making safe updates, the request returns a 409 status code or an "UpdateConflict" job status to let you know that the ticket was not updated because of a more recent update. It's a signal to get the latest ticket data and retry your update.

When updating a single ticket, instrument your code to look for a 409 status code in the response and handle it accordingly. Example 409 response:

```
Status: 409 Conflict
{
   "description": "Safe Update prevented the update due to outdated ticket data.
                   Please fetch the latest ticket data and try again.",
   "error": "UpdateConflict"
```

```
  }
```

When updating tickets in bulk or in batch, instrument your code to monitor the job's status for "UpdateConflict" errors and handle it accordingly. Example job status:

```
{
  "job_status": {
    "results": [
      {
        "index": 0
        "error": "UpdateConflict",
        "id": 9999,
        "details":
    "Safe Update prevented the update due to outdated ticket data. Please fetch the
    latest ticket data and try again."

      }
    ],
    ...
  }
}
```

Behavior of ticket attributes when CCs and Followers is not enabled

When CCs and Followers is not enabled, the following behaviors apply to the respective attributes:

- **Collaborators**: Collaborators can still be added using the Setting Collaborators method. These collaborators will receive updates on the ticket but will not have any visibility into the ticket history unless explicitly included in updates.
- **Followers**: The Setting Followers method will not function as intended. Followers will not receive notifications or updates related to the ticket.
- **Email CCs**: The Setting Email CCs method will not function as intended. Email addresses added as CCs will not receive any notifications or updates regarding the ticket status or changes.