Using the Incremental Exports API

ON THIS PAGE

You can use the Incremental Exports API to get items that changed or were created in Zendesk Support since the last request. However, any items that were created or changed within the minute prior to your request are excluded. This ensures the accuracy of the data.

It works like this:

- **Request at 5pm**: "Give me all the tickets that changed since noon today."
- **Response**: "Here are the tickets that changed since noon up until, and including, 5pm."
- **Request at 7pm**: "Give me the tickets that changed since 5pm."
- **Response**: "Here are the tickets that changed since 5pm up until, and including, 7pm."

Incremental export workflow

Use the API initially to export a complete list of items from some arbitrary milestone, then periodically poll the API to incrementally export items that have been added or changed since the previous poll. You should

not use this API to repeatedly export complete data sets.

The workflow is as follows:

1. Define a start time for the first scheduled export.

   The API will return all the items that were created or changed on or after the start time.

   This is a one-time requirement.
2. Export the items.

   The Incremental Ticket Export and Incremental User Export endpoints support two export methods: cursor-based and time-based. The other endpoints only support time-based exports.

   Note: These methods are different from the cursor pagination method used in other endpoints.

   If you want to use cursor-based incremental exports for tickets (recommended), use the following API path:

   ```
   GET /api/v2/incremental/tickets/cursor.json?start_time={unix_time}
   ```

   With cursor-based incremental exports, each page of results includes a cursor pointer to use as the starting point for the next page or the next export. You don't need a start time for the next page or export and the API doesn't provide one.

   Where available, choosing cursor-based exports is highly encouraged. Cursor pagination provides more consistent performance and response body sizes.

   If you choose time-based incremental exports, use the following API path for the initial export:

   ```
   GET /api/v2/incremental/tickets.json?start_time={unix_time}
   ```

   With time-based incremental exports, each page includes an end time to use as the start time for the next page or the next export. It doesn't provide you with a cursor pointer.

   In either export method, read the `end_of_stream` boolean value on each page to determine when the last page of results has been reached. When true, the last page has been reached.
3. Depending on your export method, save the cursor pointer or the end time specified on the last page of results.
4. At the next scheduled export, retrieve the saved cursor pointer or end time.
5. Use the cursor pointer or end time as the starting point for the export.

   For example, the following export uses the previous export's last cursor value as its starting point:

   ```
   GET /api/v2/incremental/tickets/cursor.json?cursor=MTU4MDc1Mzc5OC4wfHw0MzJ8
   ```

   The following export uses the previous export's end time as its starting point:

```
GET /api/v2/incremental/tickets.json?start_time=1568330298
```

6. Repeat steps 3 through 5 for subsequent exports.

   To prevent race conditions, the ticket and ticket event export endpoints will not return data for the most recent minute. In time-based exports, the returned `end_time` property (or the `start_time` parameter in the `next_page` URL) will never be more recent than one minute ago.

Cursor-based incremental exports

In cursor-based incremental exports, each page of results includes an "after" cursor pointer to use as the starting cursor for the next page of results. When all the results have been returned, save the after cursor pointer on the last page and use it as the starting cursor of the next export.

Cursor-based incremental exports are only supported for tickets and users. Using cursor-based incremental exports for these resources provides more consistent response body sizes and performance.

Use the following path for the initial request:

```
GET /api/v2/incremental/tickets/cursor.json?start_time={unix_time}
```

After using the start_time parameter in the initial request, use the cursor parameter for all subsequent results pages as well as for all subsequent exports:

```
GET /api/v2/incremental/tickets/cursor.json?cursor={cursor_pointer}
```

The cursor pointer is included in the `after_cursor` property as well as in the `after_url` property in each results page:

```
{
  "tickets": [...],
  "after_url":
"https://example.zendesk.com/api/v2/incremental/tickets/cursor.json?
cursor=MTU4MDc1Mzc5OC4wfHw0MzJ8"
,
  "after_cursor": "MTU4MDc1Mzc5OC4wfHw0MzJ8",
  ...
  "end_of_stream": false
}
```

Use the `end_of_stream` property to determine when to stop paginating. If `end_of_stream` is false, continue paginating using the `after_cursor` (or the `after_url` URL) to get the next page of results. If `end_of_stream` is true, the last page of results has been reached. Stop paginating and save the `after_cursor` pointer (or the `after_url` URL) for your next export.

In your next export, use the cursor pointer you saved to pick up where you left off. In the previous example, the `after_cursor` pointer in the last page of results was "MTU4MDc1Mzc5OC4wfHw0MzJ8". Start the

next export with the cursor pointer:

```
GET /api/v2/incremental/tickets/cursor.json?cursor=MTU4MDc1Mzc5OC4wfHw0MzJ8
```

Time-based incremental exports

In time-based incremental exports, each page of results includes an end time to use as the start time for the next page of results. When all the results have been returned, save the last page's end time and use it as the start time of the next export.

All the incremental export endpoints support time-based exports.

The path for the initial request looks as follows:

```
GET /api/v2/incremental/{items}.json?start_time={unix_time}
```

The items can be tickets, ticket events, users, organizations, and more.

After the initial request, continue using the `start_time` parameter to fetch subsequent pages. The next start time value is specified by the `end_time` property as well as the `next_page` URL included in each results page:

```
{
  "tickets": [...],
  "next_page":
"https://example.zendesk.com/api/v2/incremental/tickets.json?
start_time=1542953046"
,
  "end_time": 1542953046,
  ...
  "end_of_stream": false
}
```

The time in both cases, 1542953046, is equal to the `generated_timestamp` time of the last item in the page.

Because of limitations with time-based pagination, subsequent responses may contain duplicate items. See Excluding duplicate items for more information.

Use the `end_of_stream` property to determine when to stop paginating. If `end_of_stream` is false, continue paginating using the `end_time` (or the `next_page` URL) to get the next page of results. If `end_of_stream` is true, stop paginating and save the `end_time` value (or the `next_page` URL) for your next scheduled export.

In your next export, use the time you saved to pick up where you left off. In the previous example, the `end_time` value in the last page of results was "1542953046". You'd start the next scheduled export with

the value:

```
GET /api/v2/incremental/tickets.json?start_time=1542953046
```

Note: Time-based pagination includes a `count` property on each page but it shouldn't be used as a method to detect the last page of results. These endpoints return up to 1000 items per page in normal circumstances, but not always. The 1000-item limit may be exceeded if items share the same timestamp. As a result, `count` is not a reliable indicator of completeness. Instead, use `end_of_stream` to determine when to stop paginating.

Polling strategy for time-based exports

Time-based incremental exports don't protect against duplicates caused by requests that cover overlapping periods. Each query boils down to searching for items updated after or at your `start_time` value. The same item can be included in multiple exports if the start time of each request is earlier than the time the item was modified.

Alternatively, you can miss records if you leave gaps between the `end_time` of the previous request and the `start_time` of the next.

To prevent gaps between requests, use the last `end_time` value of the previous export as the `start_time` of the next scheduled export.

Excluding duplicate items

Because of limitations with time-based pagination, the exported data may contain duplicate items.

You can exclude the duplicate items after getting all the results by filtering out any items that share the following property values with a previous item:

Export	Filter values
tickets	`id + updated_at`
ticket events	`id + created_at`
users	`id + updated_at`
organizations	`id + updated_at`

Time-based exports contain duplicate items to prevent items that were updated or created at the same time from being skipped if they're at the end of a page. For example, assume the `per_page` parameter is 50 and three tickets in the data set were updated at the same time. If the first of the three tickets is at position 50 on page 1, then the start time of page 2 will be set to return tickets created or updated after the last ticket on page 1. If the last ticket isn't carried over to the next page, two tickets would be skipped.

Exporting tickets

Use the Incremental Ticket Export endpoint to export tickets created or updated since the last request.

You can export tickets using cursor-based exports or time-based exports. Zendesk recommends using cursor-based exports. See Cursor-based incremental exports.

To use time-based exports, see Time-based incremental exports.

Excluding deleted tickets

Deleted tickets still appear in exports because the ticket record still exists. Zendesk scrubs user-provided information in tickets 30 days after they're deleted or immediately if tickets are manually permanently deleted.

The following occurs when a ticket is scrubbed:

- The subject and description are replaced with "SCRUBBED"
- Information in text, dropdown, multi-select, and date fields are replaced with an X
- Numeric field values are replaced with a 0
- Fields that didn't have a value are blank

You can exclude these tickets after getting all the results by filtering out any tickets with a `status` of "deleted".

Note: Zendesk began scrubbing deleted tickets on October 16, 2016.

Excluding system-updated tickets (time-based exports)

Incremental ticket exports can return tickets that were updated by the system for reasons not related to ticket events occurring in the normal course of business. An example of this kind of system update is a database backfill by Zendesk.

You can exclude these tickets after getting the results by filtering out any record with an `updated_at` time that's earlier than the `start_time` time.

The reasoning for this rule is as follows. The `updated_at` property is not used to record system updates. System updates are recorded by the `generated_timestamp` property (as are all other ticket updates). The `updated_at` property is only used for ticket updates that generate a defined ticket event. Therefore, any ticket in the results that was only updated by the system will have a `generated_timestamp` that's later than the `start_time` but an `updated_at` time that's earlier than the `start_time`.

Cursor-based incremental exports don't have a `start_time` to compare with the `updated_at` time of tickets.

Exporting ticket events

Use the Incremental Ticket Event Export endpoint to export events that occurred on tickets since the last request. Each event is tied to an update on a ticket and contains all the fields that were updated in that change.

Ticket events only support time-based exports. See Time-based incremental exports.

Understanding older ticket events in responses

Ticket events don't change over time so they usually appear at their updated timestamp. In some cases however, the system may change an event after the fact, usually when the ticket is archived and deleted. As a result, the API may return events that occurred before the request's `start_time` time.

The API returns records based on the `generated_timestamp` timestamp, not the `updated_at` timestamp. When tickets are archived, the entire event history moves to the `generated_timestamp` timestamp of the close. When archived tickets are deleted, the entire event history moves to the `generated_timestamp` timestamp of the deletion.

For example, a request with a start time of January 1, 2019, may return ticket events from 2014 because a long-archived ticket was recently deleted.

Note that the event object doesn't have a `generated_timestamp` property.