blizz
by TeamViewer

API Documentation

rev 201707-1.0

Table of Contents

Introduction

Definitions

Name	Description
Organizer	The user who controls the meeting. The initiator of a meeting is always an organizer. To use the Blizz API the organizer needs to have a TeamViewer account.
Participant	A participant in a meeting is anyone who connects to a meeting. Each participant can be assigned to another role as well as certain rights by the presenter and organizers. The user does not need to have a TeamViewer account.
Client	The application (or user, if the HTTP requests are typed in manually) that uses the API.

Instant Blizz Meetings

Blizz meetings can be started instantly via URLs. The meeting ID is the personal meeting ID of the signed in organizer or the meeting ID of a manually scheduled meeting. The API is not needed for that. In case you want to automatically schedule a Blizz meeting you can use the TeamViewer API for that.

Start meeting via URL using the Web Client

The organizer can start an instant Blizz meeting with the personal Blizz meeting ID, called “My meeting ID”.

The organizer must be signed into the web client: https://go.blizz.com

The Web Client calls the installed Blizz application to start the meeting.

- Start a Blizz meeting with your shared screen
  - https://go.blizz.com/startmeeting
  - https://go.blizz.com/startmeeting/sharescreen
- Start a meeting with your web camera and microphone enabled
  - https://go.blizz.com/startmeeting/videocall
- Start a meeting with your microphone enabled
  - https://go.blizz.com/startmeeting/audiocall

Start meeting via URL using the Windows or Mac client directly

The organizer can start an instant Blizz meeting with the personal Blizz meeting ID, called “My meeting ID”.

It the organizer has scheduled a meeting in advance via the Blizz Scheduler or via the Blizz Microsoft Outlook integration, the scheduled Meeting ID can be used.

Example URLs

- Start a meeting with your personal meeting ID: `blizzv1://hostmeeting`
- Start your scheduled meeting: `blizzv1://hostmeeting?meetingid=m12345678`

Commands

- `HostMeetingCommand = L"hostmeeting";`
- `JoinMeetingCommand = L"joinmeeting";`
- `StartVideoCallCommand = L"startvideocall";`
- `StartAudioCallCommand = L"startaudiocall";`

Parameters

- `meetingPasswordParam = L"meetingpassword";`
- `meetingIDParam = L"meetingid";`

The TeamViewer REST API

The TeamViewer API is a REST API which uses the already existing HTTP methods to create (`POST`), read (`GET`), change (`PUT`) or delete (`DELETE`) single items or a collection of items. The following table shows the general use cases for these HTTP methods.

	GET	POST	PUT	DELETE
Collection	retrieve list of items in this collection	create new item in this collection	-	-
Single item	retrieve item data	-	changes the item	deletes this item

The basic URI scheme for all API functions is:
`https://host/path/to/rsources[/id][/verb][?param1=value1]`

The TeamViewer API can be found at
https://webapi.teamviewer.com/

Parameters in the URI are only allowed for GET and DELETE. Generally, there should be no need for any parameters for DELETE, though. POST and PUT need to have the parameters in the body formatted as JSON or XML.

IDs

IDs are prefixed with a type in order to make them more distinguishable. The following types are used:

- "`m`" – meeting ID

Date format

All dates and times follow the ISO 8601. They should have the following format: `YYYY-MM-DD"T"HH:MM:SS"Z"`. Times are always in UTC unless stated otherwise.

Example

```
2013-02-21T13:42:55Z = 21st February 2013, 13:42:55 UTC
```

Number format

Decimal numbers are returned in US English format, using a point as decimal separator. Digits are never grouped by a delimiter.

Example

```
12345.67
```

Rate limits

The rate limit is set to **300 requests per hour**. Rate limits apply per access token per API call.

OAuth 2.0 Authorization

Roles & Terms

For more information about OAuth 2.0, see http://oauth.net/2/ and the official specification at http://tools.ietf.org/html/rfc6749

Names used by the RFC and their meaning for the TeamViewer API:

- **resource owner** – The user behind a TeamViewer account who wants to access their resources through the API.
- **resource server** – Our servers where the API runs.
- **client** – The application, plug-in, script or user who is making the API HTTP requests.
- **authorization server** – In our case that's the same servers that run the rest of the API.
- **client ID** – A unique ID to identify the application that wants to use the TeamViewer API.
- **client secret** – A unique string only known to the creator of the client ID.
- **authorization code** – Code used during the OAuth process to prove that an authorization request was granted in the Management Console.
- **access token** – A token that has to be used to access any API function (except those explicitly marked as not requiring any access tokens).
- **refresh token** – A token that can be used once to obtain a new access token and a new refresh token.

Registering an application in the Management Console

Before using any API functionality, you need to register an application in the TeamViewer Management Console (https://login.teamviewer.com/LogOn).

When you register the application, you must specify if you want to use it for your own account only (private application, also referred to as "**Script**") or if you want to create an application to be used by any TeamViewer or Blizz user (public application, also referred to as "**App**").

In both cases you also specify if the application will have access to the data of one single account or to the data of the entire company.

Client/Application-Types

	Script	App
User Access	Access token is created that can only be used to access the user who created the application.	Client ID is created when creating the application. The Client ID can be used with OAuth to create an access token for the user granting access.

When you register an application for your own use only, you will get an access token that can be used directly for any API function that requires it. When you register the application for others to use as well, you will get a Client ID. This Client ID is used in the OAuth process described below. At the end of this process the application will also have an access token that must be used by the other API functions. This access token is tied to the account/company that **uses** the application, not the company that **created** the application.

Access Token Permissions and Scopes

Access tokens have several permissions attached to them. This is called the scope of the access token.

The following table shows all available scopes.

API function	Scopes
Account (user level access only)	`Account.Create`, `Account.Read`, `Account.ReadEmail`, `Account.ReadEmailLicense`, `Account.Modify`, `Account.ModifyEmail`, `Account.ModifyPassword`
Groups	`Groups.Create`, `Groups.Read`, `Groups.Modify`, `Groups.Share`, `Groups.Delete`
Users	`Users.CreateUsers`, `Users.CreateAdministrators`, `Users.Read`, `Users.ModifyUsers`, `Users.ModifyAdministrators`
Sessions	`Sessions.Create`, `Sessions.ReadAll`, `Sessions.ReadOwn`, `Sessions.ModifyAll`, `Sessions.ModifyOwn`
Connections	`Connections.Read`, `Connections.Modify`, `Connections.Delete`
Meetings (user level access only)	`Meetings.Create`, `Meetings.Read`, `Meetings.Modify`, `Meetings.Delete`
Devices & Contacts (user level access only)	`ContactList.Create`, `ContactList.Read`, `ContactList.Modify`, `ContactList.Delete`

The authorization process

When using private Script Tokens, there is no need for an authorization process. Access to the account/company data through the TeamViewer API is defined when creating the token. The data that is accessed is the account or company data of the user creating the token. Note that a Script Token is still valid after changing the user's password.

For public apps, the case is different. Because these applications can be used by other TeamViewer users, access to their data is controlled via OAuth 2.0. We distinguish between application with access to user level data and company level data.

User level access

If a user starts an app that requires user level access for the first time, TeamViewer will ask the user to grant a set of permissions to the app. This set of permissions was specified when creating the application. The permissions are checked against the rights of the current user. If the application asks for permissions that exceed the rights of the user (e. g. the application wants to edit connection report entries whereas the user is only allowed to view them), the permissions in question are highlighted and a warning is displayed that some parts of the application may not behave as intended because of the lacking user rights.

In any case, the user may either choose to deny or to grant access to the application. If access is granted, the app can access the user's data, as long as the user's permissions allow. If user rights are changed later, the application may be able to access more data.

Authorization Endpoint

GET login.teamviewer.com/oauth2/authorize

Requests to the authorization endpoint must be made via HTTPS. The authorization endpoint is used to interact with the resource owner and therefore must be viewed in a browser.
Security best practice is to open the authorization endpoint in a popup with address bar visible.

Parameters

- `response_type` – Must be code.
- `client_id` – Client ID, a unique string that identifies the application.
- `redirect_uri` – URI of the redirection endpoint. The client is redirected to this URI once access has been granted. The value of this parameter must match the registered redirect URI.
- `state` (optional) – Can be set if needed, and will be returned to the callback URI if it was set here.
- `display` – Must be popup.

Redirection Endpoint

The client is redirected to the redirection endpoint, which is specified when creating an application in Management Console, after the interaction with the authorization endpoint is completed. Values added to the `redirect_uri`:

- `code` – Code that can be used to get an access token.
- `state` (optional) – same as the one provided as parameter.

Description

Requests an authorization code from the server. This code is only valid for 10 minutes and should be used to obtain an access token. This is the only function that should not be called directly from a 3rd party application but it should be opened in a browser where the user can grant access to the 3rd party application.

Example

```
GET login.teamviewer.comoauth2/authorize?re-
sponse_type=code&client_id=12333-133Ea4Hdf3e9ec0543fX&re-
direct_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

POST /api/v1/oauth2/token (token endpoint)

Parameters

Parameters must be inside the body of the request and encoded with the "`application/x-www-form-urlencoded`" format. This is the only exception where the body is not JSON or XML. There are two different requests for this URI, one to **retrieve an access token** using an authorization code and one using a **refresh token**.

Authorization-Code Grant:

- `grant_type` – Must be `authorization_code`.
- `code` – Authorization code acquired from the /oauth2/authorize page.
- `redirect_uri`– Must be the same value as in the previous call to /oauth2/authorize.
- `client_id` – Client ID, a unique string that identifies the application.
- `client_secret` – The client secret, which is known only to the creator of the application.

Refresh-Token:

- `grant_type` – Must be `refresh_token`.
- `refresh_token` – Refresh-token from a previous call.
- `client_id` – Client ID, a unique string that identifies the application.
- `client_secret` - The client secret, a unique string known only to the creator of the application.

Return values

- `access_token` – Access token to use with all further API calls.
- `token_type` – Authentication-method used for this access token, currently only `bearer` is used.
- `expires_in` – Time in seconds until the access token expires and needs to be refreshed.

- `refresh_token` – Refresh-Token that needs to be used to get a new access token when the old access token expires. Requesting a new access token will also create a new refresh token. The refresh token becomes invalid after use or if the access token is revoked.

Description

Requests a new access token, either by using the code from a previous authorization step or by using an existing refresh token. Access tokens have a limited lifetime of 1 day. The response always has the Cache-Control and Pragma fields (see example below). Refresh tokens can only be used once. For this request the Content-Type header should be set to `application/x-www-form-urlencoded` (as per OAuth 2 specification), however JSON/XML also works.

Example

Request (Authorization code grant):

```
POST /api/v1/oauth2/token HTTP/1.1
    Host: webapi.teamviewer.com
    Content-Type: application/x-www-form-urlencoded

grant_type=authoriza-
tion_code&code=SplxlOBeZQQYbYS6WxSb&redi-
rect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb&cli-
ent_id=12333-133Ea4Hdf3e9ec0543fX
```

Response (Authorization code grant):

```
HTTP/1.1 200 OK
    Content-Type: application/json
    Cache-Control: no-store
    Pragma: no-cache

 {"access_token":"54213-2YotnFZFEjr1zCsicMWp",
  "token_type":"bearer",
  "expires_in":3600,
  "refresh_token":"12854-zv3JOkF0XG5Qx2TlKWIA"}
```

Requests using the access token

All API requests need to include the "Authorization" header if the API function requires an access token.

Example

```
GET /api/v1/users HTTP/1.1
  Host: webapi.teamviewer.com
  Authorization: Bearer 54213-2YotnFZFEjr1zCsicMWp
```

All examples in the following sections will have this header omitted but if an access token is required the `Authorization` header field needs to be added to the request.

If no access token is given in the header, or the access token is past its expiration date, the return will have a WWW-Authenticate header field.

Response for no access token, but access token required:

```
HTTP/1.1 401 Unauthorized
  WWW-Authenticate: Bearer
```

Response for expired access token:

```
HTTP/1.1 401 Unauthorized
  WWW-Authenticate: Bearer

{ "error" : "token_expired",
  "error_code" : 1,
  "error_description" : "The access token expired" }
```

POST /api/v1/oauth2/revoke

Parameters

None

Return values

None

Description

Revokes an access token that was created using oauth. The access token must be included in the header Authorization field. After revoking it, it and its attached refresh token cannot be used any longer.

Example

Request:

```
POST /api/v1/oauth2/revoke HTTP/1.1
  Host: https://webapi.teamviewer.com
  Authorization: Bearer 54213-2YotnFZFEjr1zCsicMWp
```

Response:

```
HTTP/1.1 200 OK
```

API Functions

Ping

GET /api/v1/ping

Parameters

None

Return values

- `token_valid` – Is set to `true` if the provided access token is OK and the message is signed correctly. In all other cases, the value is set to `false`.

Authentication

Access tokens are optional but will be verified if provided. Scope: `None required`.

Description

This function can be used to check if the API is available. It can also be used to verify if the token is valid.

Example

Request:

```
GET /api/v1/ping
```

Response:

```
HTTP/1.1 200 OK
  Content-Type: application/json

{"token_valid":false}
```

Meetings

GET /api/v1/meetings/blizz (list all scheduled meetings)

Parameters

- `from_date` (optional) – First start date for all listed meetings. Date is included in the filter. Only the date counts. If a time is provided in the parameter it will be ignored.
- `to_date` (optional) – Last start date for all listed meetings. Date is included in the filter. Only the date counts. If a time is provided in the parameter it will be ignored.

Return values

- `id` – The unique meeting ID.
- `subject` – The subject of the meeting.
- `start` – The start date and time of the meeting.
- `end` – The end date and time for the meeting.
- `password` (optional) – The meeting password. Omitted if no password is set.
- `participant_web_link` – A web link to join the meeting.

Authentication

User access token. Scope: `Meetings.Read`.

Description

Lists all scheduled meetings for the account associated with the authentication token. The list can be filtered with additional parameters. This data is the same as when using `GET /meetings/<mID>` for each of these users.

Example

Request

```
GET /api/v1/meetings/blizz
```

Response

```
HTTP/1.1 200 OK
  Content-Type: application/json

{
    "meetings": [
        {
            "id": "m12-345-678",
            "subject": "Blizz API Webinar",
            "start": "2017-11-25T14:00:00Z",
            "end": "2017-11-25T15:00:00Z",
            "password": "1234",
            "participant_web_link":
"https://go.blizz.com/m12345678"
        },
        {
            "id": "m98-765-432",
            "subject": "API Next Steps",
            "start": " 2017-11-25T14:00:00Z",
            "end": "2017-11-25T15:00:00Z ",
            "participant_web_link":
"https://go.blizz.com/m98765432"
        }
    ]
}
```

GET /api/v1/meetings/blizz/<mID> (get details of a meeting)

Parameters

None

Return values

- `id` – The unique meeting ID.
- `subject` – The subject of the meeting.
- `start` – The start date and time of the meeting.
- `end` – The end date and time for the meeting.

- `password` (optional) – The meeting password. Omitted if no password is set.
- `participant_web_link` – A web link to join the meeting.

Authentication

User access token. Scope: `Meetings.Read`.

Description

Retrieve the details of one single meeting.

Example

Request:

```
GET /api/v1/meetings/blizz/m12-345-678
```

Response:

```
HTTP/1.1 200 OK
  Content-Type: application/json

{
    "id": "m12-345-678",
    "subject": "Blizz API Webinar",
    "start": "2017-11-25T14:00:00Z",
    "end": "2017-11-25T15:00:00Z",
    "password": "1234",
    "participant_web_link":
"https://go.blizz.com/m12345678"
}
```

POST /api/v1/meetings/blizz (create a new meeting)

Parameters

- `subject` – Subject of the meeting.
- `start` – Start date and time.
- `end` – End date and time.
- `password` (optional) – A password that participants must enter to join the meeting.

Return values

- `id` – The unique meeting ID.
- `subject` (optional) – Subject of the meeting. Omitted for instant meetings.
- `start` (optional) – The start date and time of the meeting. Omitted for instant meetings.
- `end` (optional) – The end date and time for the meeting. Omitted for instant meetings.
- `password` (optional) – The meeting password. Omitted if no password is set.
- `participant_web_link` – A web link to join the meeting.

Authentication

User access token. Scope: `Meetings.Create`.

Description

Creates a new meeting. The response contains the values for the new meeting.

Example for a scheduled meeting

Request

```
POST /api/v1/meetings/blizz
    Content-Type: application/json
{
    "subject": "Blizz API Webinar",
    "start": "2017-11-25T14:00:00Z",
    "end": "2017-11-25T15:00:00Z",
    "password": "1234",
}
```

Response

```
HTTP/1.1 200 OK
    Content-Type: application/json

{
    "id": "m12-345-678",
    "subject": "Blizz API Webinar",
    "start": "2017-11-25T14:00:00Z",
    "end": "2017-11-25T15:00:00Z",
    "password": "1234",
    "participant_web_link":
"https://go.blizz.com/m12345678"
}
```

Errors

HTTP response codes

- `200` – OK: Used for successful GET, POST and DELETE.
- `204` – No Content: Used for PUT to indicate that the update succeeded, but no content is included in the response.
- `400` – Bad Request: One or more parameters for this function is either missing, invalid or unknown. Details should be included in the returned JSON.
- `401` – Unauthorized: Access token not valid (expired, revoked, ...) or not included in the header.
- `403` – Forbidden / Rate Limit Reached: IP blocked or rate limit reached.
- `500` – Internal Server Error: Some (unexpected) error on the server. The same request should work if the server works as intended.

JSON error responses

If there is an error while processing a request, the API server returns a 4xx/5xx HTTP status code with a JSON in the body with the following parameters:

- `error` – A short string describing the category of error.
- `error_description` – A longer string containing a human readable error message.
- `error_code` – A number that is unique for each type of error.
- `error_signature` (optional) – A number that we can use to find the log entry if there is one. This should be unique for every time an error happens. The parameter may be omitted if there was nothing logged.

Valid values for the error field are:

- `invalid_request` – The request is missing a required parameter, includes an unsupported parameter or parameter value, repeats the same parameter, uses more than one method for including an access token, or is otherwise malformed. Should be used with HTTP response code 400.
- `invalid_token` – The access token provided is revoked, malformed, or invalid. Should be used with HTTP response code 401 (Unauthorized).
- `internal_error` – There was an error while processing the request. The error was caused by an error on our servers that should not happen and can indicate some problems at our end. Error code and signature can be used to debug the error. Should be used with HTTP status code 500 (Internal Server Error).
- `blocked` – The request was blocked. That should only happen when the IP was blocked.
- `rate_limit_reached` – Too many calls to a single function with the same access token.
- `token_expired` – Access token is expired. A new access token needs to be requested.
- `invalid_client` – Client ID was invalid.
- `email_in_use` – Returned during account creation or when changing the email if the email is already used by another account.

`invalid_request` and `invalid_token` are taken from http://self-issued.info/docs/draft-ietf-oauth-v2-bearer.html#resource-error-codes (with the exception that they are not included in the WWW-Authenticate header field but the returned JSON).

Contact

If you have questions or feedback, please visit
https://www.teamviewer.com/ticket.