API Docs

If you end up using the API, I'd love to know about how you're using it. Tell me about it by making a GitHub issue or emailing me :)

Automating submissions is *not* allowed.

The API and database follow this license (https://github.com/ajayyy/SponsorBlock/wiki/Database-and-API-License) unless you have explicit permission. Attribution Template (https://gist.github.com/ajayyy/4b27dfc66e33941a45aeaadccb51de71)

Public API available at https://sponsor.ajay.app.

Database download: https://sponsor.ajay.app/database

Database Mirror (30 min update time, provided by Lartza): https://sb.ltn.fi/database/

Database Mirror (10 min update time, provided by blab): https://mirror.sb.mchang.xyz/

Database Mirror (90 min update time, provided by mini_bomba): https://sb.minibomba.pro/mirror/

Note: Database mirrors listed above do ***NOT*** provide an API for the extensions. Setting the server address to any of these URLs will result in no segments or video details being shown.

DeArrow-only read-only API Mirror (90 min update time, provided by mini_bomba): https://dearrow.minibomba.pro/sbserver/

If you are looking for DeArrow API Docs, see its page.

Libraries: Node.js (https://www.npmjs.com/package/sponsorblock-api), Python (https://github.com/wasi-master/sponsorblock.py), Rust (https://crates.io/crates/sponsor-block), Go (https://github.com/porjo/sponsorblockgo), Kotlin (https://github.com/zt64/sponsorblock-kt)

Webhook Docs (https://github.com/ajayyy/SponsorBlock/wiki/Webhooks) | OpenAPI Docs (https://github.com/mchangrh/sb-openapi)

Contents

GET `/api/skipSegments`

Get segments for a video.

Input (URL Parameters):

```
{
  videoID: string,

  category: string, // Optional, defaults to "sponsor", can be repeated for multiple categories. [1]
  // OR
  categories: string[], // Optional [1]

  requiredSegment: string, // Segment UUID to require to be retrieved, even if they don't meet the minimum
vote threshold. Can be repeated for multiple segments.
  // OR
  requiredSegments: string[], // Optional, array of required segment UUIDs

  actionType: string // Optional, default skip. Can be repeated for multiple types. [3]
  // OR
  actionTypes: string[] // Optional, array of action types

  service: string, // Optional, default is 'YouTube' [2]
}
```

References: [1] [2] [3]

Response:

```
[{ // Array of this object
   segment: float[], //[0, 15.23] start and end time in seconds
   UUID: string,
   category: string, // [1]
   videoDuration: float // Duration of video when submission occurred (to be used to determine when a
submission is out of date). 0 when unknown. +- 1 second
   actionType: string, // [3]
   locked: int, // if submission is locked
   votes: int, // Votes on segment
   description: string, // title for chapters, empty string for other segments
}]
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

404: Not Found

GET `/api/skipSegments/:sha256HashPrefix`

Get segments for a video with extra privacy

`sha256HashPrefix` is a hash of the YouTube `videoID`. It should be the first 4 - 32 characters (4 is recommended). This provides extra privacy by potentially finding more than just the video you are looking for since the server will not know exactly what video you are looking for.

Input (URL Parameters):

```
{
  prefix: string, // Can be used instead of path

  category: string, // Optional, defaults to "sponsor", can be repeated for multiple categories. [1]
  // OR
  categories: string[], // Optional, array of categories [1]

  requiredSegment: string, // Segment UUID to require to be retrieved, even if they don't meet the minimum
vote threshold. Can be repeated for multiple segments.
  // OR
  requiredSegments: string[], // Optional, array of required segment UUIDs
  actionType: string // Optional, default skip. Can be repeated for multiple types. [3]
  // OR
  actionTypes: string[] // Optional, array of action types [3]

  service: string // Optional, default is 'YouTube'. [2]

  trimUUIDs: boolean // Optional, if true then UUIDs in the response will be shorter to reduce bandwidth
}
```

References: [1] [2] [3]

Response

```
[{ // Array of this object
   videoID: string,
   segments: [{ // Array of this object
       segment: float[], // [0, 15.23] start and end time in seconds
       UUID: string,
       category: string, [1]
       actionType: string, // [1]
       locked: int, // if segment is locked
       votes: int, // votes on segment
       videoDuration: int, // Duration of video when submissions occurred
       description: string // title for chapters, empty string for other segments
   }]
}]
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

404: Not Found

POST `/api/skipSegments`

> **Automating submissions is not allowed.**
> Please see the Automating submissions page for more information.

Create a segment on a video

Input (URL Parameters):

```
{
  videoID: string,
  startTime: float,
  endTime: float,
  category: string, // [1]
  userID: string, // This should be a randomly generated 30 char string stored locally (not the public one)
  userAgent: string, // "Name of Client/Version" or "[BOT] Name of Bot/Version" ex. "Chromium/1.0.0"
  service: string, // Optional, default is 'YouTube'. [2]
  videoDuration: float, // Optional, duration of video, will attempt to retrieve from the YouTube API if
missing (to be used to determine when a submission is out of date)
  actionType: string // Optional, default is "skip". [3]
  description: string // Chapter title for chapters, must be an empty string or not present for other
segment types
}
```

References: [1] [2] [3]

OR

Input (JSON Body):

```
{
  videoID: string,
  userID: string, // This should be a randomly generated 30 character string stored locally (not the public one)
  userAgent: string, // "Name of Client/Version" or "[BOT] Name of Bot/Version" ex. "Chromium/1.0.0"
  service: string, // Optional, default is 'YouTube'.[2]
  videoDuration: float, // Optional, duration of video, will attempt to retrieve from the YouTube API if missing (to be used to determine when a submission is out of date)

  segments: [{ // Array of this object
     segment: float[], // [0, 15.23] start and end time in seconds
     category: string, // [1]
     actionType: string // Optional, defaults to "skip". [3]
     description: string // Chapter title for chapters, must be an empty string or not present for other segment types
  }]
}
```

References: [1] [2] [3]

Response:

```
{ // array of this object
  UUID: string, // UUID of submitted segment
  category: string, // submitted category [1]
  segment: float[] // start and end time of submitted segment
}[]
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Rejected by auto moderator (Reason will be sent in the response)

429: Rate Limit (Too many for the same user or IP)

409: Duplicate

POST `/api/voteOnSponsorTime`

Vote on a segment or vote to change the category of the segment.

Creators of the segment and VIPs can remove the segment or change the category with only one vote.

VIP voting notes [Expand]

Input: Normal Vote (URL Parameters):

```
{
  UUID: string, // UUID of the segment being voted on
  videoID: string // Optional, only required if the provided UUID is a trimmed UUID (to reduce bandwidth, skipSegments API can return trimmed UUIDs)
  userID: string, // Local userID
  type: int // 0 for downvote, 1 for upvote, 20 to undo vote
}
```

OR

Input: Category Vote (URL Parameters):

```
{
  UUID: string, // UUID of the segment being voted on
  userID: string, // Local userID
  category: string // Category to change this submission to [1]
}
```

References: [1]

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Reason given in request (moderation)

POST `/api/viewedVideoSponsorTime`

Add view to segment

Input (URL Parameters):

```
{
  UUID: string // UUID of segment viewed
}
```

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

GET `/api/userInfo`

Get information about a user

Input (URL Parameters):

```
{
  userID: string // local UserID
  // OR
  publicUserID: string // Public userID

  values: string[] // Optional, Values to get from userInfo
    // default values are
    // ["userID", "userName", "minutesSaved", "segmentCount", "ignoredSegmentCount",
    // "viewCount", "ignoredViewCount", "warnings", "warningReason", "reputation",
    // "vip", "lastSegmentID"]
  // OR
  value: string // Optional, Value to get from userInfo, can be repeated for multiple values
}
```

Response:

```
{
  userID: string, // Public userID
  userName: string, // Public userID if not set
  minutesSaved: float, // Minutes saved
  segmentCount: int, // Total number of segments excluding ignored/ hidden segments
  ignoredSegmentCount: int, // Total number of ignored/ hidden segments
  viewCount: int, // Total number of views excluding view on ignored/ hidden segments
  ignoredViewCount: int, // Total number of view on ignored/ hidden segments
  warnings: int, // Currently enabled warnings
  reputation: float,
  vip: int, // VIP status
  lastSegmentID: string, // UUID of last submitted segment
  permissions: { // Can the user submit segments of this category?
    category: boolean // [1]
  }
}
```

References: [1]

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

GET `/api/userStats`

Get stats for a user

Input (URL Parameters):

```
{
  userID: string // local UserID
  // OR
  publicUserID: string // Public userID

  fetchCategoryStats: boolean // default false, display category stats
  fetchActionTypeStats: boolean // default false, display type stats
}
```

Response:

```
{
  userID: string // hashed userID
  userName: string // userName
  overallStats: {
    minutesSaved: integer // same as userInfo
    segmentCount: integer // same as userInfo
  }
  // IF CHOSEN
  categoryCount: { // # of segments per category
    sponsor: integer
    intro: integer
    outro: integer
    interaction: integer
    selfpromo: integer
    music_offtopic: integer
    preview: integer
    poi_highlight: integer
    filler: integer,
    exclusive_access: integer,
    chapter: integer
  }
  // IF CHOSEN
  actionTypeCount: { // # of segments per type
    skip: integer,
    mute: integer,
    full: integer,
    poi: integer,
    chapter: integer
  }
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

GET `/api/getViewsForUser`

Get the number of views a user has on all their segments

Input (URL Parameters):

```
{
  userID: string // local UserID
}
```

Response:

```
{
  viewCount: int
}
```

Error codes:

404: Not Found

GET `/api/getSavedTimeForUser`

Get the total time saved from all the user's segments

Input (URL Parameters):

```
{
  userID: string // Local userID
}
```

Response:

```
{
  timeSaved: float // In minutes
}
```

Error codes:

404: Not Found

POST `/api/setUsername`

Set a username for a userID

Input (URL Parameters): Setting username for self

```
{
  userID: string, // Local userID
  username: string, // Optional
}
```

OR

Input (URL Parameters): Setting username as **admin**

```
{
  userID: string, // Public userID
  username: string, // Optional
  adminUserID: string // Admin's local userID
}
```

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not an admin)

GET `/api/getUsername`

Get current username

Input (URL Parameters):

```
{
  userID: string // Local userID
}
```

Response:

```
{
  userName: string // Public userID if no username has been set
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

GET `/api/segmentInfo`

Get information about segments

Input (URL Parameters):

```
{
  // Only the first 10 entries will be processed
  UUID: string, // Can be repeated for multiple segments
  // OR
  UUIDs: string[] // Looks like ["a...0", "b...1"]
}
```

Response:

```
[{ // Array of this object
  videoID: string,
  startTime: float,
  endTime: float,
  votes: int,
  locked: int, // Status of lock - If upvoted by a VIP, the segment is locked
  UUID: string,
  userID: string, // PublicID of submitter
  timeSubmitted: int,
  views: int, // Number of reported views on the segment
  category: string, // [1]
  service: string, // [2]
  actionType: string, // [3]
  videoDuration: int,
  hidden: int, // If the segment has 2 downvotes or was downvoted by a VIP
  reputation: int, // Reputation of submitter at time of submission
  shadowHidden: int, // If the submitter is shadowbanned
  hashedVideoID: string, // sha256 hash of the videoID
  userAgent: string, // userAgent of the submitter,
  description: string // title for chapters, empty string for other segments
}]
```

References: [1] [2] [3]

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

404: Not Found

GET `/api/userID`

List all users matching the username search

Input (URL Parameters):

```
{
  username: string, // search string for username
    // case sensitive
    // minimum for non-exact search is 3 characters, maximum is 64 characters
  exact: boolean // searches for exact username with no wildcard at end
}
```

Response:

```
[{ // Array of this object - maximum 10 results
  userName: string,
  userID: string
}]
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible) or exceed the character limits

404: Not Found

GET `/api/lockCategories`

Get locked categories for a video

Input (URL Parameters):

```
{
  videoID: string,
  actionTypes: string[] // [3]
    // default [skip, mute]
}
```

Response:

```
{
  categories: string[], // [1]
  reason: string, // Specified reason for the lock
    // Only the most recent reason will be returned
  actionTypes: string[] // [3]
}
```

References: [1] [3]

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

404: Not Found

GET `/api/lockCategories/:sha256HashPrefix`

Get locked categories for a video with extra privacy

`sha256HashPrefix` is a hash of the YouTube `videoID`. It should be the first 4 - 32 characters (4 is recommended). This provides extra privacy by potentially finding more than just the video you are looking for. This makes the server not know exactly what video you are looking for.

Input (URL Parameters):

```
{
  prefix: sha256HashPrefix // Optional if not sent through path
}
```

Response:

```
[{ // Array of this object
   videoID: string,
   hash: string, // The full hash of the videoID
   categories": string[], // [1]
   reason: string // Specified reason for the lock
}]
```

References: [1]

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

404: Not Found

GET `/api/lockReason`

Get reason for lock(s)

Input (URL Parameters):

```
{
  videoID: string

  // Categories to get reasons for, defaults to all [1]
  category: string
  // OR
  categories: string[],
  actionTypes: string[] // [3]
}
```

References: [1] [3]

Response:

```
[{ // Array of this object
  category: string, // category [1]
  locked: integer, // status of lock
  reason: string, // reason for lock
  userID: string, // publicID of locking VIP
  userName: string // username of locking VIP
}]
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

GET `/api/searchSegments`

Get all segments of a video based on specified filters. Note: It is suggested that you don't use this for knowing which segments to skip on your client, as thresholds and values that determine which segments are the best change over time. Using /api/skipSegments (https://wiki.sponsor.ajay.app/index.php/API_Docs#GET_.2Fapi.2FskipSegments) ensures that you will always get the best segments.

Input (URL Parameters) **OR** (JSON Body):

```
{
  // See skipSegments
  videoID: string

  category: string // [1]
  // OR
  categories: string[]

  actionType: string // [3]
  // OR
  actionTypes: string[]

  service: string // [2]
  // End SkipSegments

  page: int // Page to start from (default 0)
  // Vote/ view thresholds, inclusive, default includes all segments
  minVotes: int
  maxVotes: int

  minViews: int
  maxViews: int

  // Default true - if false, don't show segments that match type
  locked: boolean
  hidden: boolean
  ignored: boolean // hidden or below vote threshold
}
```

References: [1] [2] [3]

Response:

```
{
  segmentCount: int, // Total number of segments matching query
  page: int, // Page number
  segments: [{ // Array of this object, max 10
    // see segmentInfo
    UUID: string,
    timeSubmitted: int,
    startTime: int,
    endTime: int,
    category: string, // [1]
    actionType: string, // [3]
    votes: int,
    views: int,
    locked: int,
    hidden: int,
    shadowHidden: int,
    userID: string, // UUID of submitter
    description: string // title for chapters, empty string for other segments
  }]
}
```

References: [1] [3]

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

404: Not Found

GET `/api/status/:value`

Get status of server

Input: (URL path)

Can be any key value in response, requests without path will return all values.

Response:

```
{
  uptime: int, // Uptime of server in seconds
  commit: string, // Full SHA hash of latest git commit, development or test
  db: int, // Current database version
  startTime: int, // Unix time (miliseconds) that request was received
  processTime: int, // Delay between DB request made and response received (miliseconds)
  redisProcessTime: int, // Delay between redis request made and response received (miliseconds)
  loadavg: int[], // 5 and 15 minute loadavg
  statusRequests: int, // number of /status requests in the last minute
  hostname: string // hostname of current server
}
```

Error codes:

404: Not Found

Stats Calls

GET `/api/getTopUsers`

This endpoint is currently disabled and will always return 404.

Get top submitters

Input (URL Parameters):

```
{
  sortType: int
    // 0 for by minutes saved
    // 1 for by view count
    // 2 for by total submissions
}
```

Response:

```
{
  userNames: string[],
  viewCounts: int[],
  totalSubmissions: int[],
  minutesSaved: float[]
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

GET `/api/getTopCategoryUsers`

This endpoint is currently disabled and will always return 404.

Get top submitters by category

Input (URL Parameters):

```
{
  sortType: int,
    // 0 for by minutes saved
    // 1 for by view count
    // 2 for by total submissions
  category: string // category to fetch stats for
}
```

Response:

```
{
  userNames: string[],
  viewCounts: int[],
  totalSubmissions: int[],
  minutesSaved: float[]
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

GET `/api/getTotalStats`

Get total stats

Input (URL Parameters):

```
{
  countContributingUsers: boolean // Optional, default false
}
```

Response:

```
{
  userCount: int, // Only if countContributingUsers was true
  activeUsers: int, // Sum of public install stats from Chrome webstore and Firefox addons store
  apiUsers: int, // 48-hour active API users (https://github.com/ajayyy/PrivacyUserCount)
  viewCount: int,
  totalSubmissions: int,
  minutesSaved: float
}
```

Error codes:

None

GET `/api/getDaysSavedFormatted`

Get days saved by all skips

Input:

```
{
  Nothing
}
```

Response:

```
{
  daysSaved: float (2 decimal places)
}
```

Error codes:

None

VIP Calls

These can only be called by the users added to the VIP table.

GET `/api/isUserVIP`

If the user is a VIP

Input (URL Parameters):

```
{
  userID: string, // Local userID
}
```

Response:

```
{
  hashedUserID: string, // Public userID
  vip: boolean
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

POST `/api/lockCategories`

Create a category lock on the video, disallowing further submissions for that category

Input (Request Body):

```
{
  videoID: string,
  userID: string, // Local userID
  categories: string[], // [1]
  actionTypes: string[], // [3]
  reason: string, // Reason for lock
}
```

References: [1] [3]

Response:

```
{
  submitted: string[], // categories [1]
  submittedValues: {
    actionType: string, // [3]
    category: string // [1]
  } // array of this object
}
```

References: [1] [3]

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not a VIP)

DELETE `/api/lockCategories`

Delete existing category locks on that video

Input (Request Body):

```
{
  videoID: string,
  userID: string, // Local userID
  categories: string[] // [1]
}
```

References: [1]

Response:

```
{
  message: "Removed lock categories entries for video videoID"
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not a VIP)

POST `/api/shadowBanUser`

Shadow banned submissions are hidden for everyone but the IP that originally submitted it. Shadow banning a user shadow bans all future submissions.

User can be re-shadowbanned if segments were not previously hidden

Input (URL Parameters):

```
{
  userID: string, // Public userID of the user you want to shadowBan
  adminUserID: string, // Local userID of VIP or Admin
  enabled: boolean, // Optional, default true, true to ban and false to unban
  unHideOldSubmissions: boolean, // Optional, depends on the enabled parameter, should all previous
submissions be (un)hidden as well?
  categories: string // Optional, defaults to all categories, in the format "["sponsor", "selfpromo"]"
etc...
}
```

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not a VIP)

409: Duplicate (User already shadowbanned & unHideOldSubmissions not changed)

POST `/api/warnUser`

Temporary ban that shows a warning asking them to contact us.

If a user is re-warned but there is still a non-expired warning, it is reenabled

Input (Request Body):

```
 {
  issuerUserID: string, // Issuer userID (Local userID)
  userID: string, // Public userID you are warning
  reason: string, // Optional
  enabled: boolean // Optional, default true
}
```

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not a VIP)

409: User already warned

POST `/api/clearCache`

Clear redis cache for video.

Input (Request Body):

```
{
  userID: string, // Local userID
  videoID: string
}
```

Response:

```
{
  Cache cleared on video videoID (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not a VIP)

POST `/api/purgeAllSegments`

Hide all segments on a video without affecting submitters' reputation

Input (Request Body):

```
{
  userID: string, // Local userID
  videoID: string,
  service: string // Service of video, defaults to YouTube
}
```

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not a VIP)

POST `/api/segmentShift`

Shift all segments on a video

Input (Request Body):

```
{
  videoID: string,
  userID: string, // Local userID
  startTime: float,
  endTime: float
}
```

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not a VIP)

POST `/api/addUserAsTempVIP`

Add temporary 24 Hour Channel VIP

A user cannot be a VIP of multiple channels, the most recent channel will take precedence and override

Input (URL Parameters):

```
{
  userID: string, // User to grant temp VIP to
  adminUserID: string, // Local userID of existing VIP
  channelVideoID: string, // videoID of channel to grant VIP on
  enabled: string // default "true" Enable or disable VIP status
}
```

Response:

```
{
  Temp VIP added on channel channelName (status code 200)
  Temp VIP removed (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not a VIP)

404: Not Found (No channel found for videoID)

409: Duplicate (User is already a permanent VIP)

POST `/api/feature`

Add or remove user features

Input (Request Body):

```
{
  userID: string, // User to add or remove features from
  adminUserID: string, // Local userID of existing VIP
  feature: int, // [4]
  enabled: boolean // default "true" Enable or disable feature
}
```

References: [4]

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible or the feature does not exist)

403: Unauthorized (You are not a VIP)

Admin Calls

These can only be called by the server administrator, set in the config.

POST `/api/addUserAsVIP`

VIPs have extra privileges and their votes count more.

Input (URL Parameters):

```
{
  userID: string, // Public userID of the user you want to add to the VIP list
  adminUserID: string, // Admin's local userID
  enabled: boolean // Optional, to be able to add and remove users (default: false)
}
```

Response:

```
{
  Nothing (status code 200)
}
```

Error codes:

400: Bad Request (Your inputs are wrong/impossible)

403: Unauthorized (You are not an admin)

Legacy API

https://github.com/ajayyy/SponsorBlock/wiki/Legacy-API

Local userID vs Public userID

The local userID should be a randomly generated and saved client side and must be 32 characters (or more). If it is not 32 characters or more, you will not be able to vote or submit. The public userID is what is used as an identifier in the database. This is the local userID with a SHA 256 hash 5000 times.

References

1. See Types for full list of possible categories. To get multiple, create an array with the format `["sponsor", intro"]`.
2. Service to get segments for. See Types for supported services
3. Action Types: See Types for possible values. Select multiple with the format `["skip","mute]`
4. See Types for a full list of possible user features.

Retrieved from "https://wiki.sponsor.ajay.app/index.php?title=API_Docs&oldid=3893"

This page was last edited on 21 February 2025, at 03:06.