Grindr API v3 documentation

Grindr's service is comprised of:

1. A REST API endpoint, at https://grindr.mobi
2. A regular HTTP CDN, at http://cdns.grindr.com
3. An XMPP server at chat.grindr.com:5222

All API calls to the REST endpoint must be sent with the HTTP user-agent set to something like `grindr3/3.0.1.4529;4529;Unknown;Android 4.4.4`. The REST endpoint will fail requests with HTTP/412 if it doesn't like the user-agent.

Updating dynamic configuration

Some configuration is pulled from the server, presumably so Grindr can change it at any time. In pratice, these don't seem to change often, so you can consider caching them.

Service endpoints

The official app first downloads a list of service endpoints. Issue an unauthenticated GET request to `https://grindr.mobi/v3/bootstrap` to get a list of server names you'll use. Example response:

```
{
    "configuration": {
        "app": {
            "broadcastsToDisplay": 3,
            "cascadeTtl": 420000,
            "maxBlocksPerDay": 10,
            "maxCascadePages": 1,
            "maxFavoritesPerDay": 10,
            "maxPhotosAtOnce": 1,
            "profileQuantityPerCascadePage": 100,
            "profileTtl": 120000,
            "sendSavedPhrases": {
                "enabled": false
            }
        },
        "serverTime": <integer with Unix timestamp in milliseconds>,
        "serverTimeMillis": <integer with Unix timestamp in milliseconds>,
        "user": {
            "eligibleForFreeTrial": true,
            "subscription": {
                "remainingTimeMillis": 0,
                "subscriptionId": null
            },
            "userRole": "Free"
        }
    },
```

```
    "services": {
        "api": [
            {
                "domain": "g3-beta.grindr.com",
                "host": "g3-beta.grindr.com",
                "port": 443,
                "protocol": "https"
            }
        ],
        "authenticationWeb": [
            {
                "host": "neo-account.grindr.com",
                "path": "",
                "port": 443,
                "protocol": "https"
            }
        ],
        "chat": [
            {
                "domain": "chat.grindr.com",
                "host": "chat.grindr.com",
                "port": 5222,
                "protocol": "xmpp"
            }
        ],
        "media": [
            {
                "host": "cdns.grindr.com",
                "path": "",
                "port": 80,
                "protocol": "http"
            }
        ],
        "upload": [
            {
                "host": "g3-beta-upload.grindr.com",
                "path": "",
                "port": 443,
                "protocol": "https"
            }
        ],
        "web": [
            {
                "host": "neo-store.grindr.com",
                "path": "/",
                "port": 443,
                "protocol": "https"
            }
        ]
    }
}
```

It's unclear what the v3 client uses the `services` table for, as the v3 client hardcodes the API server to `grindr.mobi`. In v2, the `web` service was used for upsells and the `authenticationWeb` was used for account creation, but in v3, these are all REST APIs on the `grindr.mobi` server now. Possibly the `services` table isn't used by the v3 client.

The `configuration.app` section seems to correspond to premium client-side features. `serverTime` and `serverTimeMillis` are the same integer; possibly the redundancy is there for compatiblity with various clients.

The `configuration` section was present in v2, but was initially missing from the 3.0.x series. It was added back in either 3.0.2 or 3.0.3.

Managed fields

Certain enums and flags are downloaded from the server. For example, the list of Ethnicities shown on a user's profile is a managed field. Grindr can add a new Ethnicity at any time without a client update.

Issue an unauthenticated GET request to `https://grindr.mobi/v3/managed-fields` to get a list of fields. Example response (partly redacted to conserve space):

```
{
    "fields": {
        "lookingFor":[{"fieldId":2,"name":"Chat"},{"fieldId":3,"name":"Dates"}, .
. . etc],
        "relationshipStatus":[{"fieldId":4,"name":"Committed"},
{"fieldId":2,"name":"Dating"}, . . . etc],
        "bodyType":[{"fieldId":1,"name":"Toned"},{"fieldId":2,"name":"Average"}, .
. . etc],
        "ethnicity":[{"fieldId":1,"name":"Asian"},{"fieldId":2,"name":"Black"}, .
. . etc],
        "grindrTribes":[{"fieldId":1,"name":"Bear"},{"fieldId":2,"name":"Clean-
Cut"},. . . etc],
        "reportReasons":[{"fieldId":1,"name":"Offensive profile image"},
{"fieldId":2,"name":"Offensive profile text"}, . . . etc]
        }
}
```

Most fields are self-explanatory elements on the profile. The exception is `reportReasons`, which is used to report a policy violation to Grindr's moderators.

Authentication

Terminology (not official, just used in this document):

- **Email**. The "username" as far as the user is concerned. It is just an email address.
- **Password**. The password the user selected when creating the account. The user types this in once during first setup, and probably forgets it immediately.

- **Profile Id**. An integer that is the actual username.
- **Authentication Token**. A persistent key that is stored on disk to allow signing in without needing to retype the password.
- **Session Id**. A short-lived token that is used with each HTTP API call.
- **XMPP Token**. A short-lived token that is used to log into the XMPP server.
- **GCM Token**. A Google Cloud Messaging (GCM) token generated by calling [InstanceId.getToken](https://developers.google.com/android/reference/com/google/android/gms/iid/InstanceID.html#getToken(java.lang.String, java.lang.String)).
- **captcha Token** A Google captcha token from their picture puzzle

Creating a new account

Ask the user for the following information:

- Email address
- Password
- Birthday (for age verification, and presumably also better ad targeting)
- Whether the user would like to receive emails from Grindr's marketing department
- Solve a captcha

Issue a POST request to `https://grindr.mobi/v3.1/users` with this payload:

```
{
    "birthday": <timestamp>,
    "captchaToken": <captcha token>,
    "email": "user@example.com",
    "optIn": false,
    "password": "my cool password",
    "token": "<GCM Token; see description below>"
}
```

Encode `birthday` as an integer holding a Unix timestamp multiplied by 1000 (i.e., Unix milliseconds). Note that people who were born before 1970 will have a *negative* birthday. For example, someone born on 1 Jan 1923 00:00:00 UTC would have a birthday of `-1483228800000`.

The `token` is the GCM Token used to authorize the app to deliver push notifications. It appears to be validated by the server, so you'll have to find a way to get an authentic GCM authorization token. Normally, an Android or iOS app can simply call InstanceID.getToken. However, if your platform doesn't have a Google SDK, you're in trouble.

If there's an error creating the account, the server will return an HTTP 4xx error code. For example, if the email account is already signed up for Grindr, the server returns HTTP/409 with this payload:

```
{
    "code": 22,
    "message": "Email id already exists"
}
```

(Unfortunately, this means you can easily out someone as a Grindr user, simply by knowing his email address. This seems like a severe privacy problem.)

If account creation is successful, the server returns HTTP/201 (note not the usual 200) with this response:

```
{
    "authToken": "<the Authentication Token>",
    "profileId": "<the Profile Id>",
    "sessionId": "<a Session Id for immediate use>",
    "xmppToken": "<an XMPP Token for immediate use>"
}
```

Save the Email Address, Authentication Token, and Profile Id to disk. The Authentication Token is a password equivalent, so store it securely.

You can use the Session Id and XMPP Token to immediately begin using the service. You do not need to log in or create a new session; the server has conveniently created a session for you already.

Logging into an existing account without an Authentication Token

If the account already exists, but you don't have an Authentication Token yet, you'll need to get an Authentication Token. Typically, this would be the first time the user runs your app.

POST a request to `https://grindr.mobi/v3/sessions` with this payload:

```
{
    "email": "user@example.com",
    "password": "my cool password",
    "token": "<GCM Token; see discussion above>"
}
```

If successful, the server returns the same response as in the previous section.

Every subsequent session

Assuming you know the Authentication Token and Email Address, you'll just need to create a new session. Issue a POST to `https://grindr.mobi/v3/sessions` with the following payload

```
{
    "authToken": "<the Authentication Token>",
    "email": "user@example.com",
    "token": "<GCM Token; see discussion above>"
}
```

If successful, the server returns this HTTP/200 response:

```
{
    "profileId": "<the user's Profile Id>",
    "sessionId": "<a Session Id>",
    "xmppToken": "<an XMPP Token>"
}
```

Issuing authenticated HTTP requests

By now you should have a Session Id (either from creating a new account, signing in with an email, or creating a session from an Authentication Token). To issue an authenticated HTTP request, add the HTTP header

```
Authorization: Grindr3 <Session Id>
```

That is, take the literal string "Grindr3 " and append the Session Id. That's your `Authorization` header.

Account management

Update the user's public profile

Issue an authenticated PUT to `https://grindr.mobi/v3/me/profile` with payload like this:

```
{
    "aboutMe": "",
    "age": 18,
    "bodyType": 0,
    "displayName": "",
    "ethnicity": 0,
    "grindrTribes": [],
    "height": -1.0,
    "lookingFor": [],
    "relationshipStatus": 0,
    "showAge": false,
    "showDistance": false,
    "socialNetworks": {
        "facebook": {
            "userId": ""
        },
        "instagram": {
            "userId": ""
        },
        "twitter": {
            "userId": ""
        }
    },
```

```
    "weight": -1.0
}
```

`bodyType`, `ethnicity`, `grindrTribes`, `lookingFor`, and `relationshipStatus` are managed fields, as described above. `height` is a number of centimeters, or `-1.0` to hide it entirely. `weight` is a number of grams, or `-1.0` to hide it entirely.

If successful, the server responds with HTTP/200 and an empty JSON object.

Query user preferences

Issue an authenticated GET to `https://grindr.mobi/v3/me/prefs`. Example response:

```
{
    "chatPix": {
        "<hash of image>": {
            "mediaHash": "<hash of image>",
            "timestamp": <integer of unix milliseconds>
        }
    },
    "filters": {},
    "phrases": {},
    "quickyChatPixHash": null,
    "quickyPhraseId": null,
    "settings": {
        "unitSystem": 1
    }
}
```

Note that the image hash is indeed repeated for some reason.

`unitSystem` = 1 means USA Imperial units.

Get system messages

Issue an authenticated GET to `https://grindr.mobi/v3/systemMessages` with no payload.

If successful, the server returns HTTP/200 with a payload that contains a `systemMessages` object with zero or more messages:

```
{
    "systemMessages": [
        {
            "mediaHash": "<Media hash>",
            "message": "Your photo has been rejected.",
            "messageId": <integer>,
            "type": "ProfileImageRejected"
        }
```

```
    ]
}
```

Note that if you're migrating from v2, the return value has changed from a JSON array to a JSON object that wraps an array.

Change user password

Issue an authenticated POST to `https://grindr.mobi/v3/users/update-password` with the payload:

```
{
    "newPassword": "my c00l pa$$word",
    "oldPassword": "my cool password"
}
```

If successful, the server response with HTTP/200 and a new Authentication Token and session:

```
{
    "authToken": "<the Authentication Token>",
    "profileId": "<the Profile Id>",
    "sessionId": "<a Session Id for immediate use>",
    "xmppToken": "<an XMPP Token for immediate use>"
}
```

Replace the Authentication Token you'd saved earlier with the new one. Start using the new Session Id in future authenticated HTTP requests.

Change the email address associated with the account

Issue an authenticated POST to `https://grindr.mobi/v3/users/email` with the payload:

```
{
    "newEmail": "user@example.com",
    "password": "my cool password"
}
```

If the operation fails, the server returns an HTTP/4xx code. For example, if the email address is already in use by a different account, the server returns HTTP/409 with this payload:

```
{
    "code": 22,
    "message": "Email id already exists"
}
```

If successful, the server response with HTTP/200 and a new Authentication Token and session:

```
{
    "authToken": "<the Authentication Token>",
    "profileId": "<the Profile Id>",
    "sessionId": "<a Session Id for immediate use>",
    "xmppToken": "<an XMPP Token for immediate use>"
}
```

Replace the Authentication Token you'd saved earlier with the new one. Start using the new Session Id in future authenticated HTTP requests.

Delete the account

To remove the account from the current device, just securely erase the Authentication Token and any other state you've accumulated.

You can optionally also delete the account *from the server*. The user will lose all his preferences and will not be able to sign in on any device.

Issue an authenticated DELETE to `https://grindr.mobi/v3/me/profile` with no payload. If successful, the server responds with HTTP/200 and an empty JSON object. The account is gone.

Interacting with other users

Get messages that were sent while you were logged out

Issue an authenticated POST to `https://grindr.mobi/v3/me/chat/messages?undelivered=true` with no payload. (Not an empty JSON object -- the Content-Length should be 0 and the Content-Type is absent.)

If successful, the server responds with HTTP/200 and an object containing an array of offline messages:

```
{
    "messages": [
        {
            "body": "Hey what's up?",
            "messageId": "<unique message id>",
            "sourceDisplayName": "<the Display Name from his profile>",
            "sourceProfileId": "<his Profile Id>",
            "targetProfileId": "<your Profile Id>",
            "timestamp": <integer of Unix timestamp, in milliseconds>,
            "type": "text"
        },
        {
            "body": "Hola",
            "messageId": "<another unique message id>",
            "sourceProfileId": "<his Profile Id>",
```

```
            "targetProfileId": "<your Profile Id>",
            "timestamp": <integer of Unix timestamp, in milliseconds>,
            "type": "text"
        }
}
```

Messages are formatted roughly the same as in XMPP chat. Not all messages have a `sourceDisplayName`.

See nearby users

Issue an authenticated GET to `https://grindr.mobi/v3/locations/<Geohash>/profiles` with no payload. You can optionally add query parameters. These are known:

```
online=false
photoOnly=false
favorite=false
pageNumber=1
```

The Geohash appears to just be the hashing algorithm described at `https://en.wikipedia.org/wiki/Geohash`.

Example:

```
https://grindr.mobi/v3/locations/hcx6g89bqfx7/online=false&photoOnly=false&favorit
e=false&pageNumber=1
```

If successful, the server responds wtih HTTP/200 and the following array:

```
{
    "ttl": 420000,
    "profiles": [
        {
            "age": null,
            "displayName": "hello world",
            "distance": null,
            "isFavorite": false,
            "profileId": "<his Profile Id>",
            "profileImageMediaHash": "<his Profile image hash>",
            "seen": <integer of unix timestamp in milliseconds>,
            "showAge": false,
            "showDistance": false
        },
        {
            "age": 28,
            "displayName": null,
```

```
            "distance": 1504.691430981795,
            "isFavorite": false,
            "profileId": "<his Profile Id>",
            "profileImageMediaHash": null,
            "seen": <integer of unix timestamp in milliseconds>,
            "showAge": true,
            "showDistance": true
        },
    ]
}
```

The Profile image hash can be dropped into either of these CDN URLs: http://cdns.grindr.com/images/profile/1024x1024/ http://cdns.grindr.com/images/thumb/320x320/

Note that the thumbnail size was increased between v2 (187px) and v3 (320px), and the URL has changed correspondingly.

The `ttl` is probably a hint on how many milliseconds to wait before polling again. The value 420000ms = 7 minutes, which correlates well with the official client's apparent polling interval.

Get the profile of specific users

If you know the Profile IDs of specific users, you can fetch the contents of their profiles by issuing an authenticated POST to `https://grindr.mobi/v3/profiles` with this payload:

```
{
    "targetProfileIds": [
        "12345678",
        "23456789"
    ]
}
```

The server responds with the profile details for each of the requested profiles. The response is the same as to the `locations` API above, and won't be repeated here.

In some cases, the client uses an alternate method to query profiles. Issue an authenticated GET to `https://grindr.mobi/v3/profiles/<his Profile Id>`. The server response is identical to above, except it only returns a single profile in each request.

It's not clear why there are two methods to query a profile.

Add a user to your favorites (add a star)

Issue an authenticated POST to `https://grindr.mobi/v3/favorites/<his profile id>` with no payload. (Not an empty JSON object -- the Content-Length should be 0 and the Content-Type is absent.) If successful, the server responds with HTTP/200 and an empty JSON object.

Remove a user from your favorites (remove a star)

Issue an authenticated DELETE to `https://grindr.mobi/v3/favorites/<his profile id>` with no payload. (Not an empty JSON object -- the Content-Length should be 0 and the Content-Type is absent.) If successful, the server responds with HTTP/200 and an empty JSON object.

See who's blocking you and who you've blocked

Issue an authenticated GET request to `https://grindr.mobi/v3/me/blocks` with no payload.

If successful, the server response with HTTP/200 and this payload:

```
{
    "blockedBy": [
        "<his Profile Id>",
        "<another Profile Id>"
    ],
    "blocking": [
        "<his Profile Id>"
    ]
}
```

In the example above, 2 users have blocked you, and you are blocking 1 user.

Note that the datatype of the Profile Id has changed -- in v2, it was a JSON integer. But in v3, the Profile Id is a JSON string.

Block a user

Issue an authenticated POST to `https://grindr.mobi/v3/blocks/<his profile id>` with no payload. (Not an empty JSON object -- the Content-Length should be 0 and the Content-Type is absent.) If successful, the server responds with HTTP/200 and an empty JSON object.

Unblock all blocked users

Issue an authenticated DELETE to `https://grindr.mobi/v3/me/blocks`. If successful, server responds HTTP/200 with an empty JSON object.

Reporting abuse to the moderator

There are several categories of abuse, for example, spamming or impersonation. Refer to **Managed fields** to see the full list of categories. Select the category that best applies to the situation.

Issue an authenticated POST to `https://grindr.mobi/v3/flags/<Profile Id>` with the payload:

```
{
    "comment": "Spam",
    "reason": 5
}
```

The `reason` is an integer that specifies which category of abuse you're reporting, taken from the managed fields. The `comment` field appears to just re-state the description from the same managed field. (In v2, the GUI allowed the user to type out an explanation of the problem, and that would be submitted as the `comment` in the equivalent v2 API. One can only imagine the quality of those comments led Grindr to remove the GUI for typing a custom explanation, but the API still retains a vestigial comment field.)

If successful, server responds HTTP/200 with an empty JSON object.

Acknowledging chats

After receiving a message from the XMPP server, the official client acknowledges the message by sending an authenticated PUT request to `https://grindr.mobi/v3/me/chat/messages?confirmed=true` with the following payload:

```
{
    "messageIds": [
        "<UUID message id>",
        "<UUID message id>"
    ]
}
```

If successful, the server responds with HTTP/200 and an empty JSON object.

Note that you'll still want to ack the messages to the XMPP server as well, using XEP-0333 and/or XEP-0198.

Photos

Grindr doesn't use photos directly. Instead, Grindr uses a hash of the photo. When you'd like to use a photo (either to modify your profile pic, or to send in chat), you first need to convert your photo into a hash.

Uploading a photo

To convert a photo into a hash, issue an authenticated POST request to `https://g3-beta-upload.grindr.com/v3/me/pics?type=chat` with `Content-Type: image/jpeg` and the payload as the actual JPEG image. If the JPEG is too large, you'll need to first resize it. (On v2, the image size threshold was about 800kb -- it's unknown exactly what the maximium size is in v3.)

If the upload is successful, the server responds with HTTP/200 and this response:

```
{
    "mediaHash": "<Image hash>"
}
```

The hash can now be used on the CDN, for example: `http://cdns.grindr.com/grindr/chat/<Image hash>`

Saving the photo to the user's favorite chat photos

Grindr maintains a server-side list of favorite chat photos. To add a photo to the list, issue an authenticated POST to `https://grindr.mobi/v3/me/prefs/chat-pix/<Image hash>`.

If successful, the server responds with HTTP/200 and this rather redundant response:

```
{
    "mediaHash": "<Image hash>"
}
```

Removing a photo from the user's favorite chat photos

Grindr maintains a server-side list of favorite chat photos. To remove a photo from the list, issue an authenticated DELETE to `https://grindr.mobi/v3/me/prefs/chat-pix/<Image hash>`. If successful, the server responds with HTTP/200 and an empty JSON object.

Retreiving a list of the user's favorite chat photos

See the **Query user preferences** section. The favorite chat photos are stored in the `chatPix` object.

Changing your profile photo

Encode the photo as a JPEG. (Unlike the v2 API, you must crop the large photo on the client side; the server only crops the thumbnail.) If the JPEG is too large, you'll need to first resize it. (On v2, the image size threshold was about 800kb -- it's unknown exactly what the maximium size is in v3.)

Issue an authenticated POST to `https://g3-beta-upload.grindr.com/v3/me/pics?type=profile&thumbCoords=<R>%2C<T>%2C<B>%2C<L>` with the large JPEG as payload.

- Set `<R>` to the x-offset of the right edge of the thumbnail.
- Set `<T>` to the y-offset of the top edge of the thumbnail.
- Set `<B>` to the y-offset of the bottom edge of the thumbnail.
- Set `<L>` to the x-offset of the left edge of the thumbnail.

Because the thumbnail must be square, you must ensure that `<R> - <L> == <B> - <T>`. You must also ensure that `<R>` is not larger than the width of the large JPEG, and `<B>` is not larger than the height of the large JPEG.

If successful, the server responds with HTTP/200 and this JSON payload:

```
{
    "action": "pending",
    "mediaHash": "<Image hash>"
}
```

You will need to poll for system messages to find out whether the profile was approved by Grindr's moderators -- see **System messages** above.

Chat

TODO: Document the XMPP side of things.

Advertising and telemetry

The official app communicates with a large number of advertising and analytics servers.

Broadcast messages

In addition to general-purpose ad networks, Grindr also runs its own ad delivery system through its REST endpoint. Should you feel the need to look at their ads, send a GET to `https://grindr.mobi/v3/broadcastMessages`. Although the API doesn't *require* authentication, ads are targeted to the individual, so you probably won't get any ads unless you authenticate.

If successful, the server responds with HTTP/200 and this payload:

```
{
    "broadcastMessages": [
        {
            "actionTitle": "More",
            "body": "Don't miss this amazing bar in <user's town>. Tap 'More' for
info.",
            "dismissTitle": "Close",
            "expirationDate": <integer with Unix timestamp in milliseconds>,
            "messageId": <integer represending the Advertisement Id>,
            "title": "Meet local singles",
            "url": "http://grindr.me/123456"
        },
        {
            "actionTitle": "More",
            "body": "But prefer travel. Tap \"More\" to download and find a room
now. ",
            "dismissTitle": "Close",
            "expirationDate": <integer with Unix timestamp in milliseconds>,
            "messageId": <integer represending the Advertisement Id>,
            "title": "We Can Host",
            "url": "https://app.adjust.com/<redacted>?campaign=
<redacted>&adgroup=Grindr&creative=<redacted>"
        }
    ]
}
```

Unknown APIs

Disassembly of the official Android app shows that there are more APIs that we haven't yet reverse-engineered:

```
/v3/me/conversations
/v3/me/conversations/{id}
/v3/me/pics
/v3/me/prefs/phrases       // probably an xtra feature to add canned phrases
```

TODO: Figure out what these do and document them.