namespace java com.twitter.tweetypie.thriftjava

#@namespace scala com.twitter.tweetypie.thriftscala

#@namespace strato com.twitter.tweetypie

namespace py gen.twitter.tweetypie.service

namespace rb TweetyPie

namespace go tweetypie

include "com/twitter/bouncer/bounce.thrift"

include "com/twitter/carousel/service/carousel\_service.thrift"

include "com/twitter/context/feature\_context.thrift"

include "com/twitter/mediaservices/commons/MediaCommon.thrift"

include "com/twitter/mediaservices/commons/MediaInformation.thrift"

include "com/twitter/servo/exceptions.thrift"

include "com/twitter/spam/features/safety\_meta\_data.thrift"

include "com/twitter/spam/rtf/safety\_label.thrift"

include "com/twitter/spam/rtf/safety\_level.thrift"

include "com/twitter/spam/rtf/safety\_result.thrift"

include "com/twitter/tseng/withholding/withholding.thrift"

include "com/twitter/tweetypie/deleted\_tweet.thrift"

include "com/twitter/tweetypie/transient\_context.thrift"

include "com/twitter/tweetypie/tweet.thrift"

include "com/twitter/tweetypie/tweet\_audit.thrift"

include "com/twitter/incentives/jiminy/jiminy.thrift"

include "unified\_cards\_contract.thrift"

typedef i16 FieldId

struct TweetGeoSearchRequestID {

1: required string id (personalDataType = 'PrivateTweetEntitiesAndMetadata, PublicTweetEntitiesAndMetadata')

}(hasPersonalData = 'true')

struct TweetCreateGeo {

1: optional tweet.GeoCoordinates coordinates

2: optional string place\_id (personalDataType = 'InferredLocation')

3: optional map<string, string> place\_metadata (personalDataTypeKey = 'InferredLocation', personalDataTypeValue = 'InferredLocation')

4: bool auto\_create\_place = 1

// deprecated; use tweet.GeoCoordinates.display

5: bool display\_coordinates = 1

6: bool override\_user\_geo\_setting = 0

7: optional TweetGeoSearchRequestID geo\_search\_request\_id

}(hasPersonalData = 'true')

enum StatusState {

/\*\*

\* The tweet was found and successfully hydrated.

\*/

FOUND = 0

/\*\*

\* The tweet was not found. It may have been deleted, or could just be an invalid or

\* unused tweet id.

\*/

NOT\_FOUND = 1

/\*\*

\* The tweet was found, but there was at least one error hydrating some data on the tweet.

\* GetTweetResult.missing\_fields indicates which fields may have not been hydrated completely.

\*/

PARTIAL = 2

/\*\*

\* @deprecated All failures, including time outs, are indicated by `Failed`.

\*/

TIMED\_OUT = 3

/\*\*

\* There was an upstream or internal failure reading this tweet. Usually indicates a

\* transient issue that is safe to retry immediately.

\*/

FAILED = 4

/\*\*

\* @deprecated tweets from deactivated users will soon be indicated via `Drop` with

\* a `FilteredReason` of `authorAccountIsInactive`.

\*/

DEACTIVATED\_USER = 5

/\*\*

\* @deprecated tweets from suspended users will soon be indicated via `Drop` with

\* a `FilteredReason` of `authorAccountIsInactive`.

\*/

SUSPENDED\_USER = 6

/\*\*

\* @deprecated tweets from protected users that the viewer can't see will soon be

\* indicated via `Drop` with a `FilteredReason` of `authorIsProtected`.

\*/

PROTECTED\_USER = 7

/\*\*

\* @deprecated tweets that have been reported by the viewer will soon be indicated

\* via `Drop` or `Suppress` with a `FilteredReason` of `reportedTweet`.

\*/

REPORTED\_TWEET = 8

// PrivateTweet was originally used for TwitterSuggest v1 but has since been removed

// obsolete: PRIVATE\_TWEET = 9

/\*\*

\* Could not return this tweet because of backpressure, should

\* not be retried immediately; try again later

\*/

OVER\_CAPACITY = 10

/\*\*

\* Returned when the requesting client is considered to not be

\* able to render the tweet properly

\*/

UNSUPPORTED\_CLIENT = 11

/\*\*

\* The tweet exists, but was not returned because it should not be seen by the

\* viewer. The reason for the tweet being filtered is indicated via

\* GetTweetResult.filtered\_reason.

\*/

DROP = 12

/\*\*

\* The tweet exists and was returned, but should not be directly shown to the

\* user without additional user intent to see the tweet, as it may be offensive.

\* The reason for the suppression is indicated via GetTweetResult.filtered\_reason.

\*/

SUPPRESS = 13

/\*\*

\* The tweet once existed and has been deleted.

\* When GetTweetOptions.enable\_deleted\_state is true, deleted tweets

\* will be returned as DELETED

\* When GetTweetOptions.enable\_deleted\_state is false, deleted tweets

\* will be returned as NOT\_FOUND.

\*/

DELETED = 14

/\*\*

\* The tweet once existed, had violated Twitter Rules, and has been deleted.

\* When GetTweetOptions.enable\_deleted\_state is true, bounce-deleted tweets

\* will be returned as BOUNCE\_DELETED

\* When GetTweetOptions.enable\_deleted\_state is false, bounce-deleted tweets

\* will be returned as NOT\_FOUND.

\*/

BOUNCE\_DELETED = 15

RESERVED\_1 = 16

RESERVED\_2 = 17

RESERVED\_3 = 18

RESERVED\_4 = 19

}

enum TweetCreateState {

/\*\*

\* Tweet was created successfully.

\*/

OK = 0,

/\*\*

\* The user\_id field from the creation request does not correspond to a user.

\*/

USER\_NOT\_FOUND = 1,

SOURCE\_TWEET\_NOT\_FOUND = 2,

SOURCE\_USER\_NOT\_FOUND = 3,

/\*\*

\* @deprecated Users can now retweet their own tweets.

\*/

CANNOT\_RETWEET\_OWN\_TWEET = 4,

CANNOT\_RETWEET\_PROTECTED\_TWEET = 5,

CANNOT\_RETWEET\_SUSPENDED\_USER = 6,

CANNOT\_RETWEET\_DEACTIVATED\_USER = 7,

CANNOT\_RETWEET\_BLOCKING\_USER = 8,

ALREADY\_RETWEETED = 9,

CONTRIBUTOR\_NOT\_SUPPORTED = 10,

/\*\*

\* The created\_via field from the creation request does not correspond to a

\* known client application.

\*/

DEVICE\_SOURCE\_NOT\_FOUND = 11,

MALWARE\_URL = 12,

INVALID\_URL = 13,

USER\_DEACTIVATED = 14,

USER\_SUSPENDED = 15,

TEXT\_TOO\_LONG = 16,

TEXT\_CANNOT\_BE\_BLANK = 17,

DUPLICATE = 18,

/\*\*

\* PostTweetRequest.in\_reply\_to\_tweet\_id was set to a tweet that cannot be found.

\*

\* This usually means that the tweet was recently deleted, but could also

\* mean that the tweet isn't visible to the reply author. (This is the

\* case for replies by blocked users.)

\*/

IN\_REPLY\_TO\_TWEET\_NOT\_FOUND = 19,

INVALID\_IMAGE = 20,

INVALID\_ADDITIONAL\_FIELD = 21,

RATE\_LIMIT\_EXCEEDED = 22,

INVALID\_NARROWCAST = 23,

/\*\*

\* Antispam systems (Scarecrow) denied the request.

\*

\* This happens for tweets that are probably spam, but there is some

\* uncertainty. Tweets that Scarecrow is certain are spammy will appear to

\* succeed, but will not be added to backends.

\*/

SPAM = 24,

SPAM\_CAPTCHA = 25,

/\*\*

\* A provided media upload ID can't be resolved.

\*/

MEDIA\_NOT\_FOUND = 26,

/\*\*

\* Catch-all for when uploaded media violate some condition.

\*

\* For example, too many photos in a multi-photo-set, or including an

\* animated gif or video in a multi-photo-set.

\*/

INVALID\_MEDIA = 27,

/\*\*

\* Returned when Scarecrow tell us to rate limit a tweet request.

\*

\* Non verified users (i.e., phone verified, email verified) have more

\* strict rate limit.

\*/

SAFETY\_RATE\_LIMIT\_EXCEEDED = 28,

/\*\*

\* Scarecrow has rejected the creation request until the user completes the

\* bounce assignment.

\*

\* This flag indicates that PostTweetResult.bounce will contain a Bounce

\* struct to be propagated to the client.

\*/

BOUNCE = 29,

/\*\*

\* Tweet creation was denied because the user is in ReadOnly mode.

\*

\* As with SPAM, tweets will appear to succeed but will not be actually

\* created.

\*/

USER\_READONLY = 30,

/\*\*

\* Maximum number of mentions allowed in a tweet was exceeded.

\*/

MENTION\_LIMIT\_EXCEEDED = 31,

/\*\*

\* Maximum number of URLs allowed in a tweet was exceeded.

\*/

URL\_LIMIT\_EXCEEDED = 32,

/\*\*

\* Maximum number of hashtags allowed in a tweet was exceeded.

\*/

HASHTAG\_LIMIT\_EXCEEDED = 33,

/\*\*

\* Maximum number of cashtags allowed in a tweet was exceeded.

\*/

CASHTAG\_LIMIT\_EXCEEDED = 34,

/\*\*

\* Maximum length of a hashtag was exceeded.

\*/

HASHTAG\_LENGTH\_LIMIT\_EXCEEDED = 35,

/\*\*

\* Returned if a request contains more than one attachment type, which

\* includes media, attachment\_url, and card\_reference.

\*/

TOO\_MANY\_ATTACHMENT\_TYPES = 36,

/\*\*

\* Returned if the request contained an attachment URL that isn't allowed.

\*/

INVALID\_ATTACHMENT\_URL = 37,

/\*\*

\* We don't allow users without screen names to be retweeted.

\*/

CANNOT\_RETWEET\_USER\_WITHOUT\_SCREEN\_NAME = 38,

/\*\*

\* Tweets may not be allowed if replying or retweeting IPI'd tweets

\* See go/tp-ipi-tdd for more details

\*/

DISABLED\_BY\_IPI\_POLICY = 39,

/\*\*

\* This state expands our transparency around which URLs are blacklisted or limited

\*/

URL\_SPAM = 40,

// Conversation controls are only valid when present on a root

// conversation tweet and quoted tweets.

INVALID\_CONVERSATION\_CONTROL = 41,

// Reply Tweet is limited due to conversation controls state set on

// root conversation Tweet.

REPLY\_TWEET\_NOT\_ALLOWED = 42,

// Nudge is returned when the client provides nudgeOptions and tweetypie receives a nudge

// from the Jiminy strato column.

NUDGE = 43,

// ApiError BadRequest (400) "Reply to a community tweet must also be a community tweet"

// -- Triggered when a user tries replying to a community tweet with a non community tweet.

COMMUNITY\_REPLY\_TWEET\_NOT\_ALLOWED = 44,

// ApiError Forbidden (403) "User is not authorized to post to this community"

// -- Triggered when a user tries posting to a public/closed community that they are not part of.

COMMUNITY\_USER\_NOT\_AUTHORIZED = 45,

// ApiError NotFound (404) "Community does not exist" -- Triggered when:

// a) A user tries posting to a private community they are not a part of.

// b) A user tries posting to a non existent community

COMMUNITY\_NOT\_FOUND = 46,

// ApiError BadRequest (400) "Cannot retweet a community tweet"

// -- Triggered when a user tries to retweet a community tweet. Community tweets can not be retweeted.

COMMUNITY\_RETWEET\_NOT\_ALLOWED = 47,

// Attempt to tweet with Conversation Controls was rejected, e.g. due to feature switch authorization.

CONVERSATION\_CONTROL\_NOT\_ALLOWED = 48,

// Super follow tweets require a special permission to create.

SUPER\_FOLLOWS\_CREATE\_NOT\_AUTHORIZED = 49,

// Not all params can go together. E.g. super follow tweets can not be community tweets.

SUPER\_FOLLOWS\_INVALID\_PARAMS = 50,

// ApiError Forbidden (403) "Protected user can not post to communities"

// -- Triggered when a protected user tries tweeting or replying

// to a community tweet. They are not allowed to create community tweets.

COMMUNITY\_PROTECTED\_USER\_CANNOT\_TWEET = 51,

// ApiError Forbidden (451) "User is not permitted to engage with this exclusive tweet."

// -- Triggered when a user tries to reply to an exclusive tweet without being

// a superfollower of the tweet author. Could be used for other engagements in the future (e.g. favorite)

EXCLUSIVE\_TWEET\_ENGAGEMENT\_NOT\_ALLOWED = 52

/\*\*

\* ApiError BadRequest (400) "Invalid parameters on Trusted Friends tweet creation"

\*

\* Returned when either of the following occur:

\* a) A user tries setting Trusted Friends Control on a reply

\* b) A user tries setting Trusted Friends Control on a tweet with any of the following set:

\* i) Conversation Control

\* ii) Community

\* iii) Exclusive Tweet Control

\*/

TRUSTED\_FRIENDS\_INVALID\_PARAMS = 53,

/\*\*

\* ApiError Forbidden (403)

\*

\* Returned when a user tries to retweet a Trusted Friends tweet.

\*/

TRUSTED\_FRIENDS\_RETWEET\_NOT\_ALLOWED = 54,

/\*\*

\* ApiError Forbidden (457)

\*

\* Returned when a user tries to reply to a Trusted Friends tweet

\* and they are not a trusted friend.

\*/

TRUSTED\_FRIENDS\_ENGAGEMENT\_NOT\_ALLOWED = 55,

/\*\*

\* ApiError BadRequest (400) "Invalid parameters for creating a CollabTweet or CollabInvitation"

\*

\* Returned when any of the following are true:

\* a) A user tries setting Collab Control on a reply

\* b) A user tries setting Collab Control on a tweet with any of the following set:

\* i) Conversation Control

\* ii) Community

\* iii) Exclusive Tweet Control

\* iv) Trusted Friends Control

\*\*/

COLLAB\_TWEET\_INVALID\_PARAMS = 56,

/\*\*

\* ApiError Forbidden (457)

\*

\* Returned when a user tries to create a Trusted Friends tweet but they are not allowed to tweet

\* to the requested Trusted Friends list.

\*/

TRUSTED\_FRIENDS\_CREATE\_NOT\_ALLOWED = 57,

/\*\*

\* Returned when the current user is not allowed to edit in general, this might be due to missing

\* roles during development, or a missing subscription.

\*/

EDIT\_TWEET\_USER\_NOT\_AUTHORIZED = 58,

/\*\*

\* Returned when a user tries to edit a Tweet which they didn't author.

\*/

EDIT\_TWEET\_USER\_NOT\_AUTHOR = 59,

/\*\*

\* Returned when a user tries edit a stale tweet, meaning a tweet which has already been edited.

\*/

EDIT\_TWEET\_NOT\_LATEST\_VERSION = 60,

/\*\*

\* ApiError Forbidden (460)

\*

\* Returned when a user tries to create a Trusted Friends tweet that quotes tweets a Trusted

\* Friends tweet.

\*/

TRUSTED\_FRIENDS\_QUOTE\_TWEET\_NOT\_ALLOWED = 61,

/\*\*

\* Returned when a user tries edit a tweet for which the editing time has already expired.

\*/

EDIT\_TIME\_LIMIT\_REACHED = 62,

/\*\*

\* Returned when a user tries edit a tweet which has been already edited maximum number of times.

\*/

EDIT\_COUNT\_LIMIT\_REACHED = 63,

/\* Returned when a user tries to edit a field that is not allowed to be edited \*/

FIELD\_EDIT\_NOT\_ALLOWED = 64,

/\* Returned when the initial Tweet could not be found when trying to validate an edit \*/

INITIAL\_TWEET\_NOT\_FOUND = 65,

/\*\*

\* ApiError Forbidden (457)

\*

\* Returned when a user tries to reply to a stale tweet

\*/

STALE\_TWEET\_ENGAGEMENT\_NOT\_ALLOWED = 66,

/\*\*

\* ApiError Forbidden (460)

\*

\* Returned when a user tries to create a tweet that quotes tweets a stale tweet

\*/

STALE\_TWEET\_QUOTE\_TWEET\_NOT\_ALLOWED = 67,

/\* Tweet cannot be edited because the initial tweet is

\* marked as not edit eligible \*/

NOT\_ELIGIBLE\_FOR\_EDIT = 68,

/\* A stale version of an edit tweet cannot be retweeted

\* Only latest version of an edit chain should be allowed to be retweeted. \*/

STALE\_TWEET\_RETWEET\_NOT\_ALLOWED = 69,

RESERVED\_32 = 70,

RESERVED\_33 = 71,

RESERVED\_34 = 72,

RESERVED\_35 = 73,

RESERVED\_36 = 74,

RESERVED\_37 = 75,

}

enum UndeleteTweetState {

/\*\*

\* The Tweet was successfully undeleted.

\*/

SUCCESS = 0,

/\*\*

\* The Tweet was deleted and is still deleted. It cannot be undeleted

\* because the tweet is no longer in the soft delete archive.

\*/

SOFT\_DELETE\_EXPIRED = 1,

/\*\*

\* The Tweet likely has never existed, and therefore cannot be undeleted.

\*/

TWEET\_NOT\_FOUND = 2,

/\*\*

\* The Tweet could not be undeleted because it was not deleted in

\* the first place.

\*/

TWEET\_ALREADY\_EXISTS = 3,

/\*\*

\* The user who created the Tweet being undeleted could not be found.

\*/

USER\_NOT\_FOUND = 4,

/\*\*

\* The Tweet could not be undeleted because it is a retweet and the original

\* tweet is gone.

\*/

SOURCE\_TWEET\_NOT\_FOUND = 5,

/\*\*

\* The Tweet could not be undeleted because it is a retweet and the author

\* of the original tweet is gone.

\*/

SOURCE\_USER\_NOT\_FOUND = 6,

/\*\*

\* The Tweet was deleted and is still deleted. It cannot be undeleted

\* because the tweet has been bounce deleted. Bounce deleted tweet

\* has been found to violate Twitter Rules. go/bouncer go/bounced-tweet

\*/

TWEET\_IS\_BOUNCE\_DELETED = 7,

/\*\*

\* This tweet cannot be undeleted because the tweet was created by a

\* user when they were under 13.

\*\*/

TWEET\_IS\_U13\_TWEET = 8,

RESERVED\_2 = 9,

RESERVED\_3 = 10

}

enum TweetDeleteState {

/\*\*

\* Tweet was deleted successfully.

\*/

OK = 0,

/\*\*

\* Tweet was not deleted because of the associated user.

\*

\* The DeleteTweetsRequest.by\_user\_id must match the tweet owner or be an

\* admin user.

\*/

PERMISSION\_ERROR = 1,

/\*\*

\* The expected\_user\_id provided in DeleteTweetsRequest does not match the

\* user\_id of the tweet owner.

\*/

EXPECTED\_USER\_ID\_MISMATCH = 2,

/\*\*

\* @deprecated.

\*

\* is\_user\_erasure was set in DeleteTweetsRequest but the user was not in

\* the erased state.

\*/

USER\_NOT\_IN\_ERASED\_STATE = 3,

/\*\*

\* Failed to Load the source Tweet while unretweeting stale revisions in an edit chain.

\*/

SOURCE\_TWEET\_NOT\_FOUND = 4,

RESERVED\_4 = 5,

RESERVED\_5 = 6,

RESERVED\_6 = 7,

RESERVED\_7 = 8

}

enum DeletedTweetState {

/\*\*

\* The tweet has been marked as deleted but has not been permanently deleted.

\*/

SOFT\_DELETED = 1

/\*\*

\* The tweet has never existed.

\*/

NOT\_FOUND = 2

/\*\*

\* The tweet has been permanently deleted.

\*/

HARD\_DELETED = 3

/\*\*

\* The tweet exists and is not currently deleted.

\*/

NOT\_DELETED = 4

RESERVED1 = 5

RESERVED2 = 6

RESERVED3 = 7

}

/\*\*

\* Hydrations to perform on the Tweet returned by post\_tweet and post\_retweet.

\*/

struct WritePathHydrationOptions {

/\*\*

\* Return cards for tweets with cards in Tweet.cards or Tweet.card2

\*

\* card2 also requires setting a valid cards\_platform\_key

\*/

1: bool include\_cards = 0

/\*\*

\* The card format version supported by the requesting client

\*/

2: optional string cards\_platform\_key

# 3: obsolete

# 4: obsolete

/\*\*

\* The argument passed to the Stratostore extension points mechanism.

\*/

5: optional binary extensions\_args

/\*\*

\* When returning a tweet that quotes another tweet, do not include

\* the URL to the quoted tweet in the tweet text and url entities.

\* This is intended for clients that use the quoted\_tweet field of

\* the tweet to display quoted tweets. Also see simple\_quoted\_tweet

\* field in GetTweetOptions and GetTweetFieldsOptions

\*/

6: bool simple\_quoted\_tweet = 0

}

struct RetweetRequest {

/\*\*

\* Id of the tweet being retweeted.

\*/

1: required i64 source\_status\_id (personalDataType = 'TweetId')

/\*\*

\* User creating the retweet.

\*/

2: required i64 user\_id (personalDataType = 'UserId')

/\*\*

\* @see PostTweetRequest.created\_via

\*/

3: required string created\_via (personalDataType = 'ClientType')

4: optional i64 contributor\_user\_id (personalDataType = 'UserId') // no longer supported

/\*\*

\* @see PostTweetRequest.tracking\_id

\*/

5: optional i64 tracking\_id (personalDataType = 'ImpressionId')

6: optional tweet.Narrowcast narrowcast

/\*\*

\* @see PostTweetRequest.nullcast

\*/

7: bool nullcast = 0

/\*\*

\* @see PostTweetRequest.dark

\*/

8: bool dark = 0

// OBSOLETE 9: bool send\_retweet\_sms\_push = 0

10: optional WritePathHydrationOptions hydration\_options

/\*\*

\* @see PostTweetRequest.additional\_fields

\*/

11: optional tweet.Tweet additional\_fields

/\*\*

\* @see PostTweetRequest.uniqueness\_id

\*/

12: optional i64 uniqueness\_id (personalDataType = 'PrivateTweetEntitiesAndMetadata, PublicTweetEntitiesAndMetadata')

13: optional feature\_context.FeatureContext feature\_context

14: bool return\_success\_on\_duplicate = 0

/\*\*

\* Passthrough data for Scarecrow that is used for safety checks.

\*/

15: optional safety\_meta\_data.SafetyMetaData safety\_meta\_data

/\*\*

\* This is a unique identifier used in both the REST and GraphQL-dark

\* requests that will be used to correlate the GraphQL mutation requests to the REST requests

\* during a transition period when clients will be moving toward tweet creation via GraphQL.

\* See also, the "Comparison Testing" section at go/tweet-create-on-graphql-tdd for additional

\* context.

\*/

16: optional string comparison\_id (personalDataType = 'UniversallyUniqueIdentifierUuid')

}(hasPersonalData = 'true')

/\*\*

\* A request to set or unset nsfw\_admin and/or nsfw\_user.

\*/

struct UpdatePossiblySensitiveTweetRequest {

/\*\*

\* Id of tweet being updated

\*/

1: required i64 tweet\_id (personalDataType = 'TweetId')

/\*\*

\* Id of the user initiating this request.

\*

\* It could be either the owner of the tweet or an admin. It is used when

\* auditing the request in Guano.

\*/

2: required i64 by\_user\_id (personalDataType = 'UserId')

/\*\*

\* New value for tweet.core\_data.nsfw\_admin.

\*/

3: optional bool nsfw\_admin

/\*\*

\* New value for tweet.core\_data.nsfw\_user.

\*/

4: optional bool nsfw\_user

/\*\*

\* Host or remote IP where the request originated.

\*

\* This data is used when auditing the request in Guano. If unset, it will

\* be logged as "<unknown>".

\*/

5: optional string host (personalDataType = 'IpAddress')

/\*\*

\* Pass-through message sent to the audit service.

\*/

6: optional string note

}(hasPersonalData = 'true')

struct UpdateTweetMediaRequest {

/\*\*

\* The tweet id that's being updated

\*/

1: required i64 tweet\_id (personalDataType = 'TweetId')

/\*\*

\* A mapping from old (existing) media ids on the tweet to new media ids.

\*

\* Existing tweet media not in this map will remain unchanged.

\*/

2: required map<i64, i64> old\_to\_new\_media\_ids (personalDataTypeKey = 'MediaId', personalDataTypeValue = 'MediaId')

}(hasPersonalData = 'true')

struct TakedownRequest {

1: required i64 tweet\_id (personalDataType = 'TweetId')

/\*\*

\* The list of takedown country codes to add to the tweet.

\*

\* DEPRECATED, reasons\_to\_add should be used instead.

\*/

2: list<string> countries\_to\_add = [] (personalDataType = 'ContentRestrictionStatus')

/\*\*

\* This field is the list of takedown country codes to remove from the tweet.

\*

\* DEPRECATED, reasons\_to\_remove should be used instead.

\*/

3: list<string> countries\_to\_remove = [] (personalDataType = 'ContentRestrictionStatus')

/\*\*

\* This field is the list of takedown reasons to add to the tweet.

\*/

11: list<withholding.TakedownReason> reasons\_to\_add = []

/\*\*

\* This field is the list of takedown reasons to remove from the tweet.

\*/

12: list<withholding.TakedownReason> reasons\_to\_remove = []

/\*\*

\* Motivation for the takedown which is written to the audit service.

\*

\* This data is not persisted with the takedown itself.

\*/

4: optional string audit\_note (personalDataType = 'AuditMessage')

/\*\*

\* Whether to send this request to the audit service.

\*/

5: bool scribe\_for\_audit = 1

// DEPRECATED, this field is no longer used.

6: bool set\_has\_takedown = 1

// DEPRECATED, this field is no longer used.

7: optional list<string> previous\_takedown\_country\_codes (personalDataType = 'ContentRestrictionStatus')

/\*\*

\* Whether this request should enqueue a TweetTakedownEvent to EventBus and

\* Hosebird.

\*/

8: bool eventbus\_enqueue = 1

/\*\*

\* ID of the user who initiated the takedown.

\*

\* This is used when writing the takedown to the audit service. If unset, it

\* will be logged as -1.

\*/

9: optional i64 by\_user\_id (personalDataType = 'UserId')

/\*\*

\* Host or remote IP where the request originated.

\*

\* This data is used when auditing the request in Guano. If unset, it will

\* be logged as "<unknown>".

\*/

10: optional string host (personalDataType = 'IpAddress')

}(hasPersonalData = 'true')

// Arguments to delete\_location\_data

struct DeleteLocationDataRequest {

1: i64 user\_id (personalDataType = 'UserId')

}(hasPersonalData = 'true')

// structs for API V2 (flexible schema)

struct GetTweetOptions {

/\*\*

\* Return the original tweet in GetTweetResult.source\_tweet for retweets.

\*/

1: bool include\_source\_tweet = 1

/\*\*

\* Return the hydrated Place object in Tweet.place for tweets with geolocation.

\*/

2: bool include\_places = 0

/\*\*

\* Language used for place names when include\_places is true. Also passed to

\* the cards service, if cards are hydrated for the request.

\*/

3: string language\_tag = "en"

/\*\*

\* Return cards for tweets with cards in Tweet.cards or Tweet.card2

\*

\* card2 also requires setting a valid cards\_platform\_key

\*/

4: bool include\_cards = 0

/\*\*

\* Return the number of times a tweet has been retweeted in

\* Tweet.counts.retweet\_count.

\*/

5: bool include\_retweet\_count = 0

/\*\*

\* Return the number of direct replies to a tweet in

\* Tweet.counts.reply\_count.

\*/

6: bool include\_reply\_count = 0

/\*\*

\* Return the number of favorites a tweet has received in

\* Tweet.counts.favorite\_count.

\*/

7: bool include\_favorite\_count = 0

# OBSOLETE 8: bool include\_unique\_users\_impressed\_count = 0

# OBSOLETE 9: bool include\_click\_count = 0

# OBSOLETE 10: bool include\_descendent\_reply\_count = 0

/\*\*

\* @deprecated Use safety\_level for spam filtering.

\*/

11: optional tweet.SpamSignalType spam\_signal\_type

/\*\*

\* If the requested tweet is not already in cache, do not add it.

\*

\* You should set do\_not\_cache to true if you are requesting old tweets

\* (older than 30 days) and they are unlikely to be requested again.

\*/

12: bool do\_not\_cache = 0

/\*\*

\* The card format version supported by the requesting client

\*/

13: optional string cards\_platform\_key (personalDataType = 'PrivateTweetEntitiesAndMetadata, PublicTweetEntitiesAndMetadata')

/\*\*

\* The user for whose perspective this request should be processed.

\*

\* If you are requesting tweets on behalf of a user, set this to their user

\* id. The effect of setting this option is:

\*

\* - Tweetypie will return protected tweets that the user is allowed to

\* access, rather than filtering out protected tweets.

\*

\* - If this field is set \*and\* `include\_perspectivals` is set, then the

\* tweets will have the `perspective` field set to a struct with flags

\* that indicate whether the user has favorited, retweeted, or reported

\* the tweet in question.

\*

\* If you have a specific need to access all protected tweets (not

\* just tweets that should be accessible to the current user), see the

\* documentation for `include\_protected`.

\*/

14: optional i64 for\_user\_id (personalDataType = 'UserId')

/\*\*

\* Do not enforce normal filtering for protected tweets, blocked quote tweets,

\* contributor data, etc. This does not affect Visibility Library (http://go/vf)

\* based filtering which executes when safety\_level is specified, see request

\* field 24 safety\_level below

\*

\* If `bypass\_visibility\_filtering` is true, Tweetypie will not enforce filtering

\* for protected tweets, blocked quote tweets, contributor data, etc. and your client

\* will receive all tweets regardless of follow relationship. You will also be able

\* to access tweets from deactivated and suspended users. This is only necessary

\* for special cases, such as indexing or analyzing tweets, or administrator access.

\* Since this elevated access is usually unnecessary, and is a security risk, you will

\* need to get your client id whitelisted to access this feature.

\*

\* If you are accessing tweets on behalf of a user, set

\* `bypass\_visibility\_filtering` to false and set `for\_user\_id`. This will

\* allow access to exactly the set of tweets that that user is authorized to

\* access, and filter out tweets the user should not be authorized to access

\* (returned with a StatusState of PROTECTED\_USER).

\*/

15: bool bypass\_visibility\_filtering = 0

/\*\*

\* Return the user-specific view of a tweet in Tweet.perspective

\*

\* for\_user\_id must also be set.

\*/

16: bool include\_perspectivals = 0

// OBSOLETE media faces are always included

17: bool include\_media\_faces = 0

/\*\*

\* The flexible schema fields of the tweet to return.

\*

\* Fields of tweets in the 100+ range will only be returned if they are

\* explicitly requested.

\*/

18: list<FieldId> additional\_field\_ids = []

// OBSOLETE 19: bool include\_topic\_labels = 0

/\*\*

\* Exclude user-reported tweets from this request. Only applicable if

\* forUserId is set.

\*

\* Users can report individual tweets in the UI as uninteresting, spam,

\* sensitive, or abusive.

\*/

20: bool exclude\_reported = 0

// if set to true, disables suggested tweet visibility checks

// OBSOLETE (TwitterSuggestInfo version of suggested tweets has been removed)

21: bool obsolete\_skip\_twitter\_suggests\_visibility\_check = 0

// OBSOLETE 22: optional set<tweet.SpamSignalType> spam\_signal\_types

/\*\*

\* Return the quoted tweet in GetTweetResult.quoted\_tweet

\*/

23: bool include\_quoted\_tweet = 0

/\*\*

\* Content filtering policy that will be used to drop or suppress tweets

\* from response. The filtering is based on the result of Visibility Library

\* and does not affect filtering of tweets from blocked or non-followed protected users, see

\* request field 15 bypass\_visibility\_filtering above

\*

\* If not specified SafetyLevel.FilterDefault will be used.

\*/

24: optional safety\_level.SafetyLevel safety\_level

// obsolete 25: bool include\_animated\_gif\_media\_entities = 0

26: bool include\_profile\_geo\_enrichment = 0

// obsolete 27: optional set<string> extensions

28: bool include\_tweet\_pivots = 0

/\*\*

\* The argument passed to the Stratostore extension points mechanism.

\*/

29: optional binary extensions\_args

/\*\*

\* Return the number of times a tweet has been quoted in Tweet.counts.quote\_count

\*/

30: bool include\_quote\_count = 0

/\*\*

\* Return media metadata from MediaInfoService in MediaEntity.additional\_metadata

\*/

31: bool include\_media\_additional\_metadata = 0

/\*\*

\* Populate the conversation\_muted field of the Tweet for the requesting

\* user.

\*

\* Setting this to true will have no effect unless for\_user\_id is set.

\*/

32: bool include\_conversation\_muted = 0

/\*\*

\* @deprecated go/sunsetting-carousels

\*/

33: bool include\_carousels = 0

/\*\*

\* When enable\_deleted\_state is true and we have evidence that the

\* tweet once existed and was deleted, Tweetypie returns

\* StatusState.DELETED or StatusState.BOUNCE\_DELETED. (See comments

\* on StatusState for details on these two states.)

\*

\* When enable\_deleted\_state is false, deleted tweets are

\* returned as StatusState.NOT\_FOUND.

\*

\* Note: even when enable\_deleted\_state is true, a deleted tweet may

\* still be returned as StatusState.NOT\_FOUND due to eventual

\* consistency.

\*

\* This option is false by default for compatibility with clients

\* expecting StatusState.NOT\_FOUND.

\*/

34: bool enable\_deleted\_state = 0

/\*\*

\* Populate the conversation\_owner\_id field of the Tweet for the requesting

\* user. Which translate into is\_conversation\_owner in birdherd

\*

\*/

// obsolete 35: bool include\_conversation\_owner\_id = 0

/\*\*

\* Populate the is\_removed\_from\_conversation field of the Tweet for the requesting

\* user.

\*

\*/

// obsolete 36: bool include\_is\_removed\_from\_conversation = 0

// To retrieve self-thread metadata request field Tweet.SelfThreadMetadataField

// obsolete 37: bool include\_self\_thread\_info = 0

/\*\*

\* This option surfaces CardReference field (118) in Tweet thrift object.

\* We use card\_uri present in card reference, to get access to stored card information.

\*/

37: bool include\_card\_uri = 0

/\*\*

\* When returning a tweet that quotes another tweet, do not include

\* the URL to the quoted tweet in the tweet text and url entities.

\* This is intended for clients that use the quoted\_tweet field of

\* the tweet to display quoted tweets.

\*/

38: bool simple\_quoted\_tweet = 0

/\*\*

\* This flag is used and only take affect if the requested tweet is creatives container backed

\* tweet. This will suprress the tweet materialization and return tweet not found.

\*

\* go/creatives-containers-tdd

\*\*/

39: bool disable\_tweet\_materialization = 0

/\*\*

\* Used for load shedding. If set to true, Tweetypie service might shed the request, if the service

\* is struggling.

\*\*/

40: optional bool is\_request\_sheddable

}(hasPersonalData = 'true')

struct GetTweetsRequest {

1: required list<i64> tweet\_ids (personalDataType = 'TweetId')

// @deprecated unused

2: optional list<i64> source\_tweet\_id\_hints (personalDataType = 'TweetId')

3: optional GetTweetOptions options

// @deprecated unused

4: optional list<i64> quoted\_tweet\_id\_hints (personalDataType = 'TweetId')

}(hasPersonalData = 'true')

/\*\*

\* Can be used to reference an arbitrary nested field of some struct via

\* a list of field IDs describing the path of fields to reach the referenced

\* field.

\*/

struct FieldByPath {

1: required list<FieldId> field\_id\_path

}

struct GetTweetResult {

1: required i64 tweet\_id (personalDataType = 'TweetId')

/\*\*

\* Indicates what happened when the tweet was loaded.

\*/

2: required StatusState tweet\_state

/\*\*

\* The requested tweet when tweet\_state is `FOUND`, `PARTIAL`, or `SUPPRESS`.

\*

\* This field will be set if the tweet exists, access is authorized,

\* and enough data about the tweet is available to materialize a

\* tweet. When this field is set, you should look at the tweet\_state

\* field to determine how to treat this tweet.

\*

\* If tweet\_state is FOUND, then this tweet is complete and passes the

\* authorization checks requested in GetTweetOptions. (See

\* GetTweetOptions.for\_user\_id for more information about authorization.)

\*

\* If tweet\_state is PARTIAL, then enough data was available to return

\* a tweet, but there was an error when loading the tweet that prevented

\* some data from being returned (for example, if a request to the cards

\* service times out when cards were requested, then the tweet will be

\* marked PARTIAL). `missing\_fields` indicates which parts of the tweet

\* failed to load. When you receive a PARTIAL tweet, it is up to you

\* whether to proceed with the degraded tweet data or to consider it a

\* failure. For example, a mobile client might choose to display a

\* PARTIAL tweet to the user, but not store it in an internal cache.

\*

\* If tweet\_state is SUPPRESS, then the tweet is complete, but soft

\* filtering is enabled. This state is intended to hide potentially

\* harmful tweets from user's view while not taking away the option for

\* the user to override our filtering decision. See http://go/rtf

\* (render-time filtering) for more information about how to treat these

\* tweets.

\*/

3: optional tweet.Tweet tweet

/\*\*

\* The tweet fields that could not be loaded when tweet\_state is `PARTIAL`

\* or `SUPPRESS`.

\*

\* This field will be set when the `tweet\_state` is `PARTIAL`, and may

\* be set when `tweet\_state` is SUPPRESS. It indicates degraded data in

\* the `tweet`. Each entry in `missing\_fields` indicates a traversal of

\* the `Tweet` thrift object terminating at the field that is

\* missing. For most non-core fields, the path will just be the field id

\* of the field that is missing.

\*

\* For example, if card2 failed to load for a tweet, the `tweet\_state`

\* will be `PARTIAL`, the `tweet` field will be set, the Tweet's `card2`

\* field will be empty, and this field will be set to:

\*

\* Set(FieldByPath(Seq(17)))

\*/

4: optional set<FieldByPath> missing\_fields

/\*\*

\* The original tweet when `tweet` is a retweet and

\* GetTweetOptions.include\_source\_tweet is true.

\*/

5: optional tweet.Tweet source\_tweet

/\*\*

\* The retweet fields that could not be loaded when tweet\_state is `PARTIAL`.

\*/

6: optional set<FieldByPath> source\_tweet\_missing\_fields

/\*\*

\* The quoted tweet when `tweet` is a quote tweet and

\* GetTweetOptions.include\_quoted\_tweet is true.

\*/

7: optional tweet.Tweet quoted\_tweet

/\*\*

\* The quoted tweet fields that could not be loaded when tweet\_state is `PARTIAL`.

\*/

8: optional set<FieldByPath> quoted\_tweet\_missing\_fields

/\*\*

\* The reason that a tweet should not be displayed when tweet\_state is

\* `SUPPRESS` or `DROP`.

\*/

9: optional safety\_result.FilteredReason filtered\_reason

/\*\*

\* Hydrated carousel if the tweet contains a carousel URL and the

\* GetTweetOptions.include\_carousel is true.

\*

\* In this case Carousel Service is requested to hydrate the carousel, and

\* the result stored in this field.

\*

\* @deprecated go/sunsetting-carousels

\*/

10: optional carousel\_service.GetCarouselResult carousel\_result

/\*\*

\* If a quoted tweet would be present, but it was filtered out, then

\* this field will be set to the reason that it was filtered.

\*/

11: optional safety\_result.FilteredReason quoted\_tweet\_filtered\_reason

}(hasPersonalData = 'true')

union TweetInclude {

/\*\*

\* Field ID within the `Tweet` struct to include. All fields may be optionally included

\* except for the `id` field.

\*/

1: FieldId tweetFieldId

/\*\*

\* Field ID within the `StatusCounts` struct to include. Only specifically requested

\* count fields will be included. Including any `countsFieldIds` values automatically

\* implies including `Tweet.counts`.

\*

\*/

2: FieldId countsFieldId

/\*\*

\* Field ID within the `MediaEntity` struct to include. Currently, only `MediaEntity.additionalMetadata`

\* may be optionally included (i.e., it will not be included by default if you include

\* `tweetFieldId` = `Tweet.media` without also including `mediaEntityFieldId` =

\* `MediaEntity.additionalMetadata`. Including any `mediaEntityFieldId` values automatically

\* implies include `Tweet.media`.

\*/

3: FieldId mediaEntityFieldId

}

/\*\*

\* An enumeration of policy options indicating how tweets should be filtered (protected tweets, blocked quote tweets,

\* contributor data, etc.). This does not affect Visibility Library (http://go/vf) based filtering.

\* This is equivalent to `bypass\_visibility\_filtering` in get\_tweets() call. This means that

\* `TweetVisibilityPolicy.NO\_FILTERING` is equivalent to `bypass\_visibility\_filtering` = true

\*/

enum TweetVisibilityPolicy {

/\*\*

\* only return tweets that should be visible to either the `forUserId` user, if specified,

\* or from the perspective of a logged-out user if `forUserId` is not specified. This option

\* should always be used if requesting data to be returned via the public API.

\*/

USER\_VISIBLE = 1,

/\*\*

\* returns all tweets that can be found, regardless of user visibility. This option should

\* never be used when gather data to be return in an API, and should only be used for internal

\* processing. because this option allows access to potentially sensitive data, clients

\* must be whitelisted to use it.

\*/

NO\_FILTERING = 2

}

struct GetTweetFieldsOptions {

/\*\*

\* Identifies which `Tweet` or nested fields to include in the response.

\*/

1: required set<TweetInclude> tweet\_includes

/\*\*

\* If true and the requested tweet is a retweet, then a `Tweet`

\* containing the requested fields for the retweeted tweet will be

\* included in the response.

\*/

2: bool includeRetweetedTweet = 0

/\*\*

\* If true and the requested tweet is a quote-tweet, then the quoted

\* tweet will also be queried and the result for the quoted tweet

\* included in `GetTweetFieldsResult.quotedTweetResult`.

\*/

3: bool includeQuotedTweet = 0

/\*\*

\* If true and the requested tweet contains a carousel URL, then the

\* carousel will also be queried and the result for the carousel

\* included in `GetTweetFieldsResult.carouselResult`.

\*

\* @deprecated go/sunsetting-carousels

\*/

4: bool includeCarousel = 0

/\*\*

\* If you are requesting tweets on behalf of a user, set this to their

\* user id. The effect of setting this option is:

\*

\* - Tweetypie will return protected tweets that the user is allowed

\* to access, rather than filtering out protected tweets, when `visibility\_policy`

\* is set to `USER\_VISIBLE`.

\*

\* - If this field is set \*and\* `Tweet.perspective` is requested, then

\* the tweets will have the `perspective` field set to a struct with

\* flags that indicate whether the user has favorited, retweeted, or

\* reported the tweet in question.

\*/

10: optional i64 forUserId (personalDataType = 'UserId')

/\*\*

\* language\_tag is used when hydrating a `Place` object, to get localized names.

\* Also passed to the cards service, if cards are hydrated for the request.

\*/

11: optional string languageTag (personalDataType = 'InferredLanguage')

/\*\*

\* if requesting card2 cards, you must specify the platform key

\*/

12: optional string cardsPlatformKey (personalDataType = 'PrivateTweetEntitiesAndMetadata, PublicTweetEntitiesAndMetadata')

/\*\*

\* The argument passed to the Stratostore extension points mechanism.

\*/

13: optional binary extensionsArgs

/\*\*

\* the policy to use when filtering tweets for basic visibility.

\*/

20: TweetVisibilityPolicy visibilityPolicy = TweetVisibilityPolicy.USER\_VISIBLE

/\*\*

\* Content filtering policy that will be used to drop or suppress tweets from response.

\* The filtering is based on the result of Visibility Library (http://go/vf)

\* and does not affect filtering of tweets from blocked or non-followed protected users, see

\* request field 20 visibilityPolicy above

\*

\* If not specified SafetyLevel.FilterNone will be used.

\*/

21: optional safety\_level.SafetyLevel safetyLevel

/\*\*

\* The tweet result won't be cached by Tweetypie if doNotCache is true.

\* You should set it as true if old tweets (older than 30 days) are requested,

\* and they are unlikely to be requested again.

\*/

30: bool doNotCache = 0

/\*\*

\* When returning a tweet that quotes another tweet, do not include

\* the URL to the quoted tweet in the tweet text and url entities.

\* This is intended for clients that use the quoted\_tweet field of

\* the tweet to display quoted tweets.

\*

\*/

31: bool simple\_quoted\_tweet = 0

/\*\*

\* This flag is used and only take affect if the requested tweet is creatives container backed

\* tweet. This will suprress the tweet materialization and return tweet not found.

\*

\* go/creatives-containers-tdd

\*\*/

32: bool disable\_tweet\_materialization = 0

/\*\*

\* Used for load shedding. If set to true, Tweetypie service might shed the request, if the service

\* is struggling.

\*\*/

33: optional bool is\_request\_sheddable

}(hasPersonalData = 'true')

struct GetTweetFieldsRequest {

1: required list<i64> tweetIds (personalDataType = 'TweetId')

2: required GetTweetFieldsOptions options

} (hasPersonalData = 'true')

/\*\*

\* Used in `TweetFieldsResultState` when the requested tweet is found.

\*/

struct TweetFieldsResultFound {

1: required tweet.Tweet tweet

/\*\*

\* If `tweet` is a retweet, `retweetedTweet` will be the retweeted tweet.

\* Just like with the requested tweet, only the requested fields will be

\* hydrated and set on the retweeted tweet.

\*/

2: optional tweet.Tweet retweetedTweet

/\*\*

\* If specified, then the tweet should be soft filtered.

\*/

3: optional safety\_result.FilteredReason suppressReason

}

/\*\*

\* Used in `TweetFieldsResultState` when the requested tweet is not found.

\*/

struct TweetFieldsResultNotFound {

// If this field is true, then we know that the tweet once existed and

// has since been deleted.

1: bool deleted = 0

// This tweet is deleted after being bounced for violating the Twitter

// Rules and should never be rendered or undeleted. see go/bounced-tweet

// In certain timelines we render a tombstone in its place.

2: bool bounceDeleted = 0

// The reason that a tweet should not be displayed. See go/vf-tombstones-in-tweetypie

// Tweets that are not found do not going through Visibility Filtering rule evaluation and thus

// are not `TweetFieldsResultFiltered`, but may still have a filtered\_reason that distinguishes

// whether the unavailable tweet should be tombstoned or hard-filtered based on the Safety Level.

3: optional safety\_result.FilteredReason filtered\_reason

}

struct TweetFieldsPartial {

1: required TweetFieldsResultFound found

/\*\*

\* The tweet fields that could not be loaded when hydration fails

\* and a backend fails with an exception. This field is populated

\* when a tweet is "partially" hydrated, i.e. some fields were

\* successfully fetched while others were not.

\*

\* It indicates degraded data in the `tweet`. Each entry in `missing\_fields`

\* indicates a traversal of the `Tweet` thrift object terminating at

\* the field that is missing. For most non-core fields, the path will

\* just be the field id of the field that is missing.

\*

\* For example, if card2 failed to load for a tweet, the tweet is marked "partial",

\* the `tweet` field will be set, the Tweet's `card2`

\* field will be empty, and this field will be set to:

\*

\* Set(FieldByPath(Seq(17)))

\*/

2: required set<FieldByPath> missingFields

/\*\*

\* Same as `missing\_fields` but for the source tweet in case the requested tweet

\* was a retweet.

\*/

3: required set<FieldByPath> sourceTweetMissingFields

}

/\*\*

\* Used in `TweetFieldsResultState` when there was a failure loading the requested tweet.

\*/

struct TweetFieldsResultFailed {

/\*\*

\* If true, the failure was the result of backpressure, which means the request

\* should not be immediately retried. It is safe to retry again later.

\*

\* If false, the failure is probably transient and safe to retry immediately.

\*/

1: required bool overCapacity

/\*\*

\* An optional message about the cause of the failure.

\*/

2: optional string message

/\*\*

\* This field is populated when some tweet fields fail to load and the

\* tweet is marked "partial" in tweetypie. It contains the tweet/RT

\* information along with the set of tweet fields that failed to

\* get populated.

\*/

3: optional TweetFieldsPartial partial

}

/\*\*

\* Used in `TweetFieldsResultState` when the requested tweet has been filtered out.

\*/

struct TweetFieldsResultFiltered {

1: required safety\_result.FilteredReason reason

}

/\*\*

\* A union of the different possible outcomes of a fetching a single tweet.

\*/

union TweetFieldsResultState {

1: TweetFieldsResultFound found

2: TweetFieldsResultNotFound notFound

3: TweetFieldsResultFailed failed

4: TweetFieldsResultFiltered filtered

}

/\*\*

\* The response to get\_tweet\_fields will include a TweetFieldsResultRow for each

\* requested tweet id.

\*/

struct GetTweetFieldsResult {

/\*\*

\* The id of the requested tweet.

\*/

1: required i64 tweetId (personalDataType = 'TweetId')

/\*\*

\* the result for the requested tweet

\*/

2: required TweetFieldsResultState tweetResult

/\*\*

\* If quoted-tweets were requested and the primary tweet was found,

\* this field will contain the result state for the quoted tweeted.

\*/

3: optional TweetFieldsResultState quotedTweetResult

/\*\*

\* If the primary tweet was found, carousels were requested and there

\* was a carousel URL in the primary tweet, this field will contain the

\* result for the carousel.

\*

\* @deprecated

\*/

4: optional carousel\_service.GetCarouselResult carouselResult

}

struct TweetCreateConversationControlByInvitation {

1: optional bool invite\_via\_mention

}

struct TweetCreateConversationControlCommunity {

1: optional bool invite\_via\_mention

}

struct TweetCreateConversationControlFollowers {

1: optional bool invite\_via\_mention

}

/\*\*

\* Specify limits on user participation in a conversation.

\*

\* This is a union rather than a struct to support adding conversation

\* controls that require carrying metadata along with them, such as a list id.

\*

\* See also:

\* Tweet.conversation\_control

\* PostTweetRequest.conversation\_control

\*/

union TweetCreateConversationControl {

1: TweetCreateConversationControlCommunity community

2: TweetCreateConversationControlByInvitation byInvitation

3: TweetCreateConversationControlFollowers followers

}

/\*

\* Specifies the exclusivity of a tweet

\* This limits the audience of the tweet to the author

\* and the author's super followers

\* While empty now, we are expecting to add additional fields in v1+

\*/

struct ExclusiveTweetControlOptions {}

struct TrustedFriendsControlOptions {

1: i64 trusted\_friends\_list\_id = 0 (personalDataType = 'TrustedFriendsListMetadata')

}(hasPersonalData = 'true')

struct CollabInvitationOptions {

1: required list<i64> collaborator\_user\_ids (personalDataType = 'UserId')

// Note: status not sent here, will be added in TweetBuilder to set all but author as PENDING

}

struct CollabTweetOptions {

1: required list<i64> collaborator\_user\_ids (personalDataType = 'UserId')

}

union CollabControlOptions {

1: CollabInvitationOptions collabInvitation

2: CollabTweetOptions collabTweet

}

/\*\*

\* When this struct is supplied, this PostTweetRequest is interpreted as

\* an edit of the Tweet whose latest version is represented by previous\_tweet\_id.

\* If this is the first edit of a Tweet, this will be the same as the initial\_tweet\_id.

\*\*/

struct EditOptions {

/\*\*

\* The ID of the previous latest version of the Tweet that is being edited.

\* If this is the first edit, this will be the same as the initial\_tweet\_id.

\*\*/

1: required i64 previous\_tweet\_id (personalDataType = 'TweetId')

}

struct NoteTweetOptions {

/\*\*

\* The ID of the NoteTweet to be associated with this Tweet.

\*\*/

1: required i64 note\_tweet\_id (personalDataType = 'TwitterArticleID')

// Deprecated

2: optional list<string> mentioned\_screen\_names (personalDataType = 'Username')

/\*\*

\* The user IDs of the mentioned users

\*\*/

3: optional list<i64> mentioned\_user\_ids (personalDataType = 'UserId')

/\*\*

\* Specifies if the Tweet can be expanded into the NoteTweet, or if they have the same text

\*\*/

4: optional bool is\_expandable

}

struct PostTweetRequest {

/\*\*

\* Id of the user creating the tweet.

\*/

1: required i64 user\_id (personalDataType = 'UserId')

/\*\*

\* The user-supplied text of the tweet.

\*/

2: required string text (personalDataType = 'PrivateTweets, PublicTweets')

/\*\*

\* The OAuth client application from which the creation request originated.

\*

\* This must be in the format "oauth:<client application id>". For requests

\* from a user this is the application id of their client; for internal

\* services this is the id of an associated application registered at

\* https://apps.twitter.com.

\*/

3: required string created\_via (personalDataType = 'ClientType')

4: optional i64 in\_reply\_to\_tweet\_id (personalDataType = 'TweetId')

5: optional TweetCreateGeo geo

6: optional list<i64> media\_upload\_ids (personalDataType = 'MediaId')

7: optional tweet.Narrowcast narrowcast

/\*\*

\* Do not deliver this tweet to a user's followers.

\*

\* When true this tweet will not be fanned out, appear in the user's

\* timeline, or appear in search results. It will be distributed via the

\* firehose and available in the public API.

\*

\* This is primarily used to create tweets that can be used as ads without

\* broadcasting them to an advertiser's followers.

\*

\*/

8: bool nullcast = 0

/\*\*

\* The impression id of the ad from which this tweet was created.

\*

\* This is set when a user retweets or replies to a promoted tweet. It is

\* used to attribute the "earned" exposure of an advertisement.

\*/

9: optional i64 tracking\_id (personalDataType = 'ImpressionId')

/\*\*

\* @deprecated.

\* TOO clients don't actively use this input param, and the v2 API does not plan

\* to expose this parameter. The value associated with this field that's

\* stored with a tweet is obtained from the user's account preferences stored in

\* `User.safety.nsfw\_user`. (See go/user.thrift for more details on this field)

\*

\* Field indicates whether a individual tweet may contain objectionable content.

\*

\* If specified, tweet.core\_data.nsfw\_user will equal this value (otherwise,

\* tweet.core\_data.nsfw\_user will be set to user.nsfw\_user).

\*/

10: optional bool possibly\_sensitive

/\*\*

\* Do not save, index, fanout, or otherwise persist this tweet.

\*

\* When true, the tweet is validated, created, and returned but is not

\* persisted. This can be used for dark testing or pre-validating a tweet

\* scheduled for later creation.

\*/

11: bool dark = 0

/\*\*

\* IP address of the user making the request.

\*

\* This is used for logging certain kinds of actions, like attempting to

\* tweet malware urls.

\*/

12: optional string remote\_host (personalDataType = 'IpAddress')

/\*\*

\* Additional fields to write with this tweet.

\*

\* This Tweet object should contain only additional fields to write with

\* this tweet. Additional fields are tweet fields with id > 100. Set

\* tweet.id to be 0; the id will be generated by Tweetypie. Any other non-

\* additional fields set on this tweet will be considered an invalid

\* request.

\*

\*/

14: optional tweet.Tweet additional\_fields

15: optional WritePathHydrationOptions hydration\_options

// OBSOLETE 16: optional bool bypass\_rate\_limit\_for\_xfactor

/\*\*

\* ID to explicitly identify a creation request for the purpose of rejecting

\* duplicates.

\*

\* If two requests are received with the same uniqueness\_id, then they will

\* be considered duplicates of each other. This only applies for tweets

\* created within the same datacenter. This id should be a snowflake id so

\* that it's globally unique.

\*/

17: optional i64 uniqueness\_id (personalDataType = 'PrivateTweetEntitiesAndMetadata, PublicTweetEntitiesAndMetadata')

18: optional feature\_context.FeatureContext feature\_context

/\*\*

\* Passthrough data for Scarecrow that is used for safety checks.

\*/

19: optional safety\_meta\_data.SafetyMetaData safety\_meta\_data

// OBSOLETE 20: bool community\_narrowcast = 0

/\*\*

\* Toggle narrowcasting behavior for leading @mentions.

\*

\* If in\_reply\_to\_tweet\_id is not set:

\* - When this flag is true and the tweet text starts with a leading mention then the tweet

\* will be narrowcasted.

\*

\* If in\_reply\_to\_tweet\_id is set:

\* - If auto\_populate\_reply\_metadata is true

\* - Setting this flag to true will use the default narrowcast determination logic where

\* most replies will be narrowcast but some special-cases of self-replies will not.

\* - Setting this flag to false will disable narrowcasting and the tweet will be fanned out

\* to all the author's followers. Previously users prefixed their reply text with "." to

\* achieve this effect.

\* - If auto\_populate\_reply\_metadata is false, this flag will control whether a leading

\* mention in the tweet text will be narrowcast (true) or broadcast (false).

\*/

21: bool enable\_tweet\_to\_narrowcasting = 1

/\*\*

\* Automatically populate replies with leading mentions from tweet text.

\*/

22: bool auto\_populate\_reply\_metadata = 0

/\*\*

\* Metadata at the tweet-asset relationship level.

\*/

23: optional map<MediaCommon.MediaId, MediaInformation.UserDefinedProductMetadata> media\_metadata

/\*\*

\* An optional URL that identifies a resource that is treated as an attachment of the

\* the tweet, such as a quote-tweet permalink.

\*

\* When provided, it is appended to the end of the tweet text, but is not

\* included in the visible\_text\_range.

\*/

24: optional string attachment\_url (personalDataType = 'CardId, ShortUrl')

/\*\*

\* Pass-through information to be published in `TweetCreateEvent`.

\*

\* This data is not persisted by Tweetypie.

\*

\* @deprecated prefer transient\_context (see field 27) over this.

\*/

25: optional map<tweet.TweetCreateContextKey, string> additional\_context

/\*\*

\* Users to exclude from the automatic reply population behavior.

\*

\* When auto\_populate\_reply\_metadata is true, screen names appearing in the

\* mention prefix can be excluded by specifying a corresponding user id in

\* exclude\_reply\_user\_ids. Because the mention prefix must always include

\* the leading mention to preserve directed-at addressing for the in-reply-

\* to tweet author, attempting to exclude that user id will have no effect.

\* Specifying a user id not in the prefix will be silently ignored.

\*/

26: optional list<i64> exclude\_reply\_user\_ids (personalDataType = 'UserId')

/\*\*

\* Used to pass structured data to Tweetypie and tweet\_events eventbus

\* stream consumers. This data is not persisted by Tweetypie.

\*

\* If adding a new passthrough field, prefer this over additional\_context,

\* as this is structured data, while additional\_context is text data.

\*/

27: optional transient\_context.TransientCreateContext transient\_context

/\*\*

\* Composer flow used to create this tweet. Unless using the News Camera (go/newscamera)

\* flow, this should be `STANDARD`.

\*

\* When set to `CAMERA`, clients are expected to display the tweet with a different UI

\* to emphasize attached media.

\*/

28: optional tweet.ComposerSource composer\_source

/\*\*

\* present if we want to restrict replies to this tweet (go/dont-at-me-api)

\* - This gets converted to Tweet.conversation\_control and changes type

\* - This is only valid for conversation root tweets

\* - This applies to all replies to this tweet

\*/

29: optional TweetCreateConversationControl conversation\_control

// OBSOLETE 30: optional jiminy.CreateNudgeOptions nudge\_options

/\*\*

\* Provided if the client wants to have the tweet create evaluated for a nudge (e.g. to notify

\* the user that they are about to create a toxic tweet). Reference: go/docbird/jiminy

\*/

31: optional jiminy.CreateTweetNudgeOptions nudge\_options

/\*\*

\* Provided for correlating requests originating from REST endpoints and GraphQL endpoints.

\* Its presence or absence does not affect Tweet mutation. It used for validation

\* and debugging. The expected format is a 36 ASCII UUIDv4.

\* Please see API specification at go/graphql-tweet-mutations for more information.

\*/

32: optional string comparison\_id (personalDataType = 'UniversallyUniqueIdentifierUuid')

/\*\*

\* Options that determine the shape of an exclusive tweet's restrictions.

\* The existence of this object indicates that the tweet is intended to be an exclusive tweet

\* While this is an empty structure for now, it will have fields added to it later in later versions.

\*/

33: optional ExclusiveTweetControlOptions exclusiveTweetControlOptions

34: optional TrustedFriendsControlOptions trustedFriendsControlOptions

/\*\*

\* Provided if tweet data is backed up by a creative container, that at tweet hydration

\* time, tweetypie would delegate to creative container service.

\*

\* go/creatives-containers-tdd

\* Please note that this id is never publically shared with clients, its only used for

\* internal purposes.

\*/

35: optional i64 underlying\_creatives\_container\_id (personalDataType = 'TweetId')

/\*\*

\* Provided if tweet is a CollabTweet or a CollabInvitation, along with a list of Collaborators

\* which includes the original author.

\*

\* go/collab-tweets

\*\*/

36: optional CollabControlOptions collabControlOptions

/\*\*

\* When supplied, this PostTweetRequest is an edit. See [[EditOptions]] for more details.

\*\*/

37: optional EditOptions editOptions

/\*\*

\* When supplied, the NoteTweet specified is associated with the created Tweet.

\*\*/

38: optional NoteTweetOptions noteTweetOptions

} (hasPersonalData = 'true')

struct SetAdditionalFieldsRequest {

1: required tweet.Tweet additional\_fields

}

struct DeleteAdditionalFieldsRequest {

1: required list<i64> tweet\_ids (personalDataType = 'TweetId')

2: required list<FieldId> field\_ids

}(hasPersonalData = 'true')

struct DeleteTweetsRequest {

1: required list<i64> tweet\_ids (personalDataType = 'TweetId')

// DEPRECATED and moved to tweetypie\_internal.thrift's CascadedDeleteTweetsRequest

2: optional i64 cascaded\_from\_tweet\_id (personalDataType = 'TweetId')

3: optional tweet\_audit.AuditDeleteTweet audit\_passthrough

/\*\*

\* The id of the user initiating this request.

\*

\* It could be either the owner of the tweet or an admin. If not specified

\* we will use TwitterContext.userId.

\*/

4: optional i64 by\_user\_id (personalDataType = 'UserId')

/\*\*

\* Where these tweets are being deleted as part of a user erasure, the process

\* of deleting tweets belonging to deactivated accounts.

\*

\* This lets backends optimize processing of mass deletes of tweets from the

\* same user. Talk to the Tweetypie team before setting this flag.

\*/

5: bool is\_user\_erasure = 0

/\*\*

\* Id to compare with the user id of the tweets being deleted.

\*

\* This provides extra protection against accidental deletion of tweets.

\* This is required when is\_user\_erasure is true. If any of the tweets

\* specified in tweet\_ids do not match expected\_user\_id a

\* EXPECTED\_USER\_ID\_MISMATCH state will be returned.

\*/

6: optional i64 expected\_user\_id (personalDataType = 'UserId')

/\*\*

\* A bounced tweet is a tweet that has been found to violate Twitter Rules.

\* This is represented as a tweet with its bounce\_label field set.

\*

\* When the Tweet owner deletes their offending bounced tweet in the Bounced workflow, Bouncer

\* will submit a delete request with `is\_bounce\_delete` set to true. If the tweet(s) being deleted

\* have a bounce\_label set, this request results in the tweet transitioning into the

\* BounceDeleted state which means the tweet is partially deleted.

\*

\* Most of the normal tweet deletion side-effects occur but the tweet remains in a

\* few tflock graphs, tweet cache, and a Manhattan marker is added. Other than timelines services,

\* bounce deleted tweets are considered deleted and will return a StatusState.BounceDelete.

\*

\* After a defined grace period, tweets in this state will be fully deleted.

\*

\* If the tweet(s) being deleted do not have the bounce\_label set, they will be deleted as usual.

\*

\* Other than Bouncer, no service should use `is\_bounce\_delete` flag.

\*/

7: bool is\_bounce\_delete = 0

/\*\*

\* This is a unique identifier used in both the REST and GraphQL-dark

\* requests that will be used to correlate the GraphQL mutation requests to the REST requests

\* during a transition period when clients will be moving toward tweet creation via GraphQL.

\* See also, the "Comparison Testing" section at go/tweet-create-on-graphql-tdd for additional

\* context.

\*/

8: optional string comparison\_id (personalDataType = 'UniversallyUniqueIdentifierUuid')

/\*\*

\* When an edited tweet is deleted via daemons, we take a different action

\* than if it was deleted normally. If deleted normally, we delete the

\* initial tweet in the chain. When deleted via daemons, we delete the actual tweet.

\*/

9: optional bool cascaded\_edited\_tweet\_deletion

}(hasPersonalData = 'true')

struct DeleteTweetResult {

1: required i64 tweet\_id (personalDataType = 'TweetId')

2: required TweetDeleteState state

}(hasPersonalData = 'true')

struct UnretweetResult {

/\*\*

\* Id of the retweet that was deleted if a retweet could be found.

\*/

1: optional i64 tweet\_id (personalDataType = 'TweetId')

2: required TweetDeleteState state

}(hasPersonalData = 'true')

struct PostTweetResult {

1: required TweetCreateState state

/\*\*

\* The created tweet when state is OK.

\*/

2: optional tweet.Tweet tweet

/\*\*

\* The original tweet when state is OK and tweet is a retweet.

\*/

3: optional tweet.Tweet source\_tweet

/\*\*

\* The quoted tweet when state is OK and tweet is a quote tweet.

\*/

4: optional tweet.Tweet quoted\_tweet

/\*\*

\* The required user remediation from Scarecrow when state is BOUNCE.

\*/

5: optional bounce.Bounce bounce

/\*\*

\* Additional information when TweetCreateState is not OK.

\*

\* Not all failures provide a reason.

\*/

6: optional string failure\_reason

// OBSOLETE 7: optional jiminy.Nudge nudge

/\*\*

\* Returned when the state is NUDGE to indicate that the tweet has not been created, and that

\* the client should instead display the nudge to the user. Reference: go/docbird/jiminy

\*/

8: optional jiminy.TweetNudge nudge

} (persisted = "true", hasPersonalData = "true")

/\*\*

\* Specifies the cause of an AccessDenied error.

\*/

enum AccessDeniedCause {

// obsolete: INVALID\_CLIENT\_ID = 0,

// obsolete: DEPRECATED = 1,

USER\_DEACTIVATED = 2,

USER\_SUSPENDED = 3,

RESERVED\_4 = 4,

RESERVED\_5 = 5,

RESERVED\_6 = 6

}

/\*\*

\* AccessDenied error is returned by delete\_tweets endpoint when

\* by\_user\_id is suspended or deactivated.

\*/

exception AccessDenied {

1: required string message

2: optional AccessDeniedCause errorCause

}

struct UndeleteTweetRequest {

1: required i64 tweet\_id (personalDataType = 'TweetId')

2: optional WritePathHydrationOptions hydration\_options

/\*\*

\* Perform the side effects of undeletion even if the tweet is not deleted.

\*

\* This flag is useful if you know that the tweet is present in Manhattan

\* but is not undeleted with respect to other services.

\*/

3: optional bool force

}(hasPersonalData = 'true')

struct UndeleteTweetResponse {

1: required UndeleteTweetState state

2: optional tweet.Tweet tweet

}

struct EraseUserTweetsRequest {

1: required i64 user\_id (personalDataType = 'UserId')

}(hasPersonalData = 'true')

struct UnretweetRequest {

/\*\*

\* The id of the user who owns the retweet.

\*/

1: required i64 user\_id (personalDataType = 'UserId')

/\*\*

\* The source tweet that should be unretweeted.

\*/

2: required i64 source\_tweet\_id (personalDataType = 'TweetId')

/\*\*

\* This is a unique identifier used in both the REST and GraphQL-dark

\* requests that will be used to correlate the GraphQL mutation requests to the REST requests

\* during a transition period when clients will be moving toward tweet creation via GraphQL.

\* See also, the "Comparison Testing" section at go/tweet-create-on-graphql-tdd for additional

\* context.

\*/

3: optional string comparison\_id (personalDataType = 'UniversallyUniqueIdentifierUuid')

}(hasPersonalData = 'true')

struct GetDeletedTweetsRequest {

1: required list<i64> tweetIds (personalDataType = 'TweetId')

}(hasPersonalData = 'true')

struct GetDeletedTweetResult {

1: required i64 tweetId (personalDataType = 'TweetId')

2: required DeletedTweetState state

4: optional deleted\_tweet.DeletedTweet tweet

}(hasPersonalData = 'true')

/\*\*

\* Flushes tweets and/or their counts from cache.

\*

\* Typically will be used manually for testing or when a particular problem is

\* found that needs to be fixed by hand. Defaults to flushing both tweet

\* struct and associated counts.

\*/

struct FlushRequest {

1: required list<i64> tweet\_ids (personalDataType = 'TweetId')

2: bool flushTweets = 1

3: bool flushCounts = 1

}(hasPersonalData = 'true')

/\*\*

\* A request to retrieve counts for one or more tweets.

\*/

struct GetTweetCountsRequest {

1: required list<i64> tweet\_ids (personalDataType = 'TweetId')

2: bool include\_retweet\_count = 0

3: bool include\_reply\_count = 0

4: bool include\_favorite\_count = 0

5: bool include\_quote\_count = 0

6: bool include\_bookmark\_count = 0

}(hasPersonalData = 'true')

/\*\*

\* A response optionally indicating one or more counts for a tweet.

\*/

struct GetTweetCountsResult {

1: required i64 tweet\_id (personalDataType = 'TweetId')

2: optional i64 retweet\_count (personalDataType = 'CountOfPrivateRetweets, CountOfPublicRetweets')

3: optional i64 reply\_count (personalDataType = 'CountOfPrivateReplies, CountOfPublicReplies')

4: optional i64 favorite\_count (personalDataType = 'CountOfPrivateLikes, CountOfPublicLikes')

5: optional i64 quote\_count (personalDataType = 'CountOfPrivateRetweets, CountOfPublicRetweets')

6: optional i64 bookmark\_count (personalDataType = 'CountOfPrivateLikes')

}(hasPersonalData = 'true')

/\*\*

\* A request to increment the cached favorites count for a tweet.

\*

\* Negative values decrement the count. This request is automatically

\* replicated to other data centers.

\*/

struct IncrTweetFavCountRequest {

1: required i64 tweet\_id (personalDataType = 'TweetId')

2: required i32 delta (personalDataType = 'CountOfPrivateLikes, CountOfPublicLikes')

}(hasPersonalData = 'true')

/\*\*

\* A request to increment the cached bookmarks count for a tweet.

\*

\* Negative values decrement the count. This request is automatically

\* replicated to other data centers.

\*/

struct IncrTweetBookmarkCountRequest {

1: required i64 tweet\_id (personalDataType = 'TweetId')

2: required i32 delta (personalDataType = 'CountOfPrivateLikes')

}(hasPersonalData = 'true')

/\*\*

\* Request to scrub geolocation from 1 or more tweets, and replicates to other

\* data centers.

\*/

struct GeoScrub {

1: required list<i64> status\_ids (personalDataType = 'TweetId')

// OBSOLETE 2: bool write\_through = 1

3: bool hosebird\_enqueue = 0

4: i64 user\_id = 0 (personalDataType = 'UserId') // should always be set for hosebird enqueue

}(hasPersonalData = 'true')

/\*\*

\* Contains different indicators of a tweets "nsfw" status.

\*/

struct NsfwState {

1: required bool nsfw\_user

2: required bool nsfw\_admin

3: optional safety\_label.SafetyLabel nsfw\_high\_precision\_label

4: optional safety\_label.SafetyLabel nsfw\_high\_recall\_label

}

/\*\*

\* Interface to Tweetypie

\*/

service TweetService {

/\*\*

\* Performs a multi-get of tweets. This endpoint is geared towards fetching

\* tweets for the API, with many fields returned by default.

\*

\* The response list is ordered the same as the requested ids list.

\*/

list<GetTweetResult> get\_tweets(1: GetTweetsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs a multi-get of tweets. This endpoint is geared towards internal

\* processing that needs only specific subsets of the data.

\*

\* The response list is ordered the same as the requested ids list.

\*/

list<GetTweetFieldsResult> get\_tweet\_fields(1: GetTweetFieldsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Execute a {@link GetTweetCountsRequest} and return one or more {@link GetTweetCountsResult}

\*/

list<GetTweetCountsResult> get\_tweet\_counts(1: GetTweetCountsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Set/Update additional fields on an existing tweet

\*/

void set\_additional\_fields(1: SetAdditionalFieldsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Delete additional fields on a tweet

\*/

void delete\_additional\_fields(1: DeleteAdditionalFieldsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Creates and saves a tweet.

\*

\* URLs contained in the text will be shortened via Talon. Validations that are

\* handled by this endpoint include:

\*

\* - tweet length not greater than 140 display characters, after URL shortening;

\* - tweet is not a duplicate of a recently created tweet by the same user;

\* - user is not suspended or deactivated;

\* - text does not contain malware urls, as determined by talon;

\*

\* Checks that are not handled here that should be handled by the web API:

\* - oauth authentication;

\* - client application has narrowcasting/nullcasting privileges;

\*/

PostTweetResult post\_tweet(1: PostTweetRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Creates and saves a retweet.

\*

\* Validations that are handled by this endpoint include:

\*

\* - source tweet exists;

\* - source-tweet user exists and is not suspended or deactivated;

\* - source-tweet user is not blocking retweeter;

\* - user has not already retweeted the source tweet;

\*

\* Checks that are not handled here that should be handled by the web API:

\* - oauth authentication;

\* - client application has narrowcasting/nullcasting privileges;

\*/

PostTweetResult post\_retweet(1: RetweetRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Remove tweets. It removes all associated fields of the tweets in

\* cache and the persistent storage.

\*/

list<DeleteTweetResult> delete\_tweets(1: DeleteTweetsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error,

3: AccessDenied access\_denied)

/\*\*

\* Restore a deleted Tweet.

\*

\* Tweets exist in a soft-deleted state for N days during which they can be

\* restored by support agents following the internal restoration guidelines.

\* If the undelete succeeds, the Tweet is given similar treatment to a new

\* tweet e.g inserted into cache, sent to the timeline service, reindexed by

\* TFlock etc.

\*/

UndeleteTweetResponse undelete\_tweet(1: UndeleteTweetRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Add or remove takedown countries associated with a Tweet.

\*/

void takedown(1: TakedownRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Set or unset the nsfw\_admin and/or nsfw\_user bit of tweet.core\_data.

\*\*/

void update\_possibly\_sensitive\_tweet(1: UpdatePossiblySensitiveTweetRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error

)

/\*\*

\* Delete all tweets for a given user. Currently only called by Test User Service, but we

\* can also use it ad-hoc.

\*

\* Note: regular user erasure is handled by the EraseUserTweets daemon.

\*/

void erase\_user\_tweets(1: EraseUserTweetsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Unretweet a given tweet.

\*

\* There are two ways to unretweet:

\* - call deleteTweets() with the retweetId

\* - call unretweet() with the retweeter userId and sourceTweetId

\*

\* This is useful if you want to be able to undo a retweet without having to

\* keep track of a retweetId.

\*/

UnretweetResult unretweet(1: UnretweetRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Get tweet content and deletion times for soft-deleted tweets.

\*

\* The response list is ordered the same as the requested ids list.

\*/

list<GetDeletedTweetResult> get\_deleted\_tweets(1: GetDeletedTweetsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Execute a {@link FlushRequest}

\*/

void flush(1: FlushRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Execute an {@link IncrTweetFavCountRequest}

\*/

void incr\_tweet\_fav\_count(1: IncrTweetFavCountRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Execute an {@link IncrTweetBookmarkCountRequest}

\*/

void incr\_tweet\_bookmark\_count(1: IncrTweetBookmarkCountRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Delete location data from all of a user's tweets.

\*

\* This endpoint initiates the process of deleting the user's location data

\* from all of their tweets, as well as clearing the has\_geotagged\_statuses

\* flag of the user. This method returns as soon as the event is enqueued,

\* but the location data won't be scrubbed until the event is processed.

\* Usually the latency for the whole process to complete is small, but it

\* could take up to a couple of minutes if the user has a very large number

\* of tweets, or if the request gets backed up behind other requests that

\* need to scrub a large number of tweets.

\*

\* The event is processed by the Tweetypie geoscrub daemon.

\*

\*/

void delete\_location\_data(1: DeleteLocationDataRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Execute a {@link GeoScrub} request.

\*

\*/

void scrub\_geo(1: GeoScrub geo\_scrub) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

}