namespace java com.twitter.tweetypie.thriftjava

#@namespace scala com.twitter.tweetypie.thriftscala

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

include "com/twitter/expandodo/cards.thrift"

include "com/twitter/gizmoduck/user.thrift"

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

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

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

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

include "com/twitter/servo/cache/servo\_repo.thrift"

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

include "com/twitter/tweetypie/delete\_location\_data.thrift"

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

include "com/twitter/tweetypie/media\_entity.thrift"

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

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

include "com/twitter/tweetypie/stored\_tweet\_info.thrift"

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

typedef i16 FieldId

struct UserIdentity {

1: required i64 id

2: required string screen\_name

3: required string real\_name

# obsolete 4: bool deactivated = 0

# obsolete 5: bool suspended = 0

}

enum HydrationType {

MENTIONS = 1,

URLS = 2,

CACHEABLE\_MEDIA = 3,

QUOTED\_TWEET\_REF = 4,

REPLY\_SCREEN\_NAME = 5,

DIRECTED\_AT = 6,

CONTRIBUTOR = 7,

SELF\_THREAD\_INFO = 8

}

struct CachedTweet {

1: required tweet.Tweet tweet

// @obsolete 2: optional set<i16> included\_additional\_fields

3: set<HydrationType> completed\_hydrations = []

// Indicates that a tweet was deleted after being bounced for violating

// the Twitter Rules.

// When set to true, all other fields in CachedTweet are ignored.

4: optional bool is\_bounce\_deleted

// Indicates whether this tweet has safety labels stored in Strato.

// See com.twitter.tweetypie.core.TweetData.hasSafetyLabels for more details.

// @obsolete 5: optional bool has\_safety\_labels

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

struct MediaFaces {

1: required map<TweetMedia.MediaSizeType, list<MediaInformation.Face>> faces

}

enum AsyncWriteEventType {

INSERT = 1,

DELETE = 2,

UNDELETE = 3,

SET\_ADDITIONAL\_FIELDS = 4,

DELETE\_ADDITIONAL\_FIELDS = 5,

UPDATE\_POSSIBLY\_SENSITIVE\_TWEET = 6,

UPDATE\_TWEET\_MEDIA = 7,

TAKEDOWN = 8,

SET\_RETWEET\_VISIBILITY = 9

}

// an enum of actions that could happen in an async-write (insert or delete)

enum AsyncWriteAction {

HOSEBIRD\_ENQUEUE = 1

SEARCH\_ENQUEUE = 2

// obsolete MAIL\_ENQUEUE = 3

FANOUT\_DELIVERY = 4

// obsolete FACEBOOK\_ENQUEUE = 5

TWEET\_INDEX = 6

TIMELINE\_UPDATE = 7

CACHE\_UPDATE = 8

REPLICATION = 9

// obsolete MONORAIL\_EXPIRY\_ENQUEUE = 10

USER\_GEOTAG\_UPDATE = 11

// obsolete IBIS\_ENQUEUE = 12

EVENT\_BUS\_ENQUEUE = 13

// obsolete HOSEBIRD\_BINARY\_ENQUEUE = 14

TBIRD\_UPDATE = 15

RETWEETS\_DELETION = 16

GUANO\_SCRIBE = 17

MEDIA\_DELETION = 18

GEO\_SEARCH\_REQUEST\_ID = 19

SEARCH\_THRIFT\_ENQUEUE = 20

RETWEET\_ARCHIVAL\_ENQUEUE = 21

}

# This struct is scribed to test\_tweetypie\_failed\_async\_write after

# an async-write action has failed multiple retries

struct FailedAsyncWrite {

1: required AsyncWriteEventType event\_type

2: required AsyncWriteAction action

3: optional tweet.Tweet tweet

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

# This struct is scribed to test\_tweetypie\_detached\_retweets after

# attempting to read a retweet for which the source tweet has been deleted.

struct DetachedRetweet {

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

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

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

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

struct TweetCacheWrite {

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

// If the tweet id is a snowflake id, this is an offset since tweet creation.

// If it is not a snowflake id, then this is a Unix epoch time in

// milliseconds. (The idea is that for most tweets, this encoding will make

// it easier to see the interval between events and whether it occured soon

// acter tweet creation.)

2: required i64 timestamp (personalDataType = 'TransactionTimestamp')

3: required string action // One of "set", "add", "replace", "cas", "delete"

4: required servo\_repo.CachedValue cached\_value // Contains metadata about the cached value

5: optional CachedTweet cached\_tweet

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

struct AsyncInsertRequest {

12: required tweet.Tweet tweet

18: required user.User user

21: required i64 timestamp

// the cacheable version of tweet from field 12

29: required CachedTweet cached\_tweet

# 13: obsolete tweet.Tweet internal\_tweet

19: optional tweet.Tweet source\_tweet

20: optional user.User source\_user

// Used for quote tweet feature

22: optional tweet.Tweet quoted\_tweet

23: optional user.User quoted\_user

28: optional i64 parent\_user\_id

// Used for delivering the requestId of a geotagged tweet

24: optional string geo\_search\_request\_id

# 7: obsolete

# if not specified, all async insert actions are performed. if specified, only

# the specified action is performed; this is used for retrying specific actions

# that failed on a previous attempt.

10: optional AsyncWriteAction retry\_action

# 11: obsolete: bool from\_monorail = 0

# 14: obsolete

15: optional feature\_context.FeatureContext feature\_context

# 16: obsolete

# 17: obsolete

# 26: obsolete: optional tweet.Tweet debug\_tweet\_copy

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

30: optional transient\_context.TransientCreateContext transient\_context

// Used to check whether the same tweet has been quoted multiple

// times by a given user.

31: optional bool quoter\_has\_already\_quoted\_tweet

32: optional InitialTweetUpdateRequest initialTweetUpdateRequest

// User ids of users mentioned in note tweet. Used for tls events

33: optional list<i64> note\_tweet\_mentioned\_user\_ids

}

struct AsyncUpdatePossiblySensitiveTweetRequest {

1: required tweet.Tweet tweet

2: required user.User user

3: required i64 by\_user\_id

4: required i64 timestamp

5: optional bool nsfw\_admin\_change

6: optional bool nsfw\_user\_change

7: optional string note

8: optional string host

9: optional AsyncWriteAction action

}

struct AsyncUpdateTweetMediaRequest {

1: required i64 tweet\_id

2: required list<media\_entity.MediaEntity> orphaned\_media

3: optional AsyncWriteAction retry\_action

4: optional list<MediaCommon.MediaKey> media\_keys

}

struct AsyncSetAdditionalFieldsRequest {

1: required tweet.Tweet additional\_fields

3: required i64 timestamp

4: required i64 user\_id

2: optional AsyncWriteAction retry\_action

}

struct AsyncSetRetweetVisibilityRequest {

1: required i64 retweet\_id

// Whether to archive or unarchive(visible=true) the retweet\_id edge in the RetweetsGraph.

2: required bool visible

3: required i64 src\_id

5: required i64 retweet\_user\_id

6: required i64 source\_tweet\_user\_id

7: required i64 timestamp

4: optional AsyncWriteAction retry\_action

}

struct SetRetweetVisibilityRequest {

1: required i64 retweet\_id

// Whether to archive or unarchive(visible=true) the retweet\_id edge in the RetweetsGraph.

2: required bool visible

}

struct AsyncEraseUserTweetsRequest {

1: required i64 user\_id

3: required i64 flock\_cursor

4: required i64 start\_timestamp

5: required i64 tweet\_count

}

struct AsyncDeleteRequest {

4: required tweet.Tweet tweet

11: required i64 timestamp

2: optional user.User user

9: optional i64 by\_user\_id

12: optional tweet\_audit.AuditDeleteTweet audit\_passthrough

13: optional i64 cascaded\_from\_tweet\_id

# if not specified, all async-delete actions are performed. if specified, only

# the specified action is performed; this is used for retrying specific actions

# that failed on a previous attempt.

3: optional AsyncWriteAction retry\_action

5: bool delete\_media = 1

6: bool delete\_retweets = 1

8: bool scribe\_for\_audit = 1

15: bool is\_user\_erasure = 0

17: bool is\_bounce\_delete = 0

18: optional bool is\_last\_quote\_of\_quoter

19: optional bool is\_admin\_delete

}

struct AsyncUndeleteTweetRequest {

1: required tweet.Tweet tweet

3: required user.User user

4: required i64 timestamp

// the cacheable version of tweet from field 1

12: required CachedTweet cached\_tweet

# 2: obsolete tweet.Tweet internal\_tweet

5: optional AsyncWriteAction retry\_action

6: optional i64 deleted\_at

7: optional tweet.Tweet source\_tweet

8: optional user.User source\_user

9: optional tweet.Tweet quoted\_tweet

10: optional user.User quoted\_user

11: optional i64 parent\_user\_id

13: optional bool quoter\_has\_already\_quoted\_tweet

}

struct AsyncIncrFavCountRequest {

1: required i64 tweet\_id

2: required i32 delta

}

struct AsyncIncrBookmarkCountRequest {

1: required i64 tweet\_id

2: required i32 delta

}

struct AsyncDeleteAdditionalFieldsRequest {

6: required i64 tweet\_id

7: required list<i16> field\_ids

4: required i64 timestamp

5: required i64 user\_id

3: optional AsyncWriteAction retry\_action

}

// Used for both tweet and user takedowns.

// user will be None for user takedowns because user is only used when scribe\_for\_audit or

// eventbus\_enqueue are true, which is never the case for user takedown.

struct AsyncTakedownRequest {

1: required tweet.Tweet tweet

// Author of the tweet. Used when scribe\_for\_audit or eventbus\_enqueue are true which is the case

// for tweet takedown but not user takedown.

2: optional user.User user

// This field is the resulting list of takedown country codes on the tweet after the

// countries\_to\_add and countries\_to\_remove changes have been applied.

13: list<withholding.TakedownReason> takedown\_reasons = []

// This field is the list of takedown reaons to add to the tweet.

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

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

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

// This field determines whether or not Tweetypie should write takedown audits

// for this request to Guano.

6: required bool scribe\_for\_audit

// This field determines whether or not Tweetypie should enqueue a

// TweetTakedownEvent to EventBus and Hosebird for this request.

7: required bool eventbus\_enqueue

// This field is sent as part of the takedown audit that's written to Guano,

// and is not persisted with the takedown itself.

8: optional string audit\_note

// This field is the ID of the user who initiated the takedown. It is used

// when auditing the takedown in Guano. If unset, it will be logged as -1.

9: optional i64 by\_user\_id

// This field is the host where the request originated or the remote IP that

// is associated with the request. It is used when auditing the takedown in

// Guano. If unset, it will be logged as "<unknown>".

10: optional string host

11: optional AsyncWriteAction retry\_action

12: required i64 timestamp

}

struct SetTweetUserTakedownRequest {

1: required i64 tweet\_id

2: required bool has\_takedown

3: optional i64 user\_id

}

enum DataErrorCause {

UNKNOWN = 0

// Returned on set\_tweet\_user\_takedown when

// the SetTweetUserTakedownRequest.user\_id does not match the author

// of the tweet identified by SetTweetUserTakedownRequest.tweet\_id.

USER\_TWEET\_RELATIONSHIP = 1

}

/\*\*

\* DataError is returned for operations that perform data changes,

\* but encountered an inconsistency, and the operation cannot

\* be meaninfully performed.

\*/

exception DataError {

1: required string message

2: optional DataErrorCause errorCause

}

struct ReplicatedDeleteAdditionalFieldsRequest {

/\*\* is a map for backwards compatibility, but will only contain a single tweet id \*/

1: required map<i64, list<i16>> fields\_map

}

struct CascadedDeleteTweetRequest {

1: required i64 tweet\_id

2: required i64 cascaded\_from\_tweet\_id

3: optional tweet\_audit.AuditDeleteTweet audit\_passthrough

}

struct QuotedTweetDeleteRequest {

1: i64 quoting\_tweet\_id

2: i64 quoted\_tweet\_id

3: i64 quoted\_user\_id

}

struct QuotedTweetTakedownRequest {

1: i64 quoting\_tweet\_id

2: i64 quoted\_tweet\_id

3: i64 quoted\_user\_id

4: list<string> takedown\_country\_codes = []

5: list<withholding.TakedownReason> takedown\_reasons = []

}

struct ReplicatedInsertTweet2Request {

1: required CachedTweet cached\_tweet

// Used to check whether the same tweet has been quoted by a user.

2: optional bool quoter\_has\_already\_quoted\_tweet

3: optional InitialTweetUpdateRequest initialTweetUpdateRequest

}

struct ReplicatedDeleteTweet2Request {

1: required tweet.Tweet tweet

2: required bool is\_erasure

3: required bool is\_bounce\_delete

4: optional bool is\_last\_quote\_of\_quoter

}

struct ReplicatedSetRetweetVisibilityRequest {

1: required i64 src\_id

// Whether to archive or unarchive(visible=true) the retweet\_id edge in the RetweetsGraph.

2: required bool visible

}

struct ReplicatedUndeleteTweet2Request {

1: required CachedTweet cached\_tweet

2: optional bool quoter\_has\_already\_quoted\_tweet

}

struct GetStoredTweetsOptions {

1: bool bypass\_visibility\_filtering = 0

2: optional i64 for\_user\_id

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

}

struct GetStoredTweetsRequest {

1: required list<i64> tweet\_ids

2: optional GetStoredTweetsOptions options

}

struct GetStoredTweetsResult {

1: required stored\_tweet\_info.StoredTweetInfo stored\_tweet

}

struct GetStoredTweetsByUserOptions {

1: bool bypass\_visibility\_filtering = 0

2: bool set\_for\_user\_id = 0

3: optional i64 start\_time\_msec

4: optional i64 end\_time\_msec

5: optional i64 cursor

6: bool start\_from\_oldest = 0

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

}

struct GetStoredTweetsByUserRequest {

1: required i64 user\_id

2: optional GetStoredTweetsByUserOptions options

}

struct GetStoredTweetsByUserResult {

1: required list<stored\_tweet\_info.StoredTweetInfo> stored\_tweets

2: optional i64 cursor

}

/\* This is a request to update an initial tweet based on the creation of a edit tweet

\* initialTweetId: The tweet to be updated

\* editTweetId: The tweet being created, which is an edit of initialTweetId

\* selfPermalink: A self permalink for initialTweetId

\*/

struct InitialTweetUpdateRequest {

1: required i64 initialTweetId

2: required i64 editTweetId

3: optional tweet.ShortenedUrl selfPermalink

}

service TweetServiceInternal extends tweet\_service.TweetService {

/\*\*

\* Performs the async portion of TweetService.erase\_user\_tweets.

\* Only tweetypie itself can call this.

\*/

void async\_erase\_user\_tweets(1: AsyncEraseUserTweetsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.post\_tweet.

\* Only tweetypie itself can call this.

\*/

void async\_insert(1: AsyncInsertRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.delete\_tweets.

\* Only tweetypie itself can call this.

\*/

void async\_delete(1: AsyncDeleteRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.undelete\_tweet.

\* Only tweetypie itself can call this.

\*/

void async\_undelete\_tweet(1: AsyncUndeleteTweetRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.update\_possibly\_sensitive\_tweet.

\* Only tweetypie itself can call this.

\*/

void async\_update\_possibly\_sensitive\_tweet(1: AsyncUpdatePossiblySensitiveTweetRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.incr\_tweet\_fav\_count.

\* Only tweetypie itself can call this.

\*/

void async\_incr\_fav\_count(1: AsyncIncrFavCountRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.incr\_tweet\_bookmark\_count.

\* Only tweetypie itself can call this.

\*/

void async\_incr\_bookmark\_count(1: AsyncIncrBookmarkCountRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.set\_additional\_fields.

\* Only tweetypie itself can call this.

\*/

void async\_set\_additional\_fields(1: AsyncSetAdditionalFieldsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetServiceInternal.set\_retweet\_visibility.

\* Only tweetypie itself can call this.

\*/

void async\_set\_retweet\_visibility(1: AsyncSetRetweetVisibilityRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Set whether the specified retweet ID should be included in its source tweet's retweet count.

\* This endpoint is invoked from a tweetypie-daemon to adjust retweet counts for all tweets a

\* suspended or fraudulent (e.g. ROPO-'d) user has retweeted to disincentivize their false engagement.

\*/

void set\_retweet\_visibility(1: SetRetweetVisibilityRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.delete\_additional\_fields.

\* Only tweetypie itself can call this.

\*/

void async\_delete\_additional\_fields(1: AsyncDeleteAdditionalFieldsRequest field\_delete) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Performs the async portion of TweetService.takedown.

\* Only tweetypie itself can call this.

\*/

void async\_takedown(1: AsyncTakedownRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Update the tweet's takedown fields when a user is taken down.

\* Only tweetypie's UserTakedownChange daemon can call this.

\*/

void set\_tweet\_user\_takedown(1: SetTweetUserTakedownRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error,

3: DataError data\_error)

/\*\*

\* Cascade delete tweet is the logic for removing tweets that are detached

\* from their dependency which has been deleted. They are already filtered

\* out from serving, so this operation reconciles storage with the view

\* presented by Tweetypie.

\* This RPC call is delegated from daemons or batch jobs. Currently there

\* are two use-cases when this call is issued:

\* \* Deleting detached retweets after the source tweet was deleted.

\* This is done through RetweetsDeletion daemon and the

\* CleanupDetachedRetweets job.

\* \* Deleting edits of an initial tweet that has been deleted.

\* This is done by CascadedEditedTweetDelete daemon.

\* Note that, when serving the original delete request for an edit,

\* the initial tweet is only deleted, which makes all edits hidden.

\*/

void cascaded\_delete\_tweet(1: CascadedDeleteTweetRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Update the timestamp of the user's most recent request to delete

\* location data on their tweets. This does not actually remove the

\* geo information from the user's tweets, but it will prevent the geo

\* information for this user's tweets from being returned by

\* Tweetypie.

\*/

void scrub\_geo\_update\_user\_timestamp(1: delete\_location\_data.DeleteLocationData request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Look up tweets quoting a tweet that has been deleted and enqueue a compliance event.

\* Only tweetypie's QuotedTweetDelete daemon can call this.

\*\*/

void quoted\_tweet\_delete(1: QuotedTweetDeleteRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Look up tweets quoting a tweet that has been taken down and enqueue a compliance event.

\* Only tweetypie's QuotedTweetTakedown daemon can call this.

\*\*/

void quoted\_tweet\_takedown(1: QuotedTweetTakedownRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates TweetService.get\_tweet\_counts from another cluster.

\*/

void replicated\_get\_tweet\_counts(1: tweet\_service.GetTweetCountsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates TweetService.get\_tweet\_fields from another cluster.

\*/

void replicated\_get\_tweet\_fields(1: tweet\_service.GetTweetFieldsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates TweetService.get\_tweets from another cluster.

\*/

void replicated\_get\_tweets(1: tweet\_service.GetTweetsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.post\_tweet InsertTweet event from another cluster.

\* Note: v1 version of this endpoint previously just took a Tweet which is why it was replaced

\*/

void replicated\_insert\_tweet2(1: ReplicatedInsertTweet2Request request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.delete\_tweets DeleteTweet event from another cluster.

\*/

void replicated\_delete\_tweet2(1: ReplicatedDeleteTweet2Request request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.incr\_tweet\_fav\_count event from another cluster.

\*/

void replicated\_incr\_fav\_count(1: i64 tweet\_id, 2: i32 delta) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.incr\_tweet\_bookmark\_count event from another cluster.

\*/

void replicated\_incr\_bookmark\_count(1: i64 tweet\_id, 2: i32 delta) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetServiceInternal.set\_retweet\_visibility event from another cluster.

\*/

void replicated\_set\_retweet\_visibility(1: ReplicatedSetRetweetVisibilityRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.scrub\_geo from another cluster.

\*/

void replicated\_scrub\_geo(1: list<i64> tweet\_ids) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.set\_additional\_fields event from another cluster.

\*/

void replicated\_set\_additional\_fields(

1: tweet\_service.SetAdditionalFieldsRequest request

) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.delete\_additional\_fields event from another cluster.

\*/

void replicated\_delete\_additional\_fields(

1: ReplicatedDeleteAdditionalFieldsRequest request

) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.undelete\_tweet event from another cluster.

\* Note: v1 version of this endpoint previously just took a Tweet which is why it was replaced

\*/

void replicated\_undelete\_tweet2(1: ReplicatedUndeleteTweet2Request request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.takedown event from another cluster.

\*/

void replicated\_takedown(1: tweet.Tweet tweet) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Replicates a TweetService.update\_possibly\_sensitive\_tweet event from another cluster.

\*/

void replicated\_update\_possibly\_sensitive\_tweet(1: tweet.Tweet tweet) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Fetches hydrated Tweets and some metadata irrespective of the Tweets' state.

\*/

list<GetStoredTweetsResult> get\_stored\_tweets(1: GetStoredTweetsRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

/\*\*

\* Fetches hydrated Tweets and some metadata for a particular user, irrespective of the Tweets'

\* state.

\*/

GetStoredTweetsByUserResult get\_stored\_tweets\_by\_user(1: GetStoredTweetsByUserRequest request) throws (

1: exceptions.ClientError client\_error,

2: exceptions.ServerError server\_error)

}