namespace java com.twitter.search.earlybird.thrift

#@namespace scala com.twitter.search.earlybird.thriftscala

#@namespace strato com.twitter.search.earlybird

namespace py gen.twitter.search.earlybird

include "com/twitter/ads/adserver/adserver\_common.thrift"

include "com/twitter/search/common/caching/caching.thrift"

include "com/twitter/search/common/constants/query.thrift"

include "com/twitter/search/common/constants/search\_language.thrift"

include "com/twitter/search/common/conversation/conversation.thrift"

include "com/twitter/search/common/features/features.thrift"

include "com/twitter/search/common/indexing/status.thrift"

include "com/twitter/search/common/query/search.thrift"

include "com/twitter/search/common/ranking/ranking.thrift"

include "com/twitter/search/common/results/expansions.thrift"

include "com/twitter/search/common/results/highlight.thrift"

include "com/twitter/search/common/results/hit\_attribution.thrift"

include "com/twitter/search/common/results/hits.thrift"

include "com/twitter/search/common/results/social.thrift"

include "com/twitter/service/spiderduck/gen/metadata\_store.thrift"

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

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

include "com/twitter/escherbird/tweet\_annotation.thrift"

enum ThriftSearchRankingMode {

// good old realtime search mode

RECENCY = 0,

// new super fancy relevance ranking

RELEVANCE = 1,

DEPRECATED\_DISCOVERY = 2,

// top tweets ranking mode

TOPTWEETS = 3,

// results from accounts followed by the searcher

FOLLOWS = 4,

PLACE\_HOLDER5 = 5,

PLACE\_HOLDER6 = 6,

}

enum ThriftSearchResultType {

// it's a time-ordered result.

RECENCY = 0,

// it's a highly relevant tweet (aka top tweet).

RELEVANCE = 1,

// top tweet result type

POPULAR = 2,

// promoted tweets (ads)

PROMOTED = 3,

// relevance-ordered (as opposed to time-ordered) tweets generated from a variety of candidates

RELEVANCE\_ORDERED = 4,

PLACE\_HOLDER5 = 5,

PLACE\_HOLDER6 = 6,

}

enum ThriftSocialFilterType {

// filter only users that the searcher is directly following.

FOLLOWS = 0,

// filter only users that are in searcher's social circle of trust.

TRUSTED = 1,

// filter both follows and trusted.

ALL = 2,

PLACE\_HOLDER3 = 3,

PLACE\_HOLDER4 = 4,

}

enum ThriftTweetSource {

///// enums set by Earlybird

REALTIME\_CLUSTER = 1,

FULL\_ARCHIVE\_CLUSTER = 2,

REALTIME\_PROTECTED\_CLUSTER = 4,

///// enums set inside Blender

ADSERVER = 0,

// from top news search, only used in universal search

TOP\_NEWS = 3,

// special tweets included just for EventParrot.

FORCE\_INCLUDED = 5,

// from Content Recommender

// from topic to Tweet path

CONTENT\_RECS\_TOPIC\_TO\_TWEET = 6,

// used for hydrating QIG Tweets (go/qig)

QIG = 8,

// used for TOPTWEETS ranking mode

TOP\_TWEET = 9,

// used for experimental candidate sources

EXPERIMENTAL = 7,

// from Scanr service

SCANR = 10,

PLACE\_HOLDER11 = 11,

PLACE\_HOLDER12 = 12

}

enum NamedEntitySource {

TEXT = 0,

URL = 1,

PLACE\_HOLDER2 = 2,

PLACE\_HOLDER3 = 3,

PLACE\_HOLDER4 = 4,

}

enum ExperimentCluster {

EXP0 = 0, // Send requests to the earlybird-realtime-exp0 cluster

PLACE\_HOLDER1 = 1,

PLACE\_HOLDER2 = 2,

}

enum AudioSpaceState {

RUNNING = 0,

ENDED = 1,

PLACE\_HOLDER2 = 2,

PLACE\_HOLDER3 = 3,

PLACE\_HOLDER4 = 4,

PLACE\_HOLDER5 = 5,

}

// Contains all scoring and relevance-filtering related controls and options for Earlybird.

struct ThriftSearchRelevanceOptions {

// Next available field ID: 31 and note that 45 and 50 have been used already

2: optional bool filterDups = 0 // filter out duplicate search results

26: optional bool keepDupWithHigherScore = 1 // keep the duplicate tweet with the higher score

3: optional bool proximityScoring = 0 // whether to do proximity scoring or not

4: optional i32 maxConsecutiveSameUser // filter consecutive results from the same user

5: optional ranking.ThriftRankingParams rankingParams // composed by blender

// deprecated in favor of the maxHitsToProcess in CollectorParams

6: optional i32 maxHitsToProcess // when to early-terminate for relevance

7: optional string experimentName // what relevance experiment is running

8: optional string experimentBucket // what bucket the user is in; DDG defaults to hard-coded 'control'

9: optional bool interpretSinceId = 1 // whether to interpret since\_id operator

24: optional i32 maxHitsPerUser // Overrides ThriftSearchQuery.maxHitsPerUser

// only used by discovery for capping direct follow tweets

10: optional i32 maxConsecutiveDirectFollows

// Note - the orderByRelevance flag is critical to understanding how merging

// and trimming works in relevance mode in the search root.

//

// When orderByRelevance is true, results are trimmed in score-order. This means the

// client will get the top results from (maxHitsToProcess \* numHashPartitions) hits,

// ordered by score.

//

// When orderByRelevance is false, results are trimmed in id-order. This means the

// client will get the top results from an approximation of maxHitsToProcess hits

// (across the entire corpus). These results ordered by ID.

14: optional bool orderByRelevance = 0

// Max blending count for results returned due to from:user rewrites

16: optional i32 maxUserBlendCount

// The weight for proximity phrases generated while translating the serialized query to the

// lucene query.

19: optional double proximityPhraseWeight = 1.0

20: optional i32 proximityPhraseSlop = 255

// Override the weights of searchable fields.

// Negative weight means the the field is not enabled for search by default,

// but if it is (e.g., by annotation), the absolute value of the weight shall be

// used (if the annotation does not specify a weight).

21: optional map<string, double> fieldWeightMapOverride

// whether disable the coordination in the rewritten disjunction query, term query and phrase query

// the details can be found in LuceneVisitor

22: optional bool deprecated\_disableCoord = 0

// Root only. Returns all results seen by root to the client without trimming

// if set to true.

23: optional bool returnAllResults

// DEPRECATED: All v2 counters will be used explicitly in the scoring function and

// returned in their own field (in either metadata or feature map in response).

25: optional bool useEngagementCountersV2 = 0

// -------- PERSONALIZATION-RELATED RELEVANCE OPTIONS --------

// Take special care with these options when reasoning about caching.

// Deprecated in SEARCH-8616.

45: optional map<i32, double> deprecated\_topicIDWeights

// Collect hit attribution on queries and likedByUserIDFilter64-enhanced queries to

// get likedByUserIds list in metadata field.

// NOTE: this flag has no affect on fromUserIDFilter64.

50: optional bool collectFieldHitAttributions = 0

// Whether to collect all hits regardless of their score with RelevanceAllCollector.

27: optional bool useRelevanceAllCollector = 0

// Override features of specific tweets before the tweets are scored.

28: optional map<i64, features.ThriftSearchResultFeatures> perTweetFeaturesOverride

// Override features of all tweets from specific users before the tweets are scored.

29: optional map<i64, features.ThriftSearchResultFeatures> perUserFeaturesOverride

// Override features of all tweets before the tweets are scored.

30: optional features.ThriftSearchResultFeatures globalFeaturesOverride

}(persisted='true')

// Facets types that may have different ranking parameters.

enum ThriftFacetType {

DEFAULT = 0,

MENTIONS\_FACET = 1,

HASHTAGS\_FACET = 2,

// Deprecated in SEARCH-13708

DEPRECATED\_NAMED\_ENTITIES\_FACET = 3,

STOCKS\_FACET = 4,

VIDEOS\_FACET = 5,

IMAGES\_FACET = 6,

NEWS\_FACET = 7,

LANGUAGES\_FACET = 8,

SOURCES\_FACET = 9,

TWIMG\_FACET = 10,

FROM\_USER\_ID\_FACET = 11,

DEPRECATED\_TOPIC\_IDS\_FACET = 12,

RETWEETS\_FACET = 13,

LINKS\_FACET = 14,

PLACE\_HOLDER15 = 15,

PLACE\_HOLDER16 = 16,

}

struct ThriftSearchDebugOptions {

// Make earlybird only score and return tweets (specified by tweet id) here, regardless

// if they have a hit for the current query or not.

1: optional set<i64> statusIds;

// Assorted structures to pass in debug options.

2: optional map<string, string> stringMap;

3: optional map<string, double> valueMap;

4: optional list<double> valueList;

}(persisted='true')

// These options control what metadata will be returned by earlybird for each search result

// in the ThriftSearchResultMetadata struct. These options are currently mostly supported by

// AbstractRelevanceCollector and partially in SearchResultsCollector. Most are true by default to

// preserve backwards compatibility, but can be disabled as necessary to optimize searches returning

// many results (such as discover).

struct ThriftSearchResultMetadataOptions {

// If true, fills in the tweetUrls field in ThriftSearchResultMetadata.

// Populated by AbstractRelevanceCollector.

1: optional bool getTweetUrls = 1

// If true, fills in the resultLocation field in ThriftSearchResultMetadata.

// Populated by AbstractRelevanceCollector.

2: optional bool getResultLocation = 1

// Deprecated in SEARCH-8616.

3: optional bool deprecated\_getTopicIDs = 1

// If true, fills in the luceneScore field in ThriftSearchResultMetadata.

// Populated by LinearScoringFunction.

4: optional bool getLuceneScore = 0

// Deprecated but used to be for Offline feature values for static index

5: optional bool deprecated\_getExpFeatureValues = 0

// If true, will omit all features derivable from packedFeatures, and set packedFeatures

// instead.

6: optional bool deprecated\_usePackedFeatures = 0

// If true, fills sharedStatusId. For replies this is the in-reply-to status id and for

// retweets this is the retweet source status id.

// Also fills in the the isRetweet and isReply flags.

7: optional bool getInReplyToStatusId = 0

// If true, fills referencedTweetAuthorId. Also fills in the the isRetweet and isReply flags.

8: optional bool getReferencedTweetAuthorId = 0

// If true, fills media bits (video/vine/periscope/etc.)

9: optional bool getMediaBits = 0

// If true, will return all defined features in the packed features. This flag does not cover

// the above defined features.

10: optional bool getAllFeatures = 0

// If true, will return all features as ThriftSearchResultFeatures format.

11: optional bool returnSearchResultFeatures = 0

// If the client caches some features schemas, client can indicate its cache schemas through

// this field based on (version, checksum).

12: optional list<features.ThriftSearchFeatureSchemaSpecifier> featureSchemasAvailableInClient

// Specific feature IDs to return for recency requests. Populated in SearchResultFeatures.

// Values must be IDs of CSF fields from EarlybirdFieldConstants.

13: optional list<i32> requestedFeatureIDs

// If true, fills in the namedEntities field in ThriftSearchResultExtraMetadata

14: optional bool getNamedEntities = 0

// If true, fills in the entityAnnotations field in ThriftSearchResultExtraMetadata

15: optional bool getEntityAnnotations = 0

// If true, fills in the fromUserId field in the ThriftSearchResultExtraMetadata

16: optional bool getFromUserId = 0

// If true, fills in the spaces field in the ThriftSearchResultExtraMetadata

17: optional bool getSpaces = 0

18: optional bool getExclusiveConversationAuthorId = 0

}(persisted='true')

// ThriftSearchQuery describes an earlybird search request, which typically consists

// of these parts:

// - a query to retrieve hits

// - relevance options to score hits

// - a collector to collect hits and process into search results

// Note that this struct is used in both ThriftBlenderRequest and EarlybirdRequest.

// Most fields are not set when this struct is embedded in ThriftBlenderRequest, and

// are filled in by the blender before sending to earlybird.

struct ThriftSearchQuery {

// Next available field ID: 42

// -------- SECTION ZERO: THINGS USED ONLY BY THE BLENDER --------

// See SEARCHQUAL-2398

// These fields are used by the blender and clients of the blender, but not by earlybird.

// blender use only

// The raw un-parsed user search query.

6: optional string rawQuery(personalDataType = 'SearchQuery')

// blender use only

// Language of the rawQuery.

18: optional string queryLang(personalDataType = 'InferredLanguage')

// blender use only

// What page of results to return, indexed from 1.

7: optional i32 page = 1

// blender use only

// Number of results to skip (for pagination). Indexed from 0.

2: optional i32 deprecated\_resultOffset = 0

// -------- SECTION ONE: RETRIEVAL OPTIONS --------

// These options control the query that will be used to retrieve documents / hits.

// The parsed query tree, serialized to a string. Restricts the search results to

// tweets matching this query.

1: optional string serializedQuery(personalDataType = 'SearchQuery')

// Restricts the search results to tweets having this minimum tweep cred, out of 100.

5: optional i32 minTweepCredFilter = -1

// Restricts the search results to tweets from these users.

34: optional list<i64> fromUserIDFilter64(personalDataType = 'PrivateAccountsFollowing, PublicAccountsFollowing')

// Restricts the search results to tweets liked by these users.

40: optional list<i64> likedByUserIDFilter64(personalDataType = 'PrivateAccountsFollowing, PublicAccountsFollowing')

// If searchStatusIds are present, earlybird will ignore the serializedQuery completely

// and simply score each of searchStatusIds, also bypassing features like duplicate

// filtering and early termination.

// IMPORTANT: this means that it is possible to get scores equal to ScoringFunction.SKIP\_HIT,

// for results skipped by the scoring function.

31: optional set<i64> searchStatusIds

35: optional set<i64> deprecated\_eventClusterIdsFilter

41: optional map<string, list<i64>> namedDisjunctionMap

// -------- SECTION TWO: HIT COLLECTOR OPTIONS --------

// These options control what hits will be collected by the hit collector.

// Whether we want to collect and return per-field hit attributions is set in RelevanceOptions.

// See SEARCH-2784

// Number of results to return (after offset/page correction).

// This is ignored when searchStatusIds is set.

3: required i32 numResults

// Maximum number of hits to process by the collector.

// deprecated in favor of the maxHitsToProcess in CollectorParams

4: optional i32 maxHitsToProcess = 1000

// Collect hit counts for these time periods (in milliseconds).

30: optional list<i64> hitCountBuckets

// If set, earlybird will also return the facet labels of the specified facet fields

// in result tweets.

33: optional list<string> facetFieldNames

// Options controlling which search result metadata is returned.

36: optional ThriftSearchResultMetadataOptions resultMetadataOptions

// Collection related Params

38: optional search.CollectorParams collectorParams

// Whether to collect conversation IDs

39: optional bool collectConversationId = 0

// -------- SECTION THREE: RELEVANCE OPTIONS --------

// These options control relevance scoring and anti-gaming.

// Ranking mode (RECENCY means time-ordered ranking with no relevance).

8: optional ThriftSearchRankingMode rankingMode = ThriftSearchRankingMode.RECENCY

// Relevance scoring options.

9: optional ThriftSearchRelevanceOptions relevanceOptions

// Limits the number of hits that can be contributed by the same user, for anti-gaming.

// Set to -1 to disable the anti-gaming filter. This is ignored when searchStatusIds

// is set.

11: optional i32 maxHitsPerUser = 3

// Disables anti-gaming filter checks for any tweets that exceed this tweepcred.

12: optional i32 maxTweepcredForAntiGaming = 65

// -------- PERSONALIZATION-RELATED RELEVANCE OPTIONS --------

// Take special care with these options when reasoning about caching. All of these

// options, if set, will bypass the cache with the exception of uiLang which is the

// only form of personalization allowed for caching.

// User ID of searcher. This is used for relevance, and will be used for retrieval

// by the protected tweets index. If set, query will not be cached.

20: optional i64 searcherId(personalDataType = 'UserId')

// Bloom filter containing trusted user IDs. If set, query will not be cached.

10: optional binary trustedFilter(personalDataType = 'UserId')

// Bloom filter containing direct follow user IDs. If set, query will not be cached.

16: optional binary directFollowFilter(personalDataType = 'UserId, PrivateAccountsFollowing, PublicAccountsFollowing')

// UI language from the searcher's profile settings.

14: optional string uiLang(personalDataType = 'GeneralSettings')

// Confidence of the understandability of different languages for this user.

// uiLang field above is treated as a userlang with a confidence of 1.0.

28: optional map<search\_language.ThriftLanguage, double> userLangs(personalDataTypeKey = 'InferredLanguage')

// An alternative to fromUserIDFilter64 that relies on the relevance bloom filters

// for user filtering. Not currently used in production. Only supported for realtime

// searches.

// If set, earlybird expects both trustedFilter and directFollowFilter to also be set.

17: optional ThriftSocialFilterType socialFilterType

// -------- SECTION FOUR: DEBUG OPTIONS, FORGOTTEN FEATURES --------

// Earlybird search debug options.

19: optional ThriftSearchDebugOptions debugOptions

// Overrides the query time for debugging.

29: optional i64 timestampMsecs = 0

// Support for this feature has been removed and this field is left for backwards compatibility

// (and to detect improper usage by clients when it is set).

25: optional list<string> deprecated\_iterativeQueries

// Specifies a lucene query that will only be used if serializedQuery is not set,

// for debugging. Not currently used in production.

27: optional string luceneQuery(personalDataType = 'SearchQuery')

// This field is deprecated and is not used by earlybirds when processing the query.

21: optional i32 deprecated\_minDocsToProcess = 0

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

struct ThriftFacetLabel {

1: required string fieldName

2: required string label

// the number of times this facet has shown up in tweets with offensive words.

3: optional i32 offensiveCount = 0

// only filled for TWIMG facets

4: optional string nativePhotoUrl

}(persisted='true')

struct ThriftSearchResultGeoLocation {

1: optional double latitude(personalDataType = 'GpsCoordinates')

2: optional double longitude(personalDataType = 'GpsCoordinates')

3: optional double distanceKm

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

// Contains an expanded url and media type from the URL facet fields in earlybird.

// Note: thrift copied from status.thrift with unused fields renamed.

struct ThriftSearchResultUrl {

// Next available field ID: 6. Fields 2-4 removed.

// Note: this is actually the expanded url. Rename after deprecated fields are removed.

1: required string originalUrl

// Media type of the url.

5: optional metadata\_store.MediaTypes mediaType

}(persisted='true')

struct ThriftSearchResultNamedEntity {

1: required string canonicalName

2: required string entityType

3: required NamedEntitySource source

}(persisted='true')

struct ThriftSearchResultAudioSpace {

1: required string id

2: required AudioSpaceState state

}(persisted='true')

// Even more metadata

struct ThriftSearchResultExtraMetadata {

// Next available field ID: 49

1: optional double userLangScore

2: optional bool hasDifferentLang

3: optional bool hasEnglishTweetAndDifferentUILang

4: optional bool hasEnglishUIAndDifferentTweetLang

5: optional i32 quotedCount

6: optional double querySpecificScore

7: optional bool hasQuote

29: optional i64 quotedTweetId

30: optional i64 quotedUserId

31: optional search\_language.ThriftLanguage cardLang

8: optional i64 conversationId

9: optional bool isSensitiveContent

10: optional bool hasMultipleMediaFlag

11: optional bool profileIsEggFlag

12: optional bool isUserNewFlag

26: optional double authorSpecificScore

28: optional bool isComposerSourceCamera

// temporary V2 engagement counters, original ones in ThriftSearchResultMetadata has log()

// applied on them and then converted to int in Thrift, which is effectively a premature

// discretization. It doesn't affect the scoring inside Earlybird but for scoring and ML training

// outside earlybird, they were bad. These newly added ones stores a proper value of these

// counts. This also provides an easier transition to v2 counter when Earlybird is eventually

// ready to consume them from DL

// See SEARCHQUAL-9536, SEARCH-11181

18: optional i32 retweetCountV2

19: optional i32 favCountV2

20: optional i32 replyCountV2

// Tweepcred weighted version of various engagement counts

22: optional i32 weightedRetweetCount

23: optional i32 weightedReplyCount

24: optional i32 weightedFavCount

25: optional i32 weightedQuoteCount

// 2 bits - 0, 1, 2, 3+

13: optional i32 numMentions

14: optional i32 numHashtags

// 1 byte - 256 possible languages

15: optional i32 linkLanguage

// 6 bits - 64 possible values

16: optional i32 prevUserTweetEngagement

17: optional features.ThriftSearchResultFeatures features

// If the ThriftSearchQuery.likedByUserIdFilter64 and ThriftSearchRelevanceOptions.collectFieldHitAttributions

// fields are set, then this field will contain the list of all users in the query that liked this tweet.

// Otherwise, this field is not set.

27: optional list<i64> likedByUserIds

// Deprecated. See SEARCHQUAL-10321

21: optional double dopamineNonPersonalizedScore

32: optional list<ThriftSearchResultNamedEntity> namedEntities

33: optional list<tweet\_annotation.TweetEntityAnnotation> entityAnnotations

// Health model scores from HML

34: optional double toxicityScore // (go/toxicity)

35: optional double pBlockScore // (go/pblock)

36: optional double experimentalHealthModelScore1

37: optional double experimentalHealthModelScore2

38: optional double experimentalHealthModelScore3

39: optional double experimentalHealthModelScore4

40: optional i64 directedAtUserId

// Health model scores from HML (cont.)

41: optional double pSpammyTweetScore // (go/pspammytweet)

42: optional double pReportedTweetScore // (go/preportedtweet)

43: optional double spammyTweetContentScore // (go/spammy-tweet-content)

// it is populated by looking up user table and it is only available in archive earlybirds response

44: optional bool isUserProtected

45: optional list<ThriftSearchResultAudioSpace> spaces

46: optional i64 exclusiveConversationAuthorId

47: optional string cardUri

48: optional bool fromBlueVerifiedAccount(personalDataType = 'UserVerifiedFlag')

}(persisted='true')

// Some basic metadata about a search result. Useful for re-sorting, filtering, etc.

//

// NOTE: DO NOT ADD NEW FIELD!!

// Stop adding new fields to this struct, all new fields should go to

// ThriftSearchResultExtraMetadata (VM-1897), or there will be performance issues in production.

struct ThriftSearchResultMetadata {

// Next available field ID: 86

// -------- BASIC SCORING METADATA --------

// When resultType is RECENCY most scoring metadata will not be available.

1: required ThriftSearchResultType resultType

// Relevance score computed for this result.

3: optional double score

// True if the result was skipped by the scoring function. Only set when the collect-all

// results collector was used - in other cases skipped results are not returned.

// The score will be ScoringFunction.SKIP\_HIT when skipped is true.

43: optional bool skipped

// optionally a Lucene-style explanation for this result

5: optional string explanation

// -------- NETWORK-BASED SCORING METADATA --------

// Found the tweet in the trusted circle.

6: optional bool isTrusted

// Found the tweet in the direct follows.

8: optional bool isFollow

// True if the fromUserId of this tweet was whitelisted by the dup / antigaming filter.

// This typically indicates the result was from a tweet that matched a fromUserId query.

9: optional bool dontFilterUser

// -------- COMMON DOCUMENT METADATA --------

// User ID of the author. When isRetweet is true, this is the user ID of the retweeter

// and NOT that of the original tweet.

7: optional i64 fromUserId = 0

// When isRetweet (or packed features equivalent) is true, this is the status id of the

// original tweet. When isReply and getReplySource are true, this is the status id of the

// original tweet. In all other circumstances this is 0.

40: optional i64 sharedStatusId = 0

// When hasCard (or packed features equivalent) is true, this is one of SearchCardType.

49: optional i8 cardType = 0

// -------- EXTENDED DOCUMENT METADATA --------

// This is additional metadata from facet fields and column stride fields.

// Return of these fields is controlled by ThriftSearchResultMetadataOptions to

// allow for fine-grained control over when these fields are returned, as an

// optimization for searches returning a large quantity of results.

// Lucene component of the relevance score. Only returned when

// ThriftSearchResultMetadataOptions.getLuceneScore is true.

31: optional double luceneScore = 0.0

// Urls found in the tweet. Only returned when

// ThriftSearchResultMetadataOptions.getTweetUrls is true.

18: optional list<ThriftSearchResultUrl> tweetUrls

// Deprecated in SEARCH-8616.

36: optional list<i32> deprecated\_topicIDs

// Facets available in this tweet, this will only be filled if

// ThriftSearchQuery.facetFieldNames is set in the request.

22: optional list<ThriftFacetLabel> facetLabels

// The location of the result, and the distance to it from the center of the query

// location. Only returned when ThriftSearchResultMetadataOptions.getResultLocation is true.

35: optional ThriftSearchResultGeoLocation resultLocation

// Per field hit attribution.

55: optional hit\_attribution.FieldHitAttribution fieldHitAttribution

// whether this has geolocation\_type:geotag hit

57: optional bool geotagHit = 0

// the user id of the author of the source/referenced tweet (the tweet one replied

// to, retweeted and possibly quoted, etc.) (SEARCH-8561)

// Only returned when ThriftSearchResultMetadataOptions.getReferencedTweetAuthorId is true.

60: optional i64 referencedTweetAuthorId = 0

// Whether this tweet has certain types of media.

// Only returned when ThriftSearchResultMetadataOptions.getMediaBits is true.

// "Native video" is either consumer, pro, vine, or periscope.

// "Native image" is an image hosted on pic.twitter.com.

62: optional bool hasConsumerVideo

63: optional bool hasProVideo

64: optional bool hasVine

65: optional bool hasPeriscope

66: optional bool hasNativeVideo

67: optional bool hasNativeImage

// Packed features for this result. This field is never populated.

50: optional status.PackedFeatures deprecated\_packedFeatures

// The features stored in earlybird

// From integer 0 from EarlybirdFeatureConfiguration:

16: optional bool isRetweet

71: optional bool isSelfTweet

10: optional bool isOffensive

11: optional bool hasLink

12: optional bool hasTrend

13: optional bool isReply

14: optional bool hasMultipleHashtagsOrTrends

23: optional bool fromVerifiedAccount

// Static text quality score. This is actually an int between 0 and 100.

30: optional double textScore

51: optional search\_language.ThriftLanguage language

// From integer 1 from EarlybirdFeatureConfiguration:

52: optional bool hasImage

53: optional bool hasVideo

28: optional bool hasNews

48: optional bool hasCard

61: optional bool hasVisibleLink

// Tweep cred aka user rep. This is actually an int between 0 and 100.

32: optional double userRep

24: optional bool isUserSpam

25: optional bool isUserNSFW

26: optional bool isUserBot

54: optional bool isUserAntiSocial

// From integer 2 from EarlybirdFeatureConfiguration:

// Retweet, fav, reply, embeds counts, and video view counts are APPROXIMATE ONLY.

// Note that retweetCount, favCount and replyCount are not original unnormalized values,

// but after a log2() function for historical reason, this loses us some granularity.

// For more accurate counts, use {retweet, fav, reply}CountV2 in extraMetadata.

2: optional i32 retweetCount

33: optional i32 favCount

34: optional i32 replyCount

58: optional i32 embedsImpressionCount

59: optional i32 embedsUrlCount

68: optional i32 videoViewCount

// Parus score. This is actually an int between 0 and 100.

29: optional double parusScore

// Extra feature data, all new feature fields you want to return from Earlybird should go into

// this one, the outer one is always reaching its limit of the number of fields JVM can

// comfortably support!!

86: optional ThriftSearchResultExtraMetadata extraMetadata

// Integer 3 is omitted, see expFeatureValues above for more details.

// From integer 4 from EarlybirdFeatureConfiguration:

// Signature, for duplicate detection and removal.

4: optional i32 signature

// -------- THINGS USED ONLY BY THE BLENDER --------

// Social proof of the tweet, for network discovery.

// Do not use these fields outside of network discovery.

41: optional list<i64> retweetedUserIDs64

42: optional list<i64> replyUserIDs64

// Social connection between the search user and this result.

19: optional social.ThriftSocialContext socialContext

// used by RelevanceTimelineSearchWorkflow, whether a tweet should be highlighted or not

46: optional bool highlightResult

// used by RelevanceTimelineSearchWorkflow, the highlight context of the highlighted tweet

47: optional highlight.ThriftHighlightContext highlightContext

// the penguin version used to tokenize the tweets by the serving earlybird index as defined

// in com.twitter.common.text.version.PenguinVersion

56: optional i8 penguinVersion

69: optional bool isNullcast

// This is the normalized ratio(0.00 to 1.00) of nth token(starting before 140) divided by

// numTokens and then normalized into 16 positions(4 bits) but on a scale of 0 to 100% as

// we unnormalize it for you

70: optional double tokenAt140DividedByNumTokensBucket

}(persisted='true')

// Query level result stats.

// Next id: 20

struct ThriftSearchResultsRelevanceStats {

1: optional i32 numScored = 0

// Skipped documents count, they were also scored but their scores got ignored (skipped), note that this is different

// from numResultsSkipped in the ThriftSearchResults.

2: optional i32 numSkipped = 0

3: optional i32 numSkippedForAntiGaming = 0

4: optional i32 numSkippedForLowReputation = 0

5: optional i32 numSkippedForLowTextScore = 0

6: optional i32 numSkippedForSocialFilter = 0

7: optional i32 numSkippedForLowFinalScore = 0

8: optional i32 oldestScoredTweetAgeInSeconds = 0

// More counters for various features.

9: optional i32 numFromDirectFollows = 0

10: optional i32 numFromTrustedCircle = 0

11: optional i32 numReplies = 0

12: optional i32 numRepliesTrusted = 0

13: optional i32 numRepliesOutOfNetwork = 0

14: optional i32 numSelfTweets = 0

15: optional i32 numWithMedia = 0

16: optional i32 numWithNews = 0

17: optional i32 numSpamUser = 0

18: optional i32 numOffensive = 0

19: optional i32 numBot = 0

}(persisted='true')

// Per result debug info.

struct ThriftSearchResultDebugInfo {

1: optional string hostname

2: optional string clusterName

3: optional i32 partitionId

4: optional string tiername

}(persisted='true')

struct ThriftSearchResult {

// Next available field ID: 22

// Result status id.

1: required i64 id

// TweetyPie status of the search result

7: optional deprecated.Status tweetypieStatus

19: optional tweet.Tweet tweetypieTweet // v2 struct

// If the search result is a retweet, this field contains the source TweetyPie status.

10: optional deprecated.Status sourceTweetypieStatus

20: optional tweet.Tweet sourceTweetypieTweet // v2 struct

// If the search result is a quote tweet, this field contains the quoted TweetyPie status.

17: optional deprecated.Status quotedTweetypieStatus

21: optional tweet.Tweet quotedTweetypieTweet // v2 struct

// Additional metadata about a search result.

5: optional ThriftSearchResultMetadata metadata

// Hit highlights for various parts of this tweet

// for tweet text

6: optional list<hits.ThriftHits> hitHighlights

// for the title and description in the card expando.

12: optional list<hits.ThriftHits> cardTitleHitHighlights

13: optional list<hits.ThriftHits> cardDescriptionHitHighlights

// Expansion types, if expandResult == False, the expansions set should be ignored.

8: optional bool expandResult = 0

9: optional set<expansions.ThriftTweetExpansionType> expansions

// Only set if this is a promoted tweet

11: optional adserver\_common.AdImpression adImpression

// where this tweet is from

// Since ThriftSearchResult used not only as an Earlybird response, but also an internal

// data transfer object of Blender, the value of this field is mutable in Blender, not

// necessarily reflecting Earlybird response.

14: optional ThriftTweetSource tweetSource

// the features of a tweet used for relevance timeline

// this field is populated by blender in RelevanceTimelineSearchWorkflow

15: optional features.ThriftTweetFeatures tweetFeatures

// the conversation context of a tweet

16: optional conversation.ThriftConversationContext conversationContext

// per-result debugging info that's persisted across merges.

18: optional ThriftSearchResultDebugInfo debugInfo

}(persisted='true')

enum ThriftFacetRankingMode {

COUNT = 0,

FILTER\_WITH\_TERM\_STATISTICS = 1,

}

struct ThriftFacetFieldRequest {

// next available field ID: 4

1: required string fieldName

2: optional i32 numResults = 5

// use facetRankingOptions in ThriftFacetRequest instead

3: optional ThriftFacetRankingMode rankingMode = ThriftFacetRankingMode.COUNT

}(persisted='true')

struct ThriftFacetRequest {

// Next available field ID: 7

1: optional list<ThriftFacetFieldRequest> facetFields

5: optional ranking.ThriftFacetRankingOptions facetRankingOptions

6: optional bool usingQueryCache = 0

}(persisted='true')

struct ThriftTermRequest {

1: optional string fieldName = "text"

2: required string term

}(persisted='true')

enum ThriftHistogramGranularityType {

MINUTES = 0

HOURS = 1,

DAYS = 2,

CUSTOM = 3,

PLACE\_HOLDER4 = 4,

PLACE\_HOLDER5 = 5,

}

struct ThriftHistogramSettings {

1: required ThriftHistogramGranularityType granularity

2: optional i32 numBins = 60

3: optional i32 samplingRate = 1

4: optional i32 binSizeInSeconds // the bin size, only used if granularity is set to CUSTOM.

}(persisted='true')

// next id is 4

struct ThriftTermStatisticsRequest {

1: optional list<ThriftTermRequest> termRequests

2: optional ThriftHistogramSettings histogramSettings

// If this is set to true, even if there is no termRequests above, so long as the histogramSettings

// is set, Earlybird will return a null->ThriftTermResults entry in the termResults map, containing

// the global tweet count histogram for current query, which is the number of tweets matching this

// query in different minutes/hours/days.

3: optional bool includeGlobalCounts = 0

// When this is set, the background facets call does another search in order to find the best

// representative tweet for a given term request, the representative tweet is stored in the

// metadata of the termstats result

4: optional bool scoreTweetsForRepresentatives = 0

}(persisted='true')

// Next id is 12

struct ThriftFacetCountMetadata {

// this is the id of the first tweet in the index that contained this facet

1: optional i64 statusId = -1

// whether the tweet with the above statusId is NSFW, from an antisocial user,

// marked as sensitive content, etc.

10: optional bool statusPossiblySensitive

// the id of the user who sent the tweet above - only returned if

// statusId is returned too

// NOTE: for native photos we may not be able to determine the user,

// even though the statusId can be returned. This is because the statusId

// can be determined from the url, but the user can't and the tweet may

// not be in the index anymore. In this case statusId would be set but

// twitterUserId would not.

2: optional i64 twitterUserId = -1

// the language of the tweet above.

8: optional search\_language.ThriftLanguage statusLanguage

// optionally whitelist the fromUserId from dup/twitterUserId filtering

3: optional bool dontFilterUser = 0;

// if this facet is a native photo we return for convenience the

// twimg url

4: optional string nativePhotoUrl

// optionally returns some debug information about this facet

5: optional string explanation

// the created\_at value for the tweet from statusId - only returned

// if statusId is returned too

6: optional i64 created\_at

// the maximum tweepcred of the hits that contained this facet

7: optional i32 maxTweepCred

// Whether this facet result is force inserted, instead of organically returned from search.

// This field is only used in Blender to mark the force-inserted facet results

// (from recent tweets, etc).

11: optional bool forceInserted = 0

}(persisted='true')

struct ThriftTermResults {

1: required i32 totalCount

2: optional list<i32> histogramBins

3: optional ThriftFacetCountMetadata metadata

}(persisted='true')

struct ThriftTermStatisticsResults {

1: required map<ThriftTermRequest,ThriftTermResults> termResults

2: optional ThriftHistogramSettings histogramSettings

// If histogramSettings are set, this will have a list of ThriftHistogramSettings.numBins binIds,

// that the corresponding histogramBins in ThriftTermResults will have counts for.

// The binIds will correspond to the times of the hits matching the driving search query for this

// term statistics request.

// If there were no hits matching the search query, numBins binIds will be returned, but the

// values of the binIds will not meaningfully correspond to anything related to the query, and

// should not be used. Such cases can be identified by ThriftSearchResults.numHitsProcessed being

// set to 0 in the response, and the response not being early terminated.

3: optional list<i32> binIds

// If set, this id indicates the id of the minimum (oldest) bin that has been completely searched,

// even if the query was early terminated. If not set no bin was searched fully, or no histogram

// was requested.

// Note that if e.g. a query only matches a bin partially (due to e.g. a since operator) the bin

// is still considered fully searched if the query did not early terminate.

4: optional i32 minCompleteBinId

}(persisted='true')

struct ThriftFacetCount {

// the text of the facet

1: required string facetLabel

// deprecated; currently matches weightedCount for backwards-compatibility reasons

2: optional i32 facetCount

// the simple count of tweets that contained this facet, without any

// weighting applied

7: optional i32 simpleCount

// a weighted version of the count, using signals like tweepcred, parus, etc.

8: optional i32 weightedCount

// the number of times this facet occurred in tweets matching the background query

// using the term statistics API - only set if FILTER\_WITH\_TERM\_STATISTICS was used

3: optional i32 backgroundCount

// the relevance score that was computed for this facet if FILTER\_WITH\_TERM\_STATISTICS

// was used

4: optional double score

// a counter for how often this facet was penalized

5: optional i32 penaltyCount

6: optional ThriftFacetCountMetadata metadata

}(persisted='true')

// List of facet labels and counts for a given facet field, the

// total count for this field, and a quality score for this field

struct ThriftFacetFieldResults {

1: required list<ThriftFacetCount> topFacets

2: required i32 totalCount

3: optional double scoreQuality

4: optional i32 totalScore

5: optional i32 totalPenalty

// The ratio of the tweet language in the tweets with this facet field, a map from the language

// name to a number between (0.0, 1.0]. Only languages with ratio higher than 0.1 will be included.

6: optional map<search\_language.ThriftLanguage, double> languageHistogram

}

struct ThriftFacetResults {

1: required map<string, ThriftFacetFieldResults> facetFields

2: optional i32 backgroundNumHits

// returns optionally a list of user ids that should not get filtered

// out by things like antigaming filters, because these users were explicitly

// queried for

// Note that ThriftFacetCountMetadata returns already dontFilterUser

// for facet requests in which case this list is not needed. However, it

// is needed for subsequent term statistics queries, were user id lookups

// are performed, but a different background query is used.

3: optional set<i64> userIDWhitelist

}

struct ThriftSearchResults {

// Next available field ID: 23

1: required list<ThriftSearchResult> results = []

// (SEARCH-11950): Now resultOffset is deprecated, so there is no use in numResultsSkipped too.

9: optional i32 deprecated\_numResultsSkipped

// Number of docs that matched the query and were processed.

7: optional i32 numHitsProcessed

// Range of status IDs searched, from max ID to min ID (both inclusive).

// These may be unset in case that the search query contained ID or time

// operators that were completely out of range for the given index.

10: optional i64 maxSearchedStatusID

11: optional i64 minSearchedStatusID

// Time range that was searched (both inclusive).

19: optional i32 maxSearchedTimeSinceEpoch

20: optional i32 minSearchedTimeSinceEpoch

12: optional ThriftSearchResultsRelevanceStats relevanceStats

// Overall quality of this search result set

13: optional double score = -1.0

18: optional double nsfwRatio = 0.0

// The count of hit documents in each language.

14: optional map<search\_language.ThriftLanguage, i32> languageHistogram

// Hit counts per time period:

// The key is a time cutoff in milliseconds (e.g. 60000 msecs ago).

// The value is the number of hits that are more recent than the cutoff.

15: optional map<i64, i32> hitCounts

// the total cost for this query

16: optional double queryCost

// Set to non-0 if this query was terminated early (either due to a timeout, or exceeded query cost)

// When getting this response from a single earlybird, this will be set to 1, if the query

// terminated early.

// When getting this response from a search root, this should be set to the number of individual

// earlybird requests that were terminated early.

17: optional i32 numPartitionsEarlyTerminated

// If ThriftSearchResults returns features in features.ThriftSearchResultFeature format, this

// field would define the schema of the features.

// If the earlybird schema is already in the client cached schemas indicated in the request, then

// searchFeatureSchema would only have (version, checksum) information.

//

// Notice that earlybird root only sends one schema back to the superroot even though earlybird

// root might receive multiple version of schemas.

//

// Earlybird roots' schema merge/choose logic when returning results to superroot:

// . pick the most occurred versioned schema and return the schema to the superroot

// . if the superroot already caches the schema, only send the version information back

//

// Superroots' schema merge/choose logic when returning results to clients:

// . pick the schema based on the order of: realtime > protected > archive

// . because of the above ordering, it is possible that archive earlybird schema with a new flush

// version (with new bit features) might be lost to older realtime earlybird schema; this is

// considered to to be rare and acceptable because one realtime earlybird deploy would fix it

21: optional features.ThriftSearchFeatureSchema featureSchema

// How long it took to score the results in earlybird (in nanoseconds). The number of results

// that were scored should be set in numHitsProcessed.

// Expected to only be set for requests that actually do scoring (i.e. Relevance and TopTweets).

22: optional i64 scoringTimeNanos

8: optional i32 deprecated\_numDocsProcessed

}

// Note: Earlybird no longer respects this field, as it does not contain statuses.

// Blender should respect it.

enum EarlybirdReturnStatusType {

NO\_STATUS = 0

// deprecated

DEPRECATED\_BASIC\_STATUS = 1,

// deprecated

DEPRECATED\_SEARCH\_STATUS = 2,

TWEETYPIE\_STATUS = 3,

PLACE\_HOLDER4 = 4,

PLACE\_HOLDER5 = 5,

}

struct AdjustedRequestParams {

// Next available field ID: 4

// Adjusted value for EarlybirdRequest.searchQuery.numResults.

1: optional i32 numResults

// Adjusted value for EarlybirdRequest.searchQuery.maxHitsToProcess and

// EarlybirdRequest.searchQuery.relevanceOptions.maxHitsToProcess.

2: optional i32 maxHitsToProcess

// Adjusted value for EarlybirdRequest.searchQuery.relevanceOptions.returnAllResults

3: optional bool returnAllResults

}

struct EarlybirdRequest {

// Next available field ID: 36

// -------- COMMON REQUEST OPTIONS --------

// These fields contain options respected by all kinds of earlybird requests.

// Search query containing general earlybird retrieval and hit collection options.

// Also contains the options specific to search requests.

1: required ThriftSearchQuery searchQuery

// Common RPC information - client hostname and request ID.

12: optional string clientHost

13: optional string clientRequestID

// A string identifying the client that initiated the request.

// Ex: macaw-search.prod, webforall.prod, webforall.staging.

// The intention is to track the load we get from each client, and eventually enforce

// per-client QPS quotas, but this field could also be used to allow access to certain features

// only to certain clients, etc.

21: optional string clientId

// The time (in millis since epoch) when the earlybird client issued this request.

// Can be used to estimate request timeout time, capturing in-transit time for the request.

23: optional i64 clientRequestTimeMs

// Caching parameters used by earlybird roots.

24: optional caching.CachingParams cachingParams

// Deprecated. See SEARCH-2784

// Earlybird requests will be early terminated in a best-effort way to prevent them from

// exceeding the given timeout. If timeout is <= 0 this early termination criteria is

// disabled.

17: optional i32 timeoutMs = -1

// Deprecated. See SEARCH-2784

// Earlybird requests will be early terminated in a best-effort way to prevent them from

// exceeding the given query cost. If maxQueryCost <= 0 this early termination criteria

// is disabled.

20: optional double maxQueryCost = -1

// -------- REQUEST-TYPE SPECIFIC OPTIONS --------

// These fields contain options for one specific kind of request. If one of these options

// is set the request will be considered to be the appropriate type of request.

// Options for facet counting requests.

11: optional ThriftFacetRequest facetRequest

// Options for term statistics requests.

14: optional ThriftTermStatisticsRequest termStatisticsRequest

// -------- DEBUG OPTIONS --------

// Used for debugging only.

// Debug mode, 0 for no debug information.

15: optional i8 debugMode = 0

// Can be used to pass extra debug arguments to earlybird.

34: optional EarlybirdDebugOptions debugOptions

// Searches a specific segment by time slice id if set and segment id is > 0.

22: optional i64 searchSegmentId

// -------- THINGS USED ONLY BY THE BLENDER --------

// These fields are used by the blender and clients of the blender, but not by earlybird.

// Specifies what kind of status object to return, if any.

7: optional EarlybirdReturnStatusType returnStatusType

// -------- THINGS USED BY THE ROOTS --------

// These fields are not in use by earlybirds themselves, but are in use by earlybird roots

// (and their clients).

// These fields live here since we currently reuse the same thrift request and response structs

// for both earlybirds and earlybird roots, and could potentially be moved out if we were to

// introduce separate request / response structs specifically for the roots.

// We have a threshold for how many hash partition requests need to succeed at the root level

// in order for the earlybird root request to be considered successful.

// Each type or earlybird queries (e.g. relevance, or term statistics) has a predefined default

// threshold value (e.g. 90% or hash partitions need to succeed for a recency query).

// The client can optionally set the threshold value to be something other than the default,

// by setting this field to a value in the range of 0 (exclusive) to 1 (inclusive).

// If this value is set outside of the (0, 1] range, a CLIENT\_ERROR EarlybirdResponseCode will

// be returned.

25: optional double successfulResponseThreshold

// Where does the query come from?

26: optional query.ThriftQuerySource querySource

// Whether to get archive results This flag is advisory. A request may still be restricted from

// getting reqults from the archive based on the requesting client, query source, requested

// time/id range, etc.

27: optional bool getOlderResults

// The list of users followed by the current user.

// Used to restrict the values in the fromUserIDFilter64 field when sending a request

// to the protectected cluster.

28: optional list<i64> followedUserIds

// The adjusted parameters for the protected request.

29: optional AdjustedRequestParams adjustedProtectedRequestParams

// The adjusted parameters for the full archive request.

30: optional AdjustedRequestParams adjustedFullArchiveRequestParams

// Return only the protected tweets. This flag is used by the SuperRoot to return relevance

// results that contain only protected tweets.

31: optional bool getProtectedTweetsOnly

// Tokenize serialized queries with the appropriate Pengin version(s).

// Only has an effect on superroot.

32: optional bool retokenizeSerializedQuery

// Flag to ignore tweets that are very recent and could be incompletely indexed.

// If false, will allow queries to see results that may violate implicit streaming

// guarantees and will search Tweets that have been partially indexed.

// See go/indexing-latency for more details. When enabled, prevents seeing tweets

// that are less than 15 seconds old (or a similarly configured threshold).

// May be set to false unless explicitly set to true.

33: optional bool skipVeryRecentTweets = 1

// Setting an experimental cluster will reroute traffic at the realtime root layer to an experimental

// Earlybird cluster. This will have no impact if set on requests to anywhere other than realtime root.

35: optional ExperimentCluster experimentClusterToUse

// Caps number of results returned by roots after merging results from different earlybird partitions/clusters.

// If not set, ThriftSearchQuery.numResults or CollectorParams.numResultsToReturn will be used to cap results.

// This parameter will be ignored if ThriftRelevanceOptions.returnAllResults is set to true.

36: optional i32 numResultsToReturnAtRoot

}

enum EarlybirdResponseCode {

SUCCESS = 0,

PARTITION\_NOT\_FOUND = 1,

PARTITION\_DISABLED = 2,

TRANSIENT\_ERROR = 3,

PERSISTENT\_ERROR = 4,

CLIENT\_ERROR = 5,

PARTITION\_SKIPPED = 6,

// Request was queued up on the server for so long that it timed out, and was not

// executed at all.

SERVER\_TIMEOUT\_ERROR = 7,

TIER\_SKIPPED = 8,

// Not enough partitions returned a successful response. The merged response will have partition

// counts and early termination info set, but will not have search results.

TOO\_MANY\_PARTITIONS\_FAILED\_ERROR = 9,

// Client went over its quota, and the request was throttled.

QUOTA\_EXCEEDED\_ERROR = 10,

// Client's request is blocked based on Search Infra's policy. Search Infra can can block client's

// requests based on the query source of the request.

REQUEST\_BLOCKED\_ERROR = 11,

CLIENT\_CANCEL\_ERROR = 12,

CLIENT\_BLOCKED\_BY\_TIER\_ERROR = 13,

PLACE\_HOLDER\_2015\_09\_21 = 14,

}

// A recorded request and response.

struct EarlybirdRequestResponse {

// Where did we send this request to.

1: optional string sentTo;

2: optional EarlybirdRequest request;

// This can't be an EarlybirdResponse, because the thrift compiler for Python

// doesn't allow cyclic references and we have some Python utilities that will fail.

3: optional string response;

}

struct EarlybirdDebugInfo {

1: optional string host

2: optional string parsedQuery

3: optional string luceneQuery

// Requests sent to dependent services. For example, superroot sends to realtime root,

// archive root, etc.

4: optional list<EarlybirdRequestResponse> sentRequests;

// segment level debug info (eg. hitsPerSegment, max/minSearchedTime etc.)

5: optional list<string> collectorDebugInfo

6: optional list<string> termStatisticsDebugInfo

}

struct EarlybirdDebugOptions {

1: optional bool includeCollectorDebugInfo

}

struct TierResponse {

1: optional EarlybirdResponseCode tierResponseCode

2: optional i32 numPartitions

3: optional i32 numSuccessfulPartitions

}

struct EarlybirdServerStats {

// The hostname of the Earlybird that processed this request.

1: optional string hostname

// The partition to which this earlybird belongs.

2: optional i32 partition

// Current Earlybird QPS.

// Earlybirds should set this field at the end of a request (not at the start). This would give

// roots a more up-to-date view of the load on the earlybirds.

3: optional i64 currentQps

// The time the request waited in the queue before Earlybird started processing it.

// This does not include the time spent in the finagle queue: it's the time between the moment

// earlybird received the request, and the moment it started processing the request.

4: optional i64 queueTimeMillis

// The average request time in the queue before Earlybird started processing it.

// This does not include the time that requests spent in the finagle queue: it's the average time

// between the moment earlybird received its requests, and the moment it started processing them.

5: optional i64 averageQueueTimeMillis

// Current average per-request latency as perceived by Earlybird.

6: optional i64 averageLatencyMicros

// The tier to which this earlybird belongs.

7: optional string tierName

}

struct EarlybirdResponse {

// Next available field ID: 17

1: optional ThriftSearchResults searchResults

5: optional ThriftFacetResults facetResults

6: optional ThriftTermStatisticsResults termStatisticsResults

2: required EarlybirdResponseCode responseCode

3: required i64 responseTime

7: optional i64 responseTimeMicros

// fields below will only be returned if debug > 1 in the request.

4: optional string debugString

8: optional EarlybirdDebugInfo debugInfo

// Only exists for merged earlybird response.

10: optional i32 numPartitions

11: optional i32 numSuccessfulPartitions

// Only exists for merged earlybird response from multiple tiers.

13: optional list<TierResponse> perTierResponse

// Total number of segments that were searched. Partially searched segments are fully counted.

// e.g. if we searched 1 segment fully, and early terminated half way through the second

// segment, this field should be set to 2.

15: optional i32 numSearchedSegments

// Whether the request early terminated, if so, the termination reason.

12: optional search.EarlyTerminationInfo earlyTerminationInfo

// Whether this response is from cache.

14: optional bool cacheHit

// Stats used by roots to determine if we should go into degraded mode.

16: optional EarlybirdServerStats earlybirdServerStats

}

enum EarlybirdStatusCode {

STARTING = 0,

CURRENT = 1,

STOPPING = 2,

UNHEALTHY = 3,

BLACKLISTED = 4,

PLACE\_HOLDER5 = 5,

PLACE\_HOLDER6 = 6,

}

struct EarlybirdStatusResponse {

1: required EarlybirdStatusCode code

2: required i64 aliveSince

3: optional string message

}

service EarlybirdService {

string getName(),

EarlybirdStatusResponse getStatus(),

EarlybirdResponse search( 1: EarlybirdRequest request )

}