package com.twitter.search.common.schema.earlybird;

import java.io.IOException;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import javax.annotation.Nonnull;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.ImmutableSet;

import com.google.common.collect.Sets;

import org.apache.commons.lang.StringUtils;

import org.apache.lucene.analysis.TokenStream;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.collections.Pair;

import com.twitter.common.text.util.TokenStreamSerializer;

import com.twitter.cuad.ner.plain.thriftjava.NamedEntity;

import com.twitter.cuad.ner.plain.thriftjava.NamedEntityContext;

import com.twitter.cuad.ner.plain.thriftjava.NamedEntityInputSourceType;

import com.twitter.cuad.ner.thriftjava.WholeEntityType;

import com.twitter.search.common.constants.SearchCardType;

import com.twitter.search.common.indexing.thriftjava.ThriftExpandedUrl;

import com.twitter.search.common.indexing.thriftjava.ThriftGeoLocationSource;

import com.twitter.search.common.indexing.thriftjava.TwitterPhotoUrl;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.schema.ThriftDocumentBuilder;

import com.twitter.search.common.schema.base.FieldNameToIdMapping;

import com.twitter.search.common.schema.base.Schema;

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

import com.twitter.search.common.util.analysis.CharTermAttributeSerializer;

import com.twitter.search.common.util.analysis.IntTermAttributeSerializer;

import com.twitter.search.common.util.analysis.TermPayloadAttributeSerializer;

import com.twitter.search.common.util.analysis.TwitterPhotoTokenStream;

import com.twitter.search.common.util.spatial.GeoUtil;

import com.twitter.search.common.util.text.TokenizerHelper;

import com.twitter.search.common.util.text.TweetTokenStreamSerializer;

import com.twitter.search.common.util.text.regex.Regex;

import com.twitter.search.common.util.url.LinkVisibilityUtils;

import com.twitter.search.common.util.url.URLUtils;

import geo.google.datamodel.GeoAddressAccuracy;

import com.twitter.search.common.schema.thriftjava.ThriftDocument;

/\*\*

\* Builder class for building a {@link ThriftDocument}.

\*/

public final class EarlybirdThriftDocumentBuilder extends ThriftDocumentBuilder {

private static final Logger LOG = LoggerFactory.getLogger(EarlybirdThriftDocumentBuilder.class);

private static final SearchCounter SERIALIZE\_FAILURE\_COUNT\_NONPENGUIN\_DEPENDENT =

SearchCounter.export("tokenstream\_serialization\_failure\_non\_penguin\_dependent");

private static final String HASHTAG\_SYMBOL = "#";

private static final String CASHTAG\_SYMBOL = "$";

private static final String MENTION\_SYMBOL = "@";

private static final SearchCounter BCP47\_LANGUAGE\_TAG\_COUNTER =

SearchCounter.export("bcp47\_language\_tag");

/\*\*

\* Used to check if a card is video card.

\*

\* @see #withSearchCard

\*/

private static final String AMPLIFY\_CARD\_NAME = "amplify";

private static final String PLAYER\_CARD\_NAME = "player";

// Extra term indexed for native retweets, to ensure that the "-rt" query excludes them.

public static final String RETWEET\_TERM = "rt";

public static final String QUESTION\_MARK = "?";

private static final Set<NamedEntityInputSourceType> NAMED\_ENTITY\_URL\_SOURCE\_TYPES =

ImmutableSet.of(

NamedEntityInputSourceType.URL\_TITLE, NamedEntityInputSourceType.URL\_DESCRIPTION);

private final TokenStreamSerializer intTermAttributeSerializer =

new TokenStreamSerializer(ImmutableList.of(

new IntTermAttributeSerializer()));

private final TokenStreamSerializer photoUrlSerializer =

new TokenStreamSerializer(ImmutableList

.<TokenStreamSerializer.AttributeSerializer>of(

new CharTermAttributeSerializer(), new TermPayloadAttributeSerializer()));

private final Schema schema;

private boolean isSetLatLonCSF = false;

private boolean addLatLonCSF = true;

private boolean addEncodedTweetFeatures = true;

@Nonnull

private final EarlybirdEncodedFeatures encodedTweetFeatures;

@Nullable

private final EarlybirdEncodedFeatures extendedEncodedTweetFeatures;

/\*\*

\* Default constructor

\*/

public EarlybirdThriftDocumentBuilder(

@Nonnull EarlybirdEncodedFeatures encodedTweetFeatures,

@Nullable EarlybirdEncodedFeatures extendedEncodedTweetFeatures,

FieldNameToIdMapping idMapping,

Schema schema) {

super(idMapping);

this.schema = schema;

this.encodedTweetFeatures = Preconditions.checkNotNull(encodedTweetFeatures);

this.extendedEncodedTweetFeatures = extendedEncodedTweetFeatures;

}

/\*\*

\* Get internal {@link EarlybirdEncodedFeatures}

\*/

public EarlybirdEncodedFeatures getEncodedTweetFeatures() {

return encodedTweetFeatures;

}

/\*\*

\* Add skip list entry for the given field.

\* This adds a term \_\_has\_fieldName in the INTERNAL field.

\*/

public EarlybirdThriftDocumentBuilder addFacetSkipList(String fieldName) {

withStringField(EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstant.getFacetSkipFieldName(fieldName));

return this;

}

/\*\*

\* Add a filter term in the INTERNAL field.

\*/

public EarlybirdThriftDocumentBuilder addFilterInternalFieldTerm(String filterName) {

withStringField(EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdThriftDocumentUtil.formatFilter(filterName));

return this;

}

/\*\*

\* Add id field and id csf field.

\*/

public EarlybirdThriftDocumentBuilder withID(long id) {

withLongField(EarlybirdFieldConstant.ID\_FIELD.getFieldName(), id);

withLongField(EarlybirdFieldConstant.ID\_CSF\_FIELD.getFieldName(), id);

return this;

}

/\*\*

\* Add created at field and created at csf field.

\*/

public EarlybirdThriftDocumentBuilder withCreatedAt(int createdAt) {

withIntField(EarlybirdFieldConstant.CREATED\_AT\_FIELD.getFieldName(), createdAt);

withIntField(EarlybirdFieldConstant.CREATED\_AT\_CSF\_FIELD.getFieldName(), createdAt);

return this;

}

/\*\*

\* Add tweet text field.

\*/

public EarlybirdThriftDocumentBuilder withTweetText(

String text, byte[] textTokenStream) throws IOException {

withTokenStreamField(EarlybirdFieldConstants.EarlybirdFieldConstant.TEXT\_FIELD.getFieldName(),

text, textTokenStream);

return this;

}

public EarlybirdThriftDocumentBuilder withTweetText(String text) throws IOException {

withTweetText(text, null);

return this;

}

/\*\*

\* Add a list of cashTags. Like $TWTR.

\*/

public EarlybirdThriftDocumentBuilder withStocksFields(List<String> cashTags) {

if (isNotEmpty(cashTags)) {

addFacetSkipList(EarlybirdFieldConstant.STOCKS\_FIELD.getFieldName());

for (String cashTag : cashTags) {

withStringField(

EarlybirdFieldConstant.STOCKS\_FIELD.getFieldName(), CASHTAG\_SYMBOL + cashTag);

}

}

return this;

}

/\*\*

\* Add a list of hashtags.

\*/

public EarlybirdThriftDocumentBuilder withHashtagsField(List<String> hashtags) {

if (isNotEmpty(hashtags)) {

int numHashtags = Math.min(

hashtags.size(),

schema.getFeatureConfigurationById(

EarlybirdFieldConstant.NUM\_HASHTAGS.getFieldId()).getMaxValue());

encodedTweetFeatures.setFeatureValue(EarlybirdFieldConstant.NUM\_HASHTAGS, numHashtags);

addFacetSkipList(EarlybirdFieldConstant.HASHTAGS\_FIELD.getFieldName());

for (String hashtag : hashtags) {

withStringField(

EarlybirdFieldConstant.HASHTAGS\_FIELD.getFieldName(), HASHTAG\_SYMBOL + hashtag);

}

}

return this;

}

/\*\*

\* Added a list of mentions.

\*/

public EarlybirdThriftDocumentBuilder withMentionsField(List<String> mentions) {

if (isNotEmpty(mentions)) {

int numMentions = Math.min(

mentions.size(),

schema.getFeatureConfigurationById(

EarlybirdFieldConstant.NUM\_HASHTAGS.getFieldId()).getMaxValue());

encodedTweetFeatures.setFeatureValue(EarlybirdFieldConstant.NUM\_MENTIONS, numMentions);

addFacetSkipList(EarlybirdFieldConstant.MENTIONS\_FIELD.getFieldName());

for (String mention : mentions) {

withStringField(

EarlybirdFieldConstant.MENTIONS\_FIELD.getFieldName(), MENTION\_SYMBOL + mention);

}

}

return this;

}

/\*\*

\* Add a list of Twitter Photo URLs (twimg URLs). These are different from regular URLs, because

\* we use the TwitterPhotoTokenStream to index them, and we also include the status ID as payload.

\*/

public EarlybirdThriftDocumentBuilder withTwimgURLs(

List<TwitterPhotoUrl> urls) throws IOException {

if (isNotEmpty(urls)) {

for (TwitterPhotoUrl photoUrl : urls) {

TokenStream ts = new TwitterPhotoTokenStream(photoUrl.getPhotoStatusId(),

photoUrl.getMediaUrl());

byte[] serializedTs = photoUrlSerializer.serialize(ts);

withTokenStreamField(EarlybirdFieldConstant.TWIMG\_LINKS\_FIELD.getFieldName(),

Long.toString(photoUrl.getPhotoStatusId()), serializedTs);

addFacetSkipList(EarlybirdFieldConstant.TWIMG\_LINKS\_FIELD.getFieldName());

}

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.HAS\_IMAGE\_URL\_FLAG);

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.HAS\_NATIVE\_IMAGE\_FLAG);

}

return this;

}

/\*\*

\* Add a list of URLs. This also add facet skip list terms for news / images / videos if needed.

\*/

public EarlybirdThriftDocumentBuilder withURLs(List<ThriftExpandedUrl> urls) {

if (isNotEmpty(urls)) {

Set<String> dedupedLinks = Sets.newHashSet();

for (ThriftExpandedUrl expandedUrl : urls) {

if (expandedUrl.isSetOriginalUrl()) {

String normalizedOriginalUrl = URLUtils.normalizePath(expandedUrl.getOriginalUrl());

dedupedLinks.add(normalizedOriginalUrl);

}

if (expandedUrl.isSetExpandedUrl()) {

dedupedLinks.add(URLUtils.normalizePath(expandedUrl.getExpandedUrl()));

}

if (expandedUrl.isSetCanonicalLastHopUrl()) {

String url = URLUtils.normalizePath(expandedUrl.getCanonicalLastHopUrl());

dedupedLinks.add(url);

String facetUrl = URLUtils.normalizeFacetURL(url);

if (expandedUrl.isSetMediaType()) {

switch (expandedUrl.getMediaType()) {

case NEWS:

withStringField(EarlybirdFieldConstant.NEWS\_LINKS\_FIELD.getFieldName(), url);

addFacetSkipList(EarlybirdFieldConstant.NEWS\_LINKS\_FIELD.getFieldName());

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.HAS\_NEWS\_URL\_FLAG);

break;

case VIDEO:

withStringField(EarlybirdFieldConstant.VIDEO\_LINKS\_FIELD.getFieldName(), facetUrl);

addFacetSkipList(EarlybirdFieldConstant.VIDEO\_LINKS\_FIELD.getFieldName());

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.HAS\_VIDEO\_URL\_FLAG);

break;

case IMAGE:

withStringField(EarlybirdFieldConstant.IMAGE\_LINKS\_FIELD.getFieldName(), facetUrl);

addFacetSkipList(EarlybirdFieldConstant.IMAGE\_LINKS\_FIELD.getFieldName());

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.HAS\_IMAGE\_URL\_FLAG);

break;

case NATIVE\_IMAGE:

// Nothing done here. Native images are handled separately.

// They are in PhotoUrls instead of expandedUrls.

break;

case UNKNOWN:

break;

default:

throw new RuntimeException("Unknown Media Type: " + expandedUrl.getMediaType());

}

}

if (expandedUrl.isSetLinkCategory()) {

withIntField(EarlybirdFieldConstant.LINK\_CATEGORY\_FIELD.getFieldName(),

expandedUrl.getLinkCategory().getValue());

}

}

}

if (!dedupedLinks.isEmpty()) {

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.HAS\_LINK\_FLAG);

addFacetSkipList(EarlybirdFieldConstant.LINKS\_FIELD.getFieldName());

for (String linkUrl : dedupedLinks) {

withStringField(EarlybirdFieldConstant.LINKS\_FIELD.getFieldName(), linkUrl);

}

}

encodedTweetFeatures.setFlagValue(

EarlybirdFieldConstant.HAS\_VISIBLE\_LINK\_FLAG,

LinkVisibilityUtils.hasVisibleLink(urls));

}

return this;

}

/\*\*

\* Add a list of places. The place are U64 encoded place IDs.

\*/

public EarlybirdThriftDocumentBuilder withPlacesField(List<String> places) {

if (isNotEmpty(places)) {

for (String place : places) {

withStringField(EarlybirdFieldConstant.PLACE\_FIELD.getFieldName(), place);

}

}

return this;

}

/\*\*

\* Add tweet text signature field.

\*/

public EarlybirdThriftDocumentBuilder withTweetSignature(int signature) {

encodedTweetFeatures.setFeatureValue(EarlybirdFieldConstant.TWEET\_SIGNATURE, signature);

return this;

}

/\*\*

\* Add geo hash field and internal filter field.

\*/

public EarlybirdThriftDocumentBuilder withGeoHash(double lat, double lon, int accuracy) {

if (GeoUtil.validateGeoCoordinates(lat, lon)) {

withGeoField(

EarlybirdFieldConstant.GEO\_HASH\_FIELD.getFieldName(),

lat, lon, accuracy);

withLatLonCSF(lat, lon);

}

return this;

}

public EarlybirdThriftDocumentBuilder withGeoHash(double lat, double lon) {

withGeoHash(lat, lon, GeoAddressAccuracy.UNKNOWN\_LOCATION.getCode());

return this;

}

/\*\*

\* Add geo location source to the internal field with ThriftGeoLocationSource object.

\*/

public EarlybirdThriftDocumentBuilder withGeoLocationSource(

ThriftGeoLocationSource geoLocationSource) {

if (geoLocationSource != null) {

withGeoLocationSource(EarlybirdFieldConstants.formatGeoType(geoLocationSource));

}

return this;

}

/\*\*

\* Add geo location source to the internal field.

\*/

public EarlybirdThriftDocumentBuilder withGeoLocationSource(String geoLocationSource) {

withStringField(EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(), geoLocationSource);

return this;

}

/\*\*

\* Add encoded lat and lon to LatLonCSF field.

\*/

public EarlybirdThriftDocumentBuilder withLatLonCSF(double lat, double lon) {

isSetLatLonCSF = true;

long encodedLatLon = GeoUtil.encodeLatLonIntoInt64((float) lat, (float) lon);

withLongField(EarlybirdFieldConstant.LAT\_LON\_CSF\_FIELD.getFieldName(), encodedLatLon);

return this;

}

/\*\*

\* Add from verified account flag to internal field.

\*/

public EarlybirdThriftDocumentBuilder withFromVerifiedAccountFlag() {

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.FROM\_VERIFIED\_ACCOUNT\_FLAG);

addFilterInternalFieldTerm(EarlybirdFieldConstant.VERIFIED\_FILTER\_TERM);

return this;

}

/\*\*

\* Add from blue-verified account flag to internal field.

\*/

public EarlybirdThriftDocumentBuilder withFromBlueVerifiedAccountFlag() {

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.FROM\_BLUE\_VERIFIED\_ACCOUNT\_FLAG);

addFilterInternalFieldTerm(EarlybirdFieldConstant.BLUE\_VERIFIED\_FILTER\_TERM);

return this;

}

/\*\*

\* Add offensive flag to internal field.

\*/

public EarlybirdThriftDocumentBuilder withOffensiveFlag() {

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG);

withStringField(

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstant.IS\_OFFENSIVE);

return this;

}

/\*\*

\* Add user reputation value to encoded feature.

\*/

public EarlybirdThriftDocumentBuilder withUserReputation(byte score) {

encodedTweetFeatures.setFeatureValue(EarlybirdFieldConstant.USER\_REPUTATION, score);

return this;

}

/\*\*

\* This method creates the fields related to document language.

\* For most languages, their isoLanguageCode and bcp47LanguageTag are the same.

\* For some languages with variants, these two fields are different.

\* E.g. for simplified Chinese, their isoLanguageCode is zh, but their bcp47LanguageTag is zh-cn.

\* <p>

\* This method adds fields for both the isoLanguageCode and bcp47LanguageTag.

\*/

public EarlybirdThriftDocumentBuilder withLanguageCodes(

String isoLanguageCode, String bcp47LanguageTag) {

if (isoLanguageCode != null) {

withISOLanguage(isoLanguageCode);

}

if (bcp47LanguageTag != null && !bcp47LanguageTag.equals(isoLanguageCode)) {

BCP47\_LANGUAGE\_TAG\_COUNTER.increment();

withISOLanguage(bcp47LanguageTag);

}

return this;

}

/\*\*

\* Adds a String field into the ISO\_LANGUAGE\_FIELD.

\*/

public EarlybirdThriftDocumentBuilder withISOLanguage(String languageString) {

withStringField(

EarlybirdFieldConstant.ISO\_LANGUAGE\_FIELD.getFieldName(), languageString.toLowerCase());

return this;

}

/\*\*

\* Add from user ID fields.

\*/

public EarlybirdThriftDocumentBuilder withFromUserID(long fromUserId) {

withLongField(EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName(), fromUserId);

withLongField(EarlybirdFieldConstant.FROM\_USER\_ID\_CSF.getFieldName(), fromUserId);

return this;

}

/\*\*

\* Add from user information fields.

\*/

public EarlybirdThriftDocumentBuilder withFromUser(

long fromUserId, String fromUser) {

withFromUser(fromUserId, fromUser, null);

return this;

}

/\*\*

\* Add from user information fields.

\*/

public EarlybirdThriftDocumentBuilder withFromUser(String fromUser) {

withFromUser(fromUser, null);

return this;

}

/\*\*

\* Add from user information fields.

\*/

public EarlybirdThriftDocumentBuilder withFromUser(

String fromUser, String tokenizedFromUser) {

withStringField(EarlybirdFieldConstant.FROM\_USER\_FIELD.getFieldName(), fromUser);

withStringField(EarlybirdFieldConstant.TOKENIZED\_FROM\_USER\_FIELD.getFieldName(),

isNotBlank(tokenizedFromUser) ? tokenizedFromUser : fromUser);

return this;

}

/\*\*

\* Add from user information fields.

\*/

public EarlybirdThriftDocumentBuilder withFromUser(

long fromUserId, String fromUser, String tokenizedFromUser) {

withFromUserID(fromUserId);

withFromUser(fromUser, tokenizedFromUser);

return this;

}

/\*\*

\* Add to user field.

\*/

public EarlybirdThriftDocumentBuilder withToUser(

String toUser) {

withStringField(EarlybirdFieldConstant.TO\_USER\_FIELD.getFieldName(), toUser);

return this;

}

/\*\*

\* Add escherbird annotation fields.

\*/

public EarlybirdThriftDocumentBuilder withAnnotationEntities(List<String> entities) {

if (isNotEmpty(entities)) {

for (String entity : entities) {

withStringField(EarlybirdFieldConstant.ENTITY\_ID\_FIELD.getFieldName(), entity);

}

}

return this;

}

/\*\*

\* Add replies to internal field and set is reply flag.

\*/

public EarlybirdThriftDocumentBuilder withReplyFlag() {

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.IS\_REPLY\_FLAG);

addFilterInternalFieldTerm(EarlybirdFieldConstant.REPLIES\_FILTER\_TERM);

return this;

}

public EarlybirdThriftDocumentBuilder withCameraComposerSourceFlag() {

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.COMPOSER\_SOURCE\_IS\_CAMERA\_FLAG);

return this;

}

/\*\*

\* Add in reply to user id.

\* <p>

\* Notice {@link #withReplyFlag} is not automatically called since retweet a tweet that is

\* a reply to some other tweet is not considered a reply.

\* The caller should call {@link #withReplyFlag} separately if this tweet is really a reply tweet.

\*/

public EarlybirdThriftDocumentBuilder withInReplyToUserID(long inReplyToUserID) {

withLongField(EarlybirdFieldConstant.IN\_REPLY\_TO\_USER\_ID\_FIELD.getFieldName(), inReplyToUserID);

return this;

}

/\*\*

\* Add reference tweet author id.

\*/

public EarlybirdThriftDocumentBuilder withReferenceAuthorID(long referenceAuthorID) {

withLongField(EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_CSF.getFieldName(), referenceAuthorID);

return this;

}

/\*\*

\* Add all native retweet related fields/label

\*/

@VisibleForTesting

public EarlybirdThriftDocumentBuilder withNativeRetweet(final long retweetUserID,

final long sharedStatusID) {

withLongField(EarlybirdFieldConstant.SHARED\_STATUS\_ID\_CSF.getFieldName(), sharedStatusID);

withLongField(EarlybirdFieldConstant.RETWEET\_SOURCE\_TWEET\_ID\_FIELD.getFieldName(),

sharedStatusID);

withLongField(EarlybirdFieldConstant.RETWEET\_SOURCE\_USER\_ID\_FIELD.getFieldName(),

retweetUserID);

withLongField(EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_CSF.getFieldName(), retweetUserID);

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.IS\_RETWEET\_FLAG);

// Add native retweet label to the internal field.

addFilterInternalFieldTerm(EarlybirdFieldConstant.NATIVE\_RETWEETS\_FILTER\_TERM);

withStringField(EarlybirdFieldConstant.TEXT\_FIELD.getFieldName(), RETWEET\_TERM);

return this;

}

/\*\*

\* Add quoted tweet id and user id.

\*/

@VisibleForTesting

public EarlybirdThriftDocumentBuilder withQuote(

final long quotedStatusId, final long quotedUserId) {

withLongField(EarlybirdFieldConstant.QUOTED\_TWEET\_ID\_FIELD.getFieldName(), quotedStatusId);

withLongField(EarlybirdFieldConstant.QUOTED\_USER\_ID\_FIELD.getFieldName(), quotedUserId);

withLongField(EarlybirdFieldConstant.QUOTED\_TWEET\_ID\_CSF.getFieldName(), quotedStatusId);

withLongField(EarlybirdFieldConstant.QUOTED\_USER\_ID\_CSF.getFieldName(), quotedUserId);

encodedTweetFeatures.setFlag(EarlybirdFieldConstant.HAS\_QUOTE\_FLAG);

// Add quote label to the internal field.

addFilterInternalFieldTerm(EarlybirdFieldConstant.QUOTE\_FILTER\_TERM);

return this;

}

/\*\*

\* Add resolved links text field.

\*/

public EarlybirdThriftDocumentBuilder withResolvedLinksText(String linksText) {

withStringField(EarlybirdFieldConstant.RESOLVED\_LINKS\_TEXT\_FIELD.getFieldName(), linksText);

return this;

}

/\*\*

\* Add source field.

\*/

public EarlybirdThriftDocumentBuilder withSource(String source) {

withStringField(EarlybirdFieldConstant.SOURCE\_FIELD.getFieldName(), source);

return this;

}

/\*\*

\* Add normalized source field.

\*/

public EarlybirdThriftDocumentBuilder withNormalizedSource(String normalizedSource) {

withStringField(

EarlybirdFieldConstant.NORMALIZED\_SOURCE\_FIELD.getFieldName(), normalizedSource);

return this;

}

/\*\*

\* Add positive smiley to internal field.

\*/

public EarlybirdThriftDocumentBuilder withPositiveSmiley() {

withStringField(

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstant.HAS\_POSITIVE\_SMILEY);

return this;

}

/\*\*

\* Add negative smiley to internal field.

\*/

public EarlybirdThriftDocumentBuilder withNegativeSmiley() {

withStringField(

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstant.HAS\_NEGATIVE\_SMILEY);

return this;

}

/\*\*

\* Add question mark label to a text field.

\*/

public EarlybirdThriftDocumentBuilder withQuestionMark() {

withStringField(EarlybirdFieldConstant.TEXT\_FIELD.getFieldName(), QUESTION\_MARK);

return this;

}

/\*\*

\* Add card related fields.

\*/

public EarlybirdThriftDocumentBuilder withSearchCard(

String name,

String domain,

String title, byte[] serializedTitleStream,

String description, byte[] serializedDescriptionStream,

String lang) {

if (isNotBlank(title)) {

withTokenStreamField(

EarlybirdFieldConstants.EarlybirdFieldConstant.CARD\_TITLE\_FIELD.getFieldName(),

title, serializedTitleStream);

}

if (isNotBlank(description)) {

withTokenStreamField(

EarlybirdFieldConstants.EarlybirdFieldConstant.CARD\_DESCRIPTION\_FIELD.getFieldName(),

description, serializedDescriptionStream);

}

if (isNotBlank(lang)) {

withStringField(EarlybirdFieldConstant.CARD\_LANG.getFieldName(), lang);

}

if (isNotBlank(domain)) {

withStringField(

EarlybirdFieldConstants.EarlybirdFieldConstant.CARD\_DOMAIN\_FIELD.getFieldName(), domain);

}

if (isNotBlank(name)) {

withStringField(

EarlybirdFieldConstants.EarlybirdFieldConstant.CARD\_NAME\_FIELD.getFieldName(), name);

withIntField(

EarlybirdFieldConstants.EarlybirdFieldConstant.CARD\_TYPE\_CSF\_FIELD.getFieldName(),

SearchCardType.cardTypeFromStringName(name).getByteValue());

}

if (AMPLIFY\_CARD\_NAME.equalsIgnoreCase(name)

|| PLAYER\_CARD\_NAME.equalsIgnoreCase(name)) {

// Add into "internal" field so that this tweet is returned by filter:videos.

addFacetSkipList(

EarlybirdFieldConstants.EarlybirdFieldConstant.VIDEO\_LINKS\_FIELD.getFieldName());

}

return this;

}

public EarlybirdThriftDocumentBuilder withNormalizedMinEngagementField(

String fieldName, int normalizedNumEngagements) throws IOException {

EarlybirdThriftDocumentUtil.addNormalizedMinEngagementField(doc, fieldName,

normalizedNumEngagements);

return this;

}

/\*\*

\* Add named entity with given canonical name and type to document.

\*/

public EarlybirdThriftDocumentBuilder withNamedEntity(NamedEntity namedEntity) {

if (namedEntity.getContexts() == null) {

// In this unlikely case, we don't have any context for named entity type or source,

// so we can't properly index it in any of our fields. We'll just skip it in this case.

return this;

}

// Keep track of the fields we've applied in the builder already, to ensure we only index

// each term (field/value pair) once

Set<Pair<EarlybirdFieldConstant, String>> fieldsApplied = new HashSet<>();

for (NamedEntityContext context : namedEntity.getContexts()) {

if (context.isSetInput\_source()

&& NAMED\_ENTITY\_URL\_SOURCE\_TYPES.contains(context.getInput\_source().getSource\_type())) {

// If the source is one of the URL\* types, add the named entity to the "from\_url" fields,

// ensuring we add it only once

addNamedEntityFields(

fieldsApplied,

EarlybirdFieldConstant.NAMED\_ENTITY\_FROM\_URL\_FIELD,

EarlybirdFieldConstant.NAMED\_ENTITY\_WITH\_TYPE\_FROM\_URL\_FIELD,

namedEntity.getCanonical\_name(),

context);

} else {

addNamedEntityFields(

fieldsApplied,

EarlybirdFieldConstant.NAMED\_ENTITY\_FROM\_TEXT\_FIELD,

EarlybirdFieldConstant.NAMED\_ENTITY\_WITH\_TYPE\_FROM\_TEXT\_FIELD,

namedEntity.getCanonical\_name(),

context);

}

}

return this;

}

/\*\*

\* Add space id fields.

\*/

public EarlybirdThriftDocumentBuilder withSpaceIdFields(Set<String> spaceIds) {

if (!spaceIds.isEmpty()) {

addFacetSkipList(EarlybirdFieldConstant.SPACE\_ID\_FIELD.getFieldName());

for (String spaceId : spaceIds) {

withStringField(EarlybirdFieldConstant.SPACE\_ID\_FIELD.getFieldName(), spaceId);

}

}

return this;

}

/\*\*

\* Add directed at user.

\*/

@VisibleForTesting

public EarlybirdThriftDocumentBuilder withDirectedAtUser(final long directedAtUserId) {

withLongField(EarlybirdFieldConstant.DIRECTED\_AT\_USER\_ID\_FIELD.getFieldName(),

directedAtUserId);

withLongField(EarlybirdFieldConstant.DIRECTED\_AT\_USER\_ID\_CSF.getFieldName(), directedAtUserId);

return this;

}

/\*\*

\* Add a white space tokenized screen name field.

\*

\* Example:

\* screenName - "super\_hero"

\* tokenized version - "super hero"

\*/

public EarlybirdThriftDocumentBuilder withWhiteSpaceTokenizedScreenNameField(

String fieldName,

String normalizedScreenName) {

String whiteSpaceTokenizableScreenName = StringUtils.join(

normalizedScreenName.split(Regex.HASHTAG\_USERNAME\_PUNCTUATION\_REGEX), " ");

withStringField(fieldName, whiteSpaceTokenizableScreenName);

return this;

}

/\*\*

\* Add a camel case tokenized screen name field.

\*/

public EarlybirdThriftDocumentBuilder withCamelCaseTokenizedScreenNameField(

String fieldName,

String screenName,

String normalizedScreenName,

TokenStream screenNameTokenStream) {

// this normalized text is consistent to how the tokenized stream is created from

// TokenizerHelper.getNormalizedCamelcaseTokenStream - ie. just lowercasing.

String camelCaseTokenizedScreenNameText =

TokenizerHelper.getNormalizedCamelcaseTokenStreamText(screenName);

try {

// Reset the token stream in case it has been read before.

screenNameTokenStream.reset();

byte[] camelCaseTokenizedScreenName =

TweetTokenStreamSerializer.getTweetTokenStreamSerializer()

.serialize(screenNameTokenStream);

withTokenStreamField(

fieldName,

camelCaseTokenizedScreenNameText.isEmpty()

? normalizedScreenName : camelCaseTokenizedScreenNameText,

camelCaseTokenizedScreenName);

} catch (IOException e) {

LOG.error("TwitterTokenStream serialization error! Could not serialize: " + screenName);

SERIALIZE\_FAILURE\_COUNT\_NONPENGUIN\_DEPENDENT.increment();

}

return this;

}

private void addNamedEntityFields(

Set<Pair<EarlybirdFieldConstant, String>> fieldsApplied,

EarlybirdFieldConstant nameOnlyField,

EarlybirdFieldConstant nameWithTypeField,

String name,

NamedEntityContext context) {

withOneTimeStringField(fieldsApplied, nameOnlyField, name, false);

if (context.isSetEntity\_type()) {

withOneTimeStringField(fieldsApplied, nameWithTypeField,

formatNamedEntityString(name, context.getEntity\_type()), true);

}

}

private void withOneTimeStringField(

Set<Pair<EarlybirdFieldConstant, String>> fieldsApplied, EarlybirdFieldConstant field,

String value, boolean addToFacets) {

Pair<EarlybirdFieldConstant, String> fieldValuePair = new Pair<>(field, value);

if (!fieldsApplied.contains(fieldValuePair)) {

if (addToFacets) {

addFacetSkipList(field.getFieldName());

}

withStringField(field.getFieldName(), value);

fieldsApplied.add(fieldValuePair);

}

}

private String formatNamedEntityString(String name, WholeEntityType type) {

return String.format("%s:%s", name, type).toLowerCase();

}

/\*\*

\* Set whether set LAT\_LON\_CSF\_FIELD or not before build

\* if LAT\_LON\_CSF\_FIELD is not set deliberately.

\*

\* @see #prepareToBuild()

\*/

public EarlybirdThriftDocumentBuilder setAddLatLonCSF(boolean isSet) {

addLatLonCSF = isSet;

return this;

}

/\*\*

\* Set if add encoded tweet feature field in the end.

\*

\* @see #prepareToBuild()

\*/

public EarlybirdThriftDocumentBuilder setAddEncodedTweetFeatures(boolean isSet) {

addEncodedTweetFeatures = isSet;

return this;

}

@Override

protected void prepareToBuild() {

if (!isSetLatLonCSF && addLatLonCSF) {

// In lucene archives, this CSF is needed regardless of whether geoLocation is set.

withLatLonCSF(GeoUtil.ILLEGAL\_LATLON, GeoUtil.ILLEGAL\_LATLON);

}

if (addEncodedTweetFeatures) {

// Add encoded\_tweet\_features before building the document.

withBytesField(

EarlybirdFieldConstant.ENCODED\_TWEET\_FEATURES\_FIELD.getFieldName(),

EarlybirdEncodedFeaturesUtil.toBytesForThriftDocument(encodedTweetFeatures));

}

if (extendedEncodedTweetFeatures != null) {

// Add extended\_encoded\_tweet\_features before building the document.

withBytesField(

EarlybirdFieldConstant.EXTENDED\_ENCODED\_TWEET\_FEATURES\_FIELD.getFieldName(),

EarlybirdEncodedFeaturesUtil.toBytesForThriftDocument(extendedEncodedTweetFeatures));

}

}

private static boolean isNotBlank(String value) {

return value != null && !value.isEmpty();

}

private static boolean isNotEmpty(List<?> value) {

return value != null && !value.isEmpty();

}

}