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

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Set;

import javax.annotation.Nullable;

import com.google.common.base.Joiner;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import org.apache.commons.lang.StringUtils;

import org.apache.http.annotation.NotThreadSafe;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.text.token.TokenizedCharSequenceStream;

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

import com.twitter.common\_internal.text.version.PenguinVersion;

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

import com.twitter.decider.Decider;

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

import com.twitter.search.common.decider.DeciderUtil;

import com.twitter.search.common.indexing.thriftjava.SearchCard2;

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

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

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

import com.twitter.search.common.relevance.entities.TwitterMessage;

import com.twitter.search.common.relevance.entities.TwitterMessageUser;

import com.twitter.search.common.relevance.features.TweetTextFeatures;

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

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

import com.twitter.search.common.schema.earlybird.EarlybirdEncodedFeatures;

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

import com.twitter.search.common.schema.earlybird.EarlybirdThriftDocumentBuilder;

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

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

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

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

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

import com.twitter.search.common.util.lang.ThriftLanguageUtil;

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

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

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

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

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

import com.twitter.service.spiderduck.gen.MediaTypes;

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

/\*\*

\* Create and populate ThriftVersionedEvents from the URL data, card data, and named entities

\* contained in a TwitterMessage. This data is delayed because these services take a few seconds

\* to process tweets, and we want to send the basic data available in the BasicIndexingConverter as

\* soon as possible, so we send the additional data a few seconds later, as an update.

\*

\* Prefer to add data and processing to the BasicIndexingConverter when possible. Only add data here

\* if your data source \_requires\_ data from an external service AND the external service takes at

\* least a few seconds to process new tweets.

\*/

@NotThreadSafe

public class DelayedIndexingConverter {

private static final SearchCounter NUM\_TWEETS\_WITH\_CARD\_URL =

SearchCounter.export("tweets\_with\_card\_url");

private static final SearchCounter NUM\_TWEETS\_WITH\_NUMERIC\_CARD\_URI =

SearchCounter.export("tweets\_with\_numeric\_card\_uri");

private static final SearchCounter NUM\_TWEETS\_WITH\_INVALID\_CARD\_URI =

SearchCounter.export("tweets\_with\_invalid\_card\_uri");

private static final SearchCounter TOTAL\_URLS =

SearchCounter.export("total\_urls\_on\_tweets");

private static final SearchCounter MEDIA\_URLS\_ON\_TWEETS =

SearchCounter.export("media\_urls\_on\_tweets");

private static final SearchCounter NON\_MEDIA\_URLS\_ON\_TWEETS =

SearchCounter.export("non\_media\_urls\_on\_tweets");

public static final String INDEX\_URL\_DESCRIPTION\_AND\_TITLE\_DECIDER =

"index\_url\_description\_and\_title";

private static class ThriftDocumentWithEncodedTweetFeatures {

private final ThriftDocument document;

private final EarlybirdEncodedFeatures encodedFeatures;

public ThriftDocumentWithEncodedTweetFeatures(ThriftDocument document,

EarlybirdEncodedFeatures encodedFeatures) {

this.document = document;

this.encodedFeatures = encodedFeatures;

}

public ThriftDocument getDocument() {

return document;

}

public EarlybirdEncodedFeatures getEncodedFeatures() {

return encodedFeatures;

}

}

// The list of all the encoded\_tweet\_features flags that might be updated by this converter.

// No extended\_encoded\_tweet\_features are updated (otherwise they should be in this list too).

private static final List<EarlybirdFieldConstants.EarlybirdFieldConstant> UPDATED\_FLAGS =

Lists.newArrayList(

EarlybirdFieldConstants.EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_LINK\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.IS\_SENSITIVE\_CONTENT,

EarlybirdFieldConstants.EarlybirdFieldConstant.TEXT\_SCORE,

EarlybirdFieldConstants.EarlybirdFieldConstant.TWEET\_SIGNATURE,

EarlybirdFieldConstants.EarlybirdFieldConstant.LINK\_LANGUAGE,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_IMAGE\_URL\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_VIDEO\_URL\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_NEWS\_URL\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_EXPANDO\_CARD\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_MULTIPLE\_MEDIA\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_CARD\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_VISIBLE\_LINK\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_CONSUMER\_VIDEO\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_PRO\_VIDEO\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_VINE\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_PERISCOPE\_FLAG,

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_NATIVE\_IMAGE\_FLAG

);

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

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

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

private final EncodedFeatureBuilder featureBuilder = new EncodedFeatureBuilder();

private final Schema schema;

private final Decider decider;

public DelayedIndexingConverter(Schema schema, Decider decider) {

this.schema = schema;

this.decider = decider;

}

/\*\*

\* Converts the given message to two ThriftVersionedEvents instances: the first one is a feature

\* update event for all link and card related flags, and the second one is the append event that

\* might contain updates to all link and card related fields.

\*

\* We need to split the updates to fields and flags into two separate events because:

\* - When a tweet is created, earlybirds get the "main" event, which does not have resolved URLs.

\* - Then the earlybirds might get a feature update from the signal ingesters, marking the tweet

\* as spam.

\* - Then the ingesters resolve the URLs and send an update event. At this point, the ingesters

\* need to send updates for link-related flags too (HAS\_LINK\_FLAG, etc.). And there are a few

\* ways to do this:

\* 1. Encode these flags into encoded\_tweet\_features and extended\_encoded\_tweet\_features and

\* add these fields to the update event. The problem is that earlybirds will then override

\* the encoded\_tweet\_features ane extended\_encoded\_tweet\_features fields in the index for

\* this tweet, which will override the feature update the earlybirds got earlier, which

\* means that a spammy tweet might no longer be marked as spam in the index.

\* 2. Send updates only for the flags that might've been updated by this converter. Since

\* ThriftIndexingEvent already has a map of field -> value, it seems like the natural place

\* to add these updates to. However, earlybirds can correctly process flag updates only if

\* they come in a feature update event (PARTIAL\_UPDATE). So we need to send the field

\* updates in an OUT\_OF\_ORDER\_UPDATE event, and the flag updates in a PARTIAL\_UPDATE event.

\*

\* We need to send the feature update event before the append event to avoid issues like the one

\* in SEARCH-30919 where tweets were returned from the card name field index before the HAS\_CARD

\* feature was updated to true.

\*

\* @param message The TwitterMessage to convert.

\* @param penguinVersions The Penguin versions for which ThriftIndexingEvents should be created.

\* @return An out of order update event for all link- and card-related fields and a feature update

\* event for all link- and card-related flags.

\*/

public List<ThriftVersionedEvents> convertMessageToOutOfOrderAppendAndFeatureUpdate(

TwitterMessage message, List<PenguinVersion> penguinVersions) {

Preconditions.checkNotNull(message);

Preconditions.checkNotNull(penguinVersions);

ThriftVersionedEvents featureUpdateVersionedEvents = new ThriftVersionedEvents();

ThriftVersionedEvents outOfOrderAppendVersionedEvents = new ThriftVersionedEvents();

ImmutableSchemaInterface schemaSnapshot = schema.getSchemaSnapshot();

for (PenguinVersion penguinVersion : penguinVersions) {

ThriftDocumentWithEncodedTweetFeatures documentWithEncodedFeatures =

buildDocumentForPenguinVersion(schemaSnapshot, message, penguinVersion);

ThriftIndexingEvent featureUpdateThriftIndexingEvent = new ThriftIndexingEvent();

featureUpdateThriftIndexingEvent.setEventType(ThriftIndexingEventType.PARTIAL\_UPDATE);

featureUpdateThriftIndexingEvent.setUid(message.getId());

featureUpdateThriftIndexingEvent.setDocument(

buildFeatureUpdateDocument(documentWithEncodedFeatures.getEncodedFeatures()));

featureUpdateVersionedEvents.putToVersionedEvents(

penguinVersion.getByteValue(), featureUpdateThriftIndexingEvent);

ThriftIndexingEvent outOfOrderAppendThriftIndexingEvent = new ThriftIndexingEvent();

outOfOrderAppendThriftIndexingEvent.setDocument(documentWithEncodedFeatures.getDocument());

outOfOrderAppendThriftIndexingEvent.setEventType(ThriftIndexingEventType.OUT\_OF\_ORDER\_APPEND);

message.getFromUserTwitterId().ifPresent(outOfOrderAppendThriftIndexingEvent::setUid);

outOfOrderAppendThriftIndexingEvent.setSortId(message.getId());

outOfOrderAppendVersionedEvents.putToVersionedEvents(

penguinVersion.getByteValue(), outOfOrderAppendThriftIndexingEvent);

}

featureUpdateVersionedEvents.setId(message.getId());

outOfOrderAppendVersionedEvents.setId(message.getId());

return Lists.newArrayList(featureUpdateVersionedEvents, outOfOrderAppendVersionedEvents);

}

private ThriftDocument buildFeatureUpdateDocument(EarlybirdEncodedFeatures encodedFeatures) {

ThriftDocument document = new ThriftDocument();

for (EarlybirdFieldConstants.EarlybirdFieldConstant flag : UPDATED\_FLAGS) {

ThriftField field = new ThriftField();

field.setFieldConfigId(flag.getFieldId());

field.setFieldData(new ThriftFieldData().setIntValue(encodedFeatures.getFeatureValue(flag)));

document.addToFields(field);

}

return document;

}

private ThriftDocumentWithEncodedTweetFeatures buildDocumentForPenguinVersion(

ImmutableSchemaInterface schemaSnapshot,

TwitterMessage message,

PenguinVersion penguinVersion) {

EarlybirdEncodedFeatures encodedFeatures = featureBuilder.createTweetFeaturesFromTwitterMessage(

message, penguinVersion, schemaSnapshot).encodedFeatures;

EarlybirdThriftDocumentBuilder builder = new EarlybirdThriftDocumentBuilder(

encodedFeatures,

null,

new EarlybirdFieldConstants(),

schemaSnapshot);

builder.setAddLatLonCSF(false);

builder.withID(message.getId());

buildFieldsFromUrlInfo(builder, message, penguinVersion, encodedFeatures);

buildCardFields(builder, message, penguinVersion);

buildNamedEntityFields(builder, message);

builder.withTweetSignature(message.getTweetSignature(penguinVersion));

buildSpaceAdminAndTitleFields(builder, message, penguinVersion);

builder.setAddEncodedTweetFeatures(false);

return new ThriftDocumentWithEncodedTweetFeatures(builder.build(), encodedFeatures);

}

public static void buildNamedEntityFields(

EarlybirdThriftDocumentBuilder builder, TwitterMessage message) {

for (NamedEntity namedEntity : message.getNamedEntities()) {

builder.withNamedEntity(namedEntity);

}

}

private void buildFieldsFromUrlInfo(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message,

PenguinVersion penguinVersion,

EarlybirdEncodedFeatures encodedFeatures) {

// We need to update the RESOLVED\_LINKS\_TEXT\_FIELD, since we might have new resolved URLs.

// Use the same logic as in EncodedFeatureBuilder.java.

TweetTextFeatures textFeatures = message.getTweetTextFeatures(penguinVersion);

String resolvedUrlsText = Joiner.on(" ").skipNulls().join(textFeatures.getResolvedUrlTokens());

builder.withResolvedLinksText(resolvedUrlsText);

buildURLFields(builder, message, encodedFeatures);

buildAnalyzedURLFields(builder, message, penguinVersion);

}

private void buildAnalyzedURLFields(

EarlybirdThriftDocumentBuilder builder, TwitterMessage message, PenguinVersion penguinVersion

) {

TOTAL\_URLS.add(message.getExpandedUrls().size());

if (DeciderUtil.isAvailableForRandomRecipient(

decider,

INDEX\_URL\_DESCRIPTION\_AND\_TITLE\_DECIDER)) {

for (ThriftExpandedUrl expandedUrl : message.getExpandedUrls()) {

/\*

Consumer Media URLs are added to the expanded URLs in

TweetEventParserHelper.addMediaEntitiesToMessage. These Twitter.com media URLs contain

the tweet text as the description and the title is "<User Name> on Twitter". This is

redundant information at best and misleading at worst. We will ignore these URLs to avoid

polluting the url\_description and url\_title field as well as saving space.

\*/

if (!expandedUrl.isSetConsumerMedia() || !expandedUrl.isConsumerMedia()) {

NON\_MEDIA\_URLS\_ON\_TWEETS.increment();

if (expandedUrl.isSetDescription()) {

buildTweetTokenizerTokenizedField(builder,

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

expandedUrl.getDescription(),

penguinVersion);

}

if (expandedUrl.isSetTitle()) {

buildTweetTokenizerTokenizedField(builder,

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

expandedUrl.getTitle(),

penguinVersion);

}

} else {

MEDIA\_URLS\_ON\_TWEETS.increment();

}

}

}

}

/\*\*

\* Build the URL based fields from a tweet.

\*/

public static void buildURLFields(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message,

EarlybirdEncodedFeatures encodedFeatures

) {

Map<String, ThriftExpandedUrl> expandedUrlMap = message.getExpandedUrlMap();

for (ThriftExpandedUrl expandedUrl : expandedUrlMap.values()) {

if (expandedUrl.getMediaType() == MediaTypes.NATIVE\_IMAGE) {

EncodedFeatureBuilder.addPhotoUrl(message, expandedUrl.getCanonicalLastHopUrl());

}

}

// now add all twitter photos links that came with the tweet's payload

Map<Long, String> photos = message.getPhotoUrls();

List<TwitterPhotoUrl> photoURLs = new ArrayList<>();

if (photos != null) {

for (Map.Entry<Long, String> entry : photos.entrySet()) {

TwitterPhotoUrl photo = new TwitterPhotoUrl(entry.getKey());

String mediaUrl = entry.getValue();

if (mediaUrl != null) {

photo.setMediaUrl(mediaUrl);

}

photoURLs.add(photo);

}

}

try {

builder

.withURLs(Lists.newArrayList(expandedUrlMap.values()))

.withTwimgURLs(photoURLs);

} catch (IOException ioe) {

LOG.error("URL field creation threw an IOException", ioe);

}

if (encodedFeatures.isFlagSet(

EarlybirdFieldConstants.EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG)) {

builder.withOffensiveFlag();

}

if (encodedFeatures.isFlagSet(

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_CONSUMER\_VIDEO\_FLAG)) {

builder.addFilterInternalFieldTerm(

EarlybirdFieldConstants.EarlybirdFieldConstant.CONSUMER\_VIDEO\_FILTER\_TERM);

}

if (encodedFeatures.isFlagSet(

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_PRO\_VIDEO\_FLAG)) {

builder.addFilterInternalFieldTerm(

EarlybirdFieldConstants.EarlybirdFieldConstant.PRO\_VIDEO\_FILTER\_TERM);

}

if (encodedFeatures.isFlagSet(EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_VINE\_FLAG)) {

builder.addFilterInternalFieldTerm(

EarlybirdFieldConstants.EarlybirdFieldConstant.VINE\_FILTER\_TERM);

}

if (encodedFeatures.isFlagSet(

EarlybirdFieldConstants.EarlybirdFieldConstant.HAS\_PERISCOPE\_FLAG)) {

builder.addFilterInternalFieldTerm(

EarlybirdFieldConstants.EarlybirdFieldConstant.PERISCOPE\_FILTER\_TERM);

}

}

/\*\*

\* Build the card information inside ThriftIndexingEvent's fields.

\*/

static void buildCardFields(EarlybirdThriftDocumentBuilder builder,

TwitterMessage message,

PenguinVersion penguinVersion) {

if (message.hasCard()) {

SearchCard2 card = buildSearchCardFromTwitterMessage(

message,

TweetTokenStreamSerializer.getTweetTokenStreamSerializer(),

penguinVersion);

buildCardFeatures(message.getId(), builder, card);

}

}

private static SearchCard2 buildSearchCardFromTwitterMessage(

TwitterMessage message,

TokenStreamSerializer streamSerializer,

PenguinVersion penguinVersion) {

SearchCard2 card = new SearchCard2();

card.setCardName(message.getCardName());

if (message.getCardDomain() != null) {

card.setCardDomain(message.getCardDomain());

}

if (message.getCardLang() != null) {

card.setCardLang(message.getCardLang());

}

if (message.getCardUrl() != null) {

card.setCardUrl(message.getCardUrl());

}

if (message.getCardTitle() != null && !message.getCardTitle().isEmpty()) {

String normalizedTitle = NormalizerHelper.normalize(

message.getCardTitle(), message.getLocale(), penguinVersion);

TokenizerResult result = TokenizerHelper.tokenizeTweet(

normalizedTitle, message.getLocale(), penguinVersion);

TokenizedCharSequenceStream tokenSeqStream = new TokenizedCharSequenceStream();

tokenSeqStream.reset(result.tokenSequence);

try {

card.setCardTitleTokenStream(streamSerializer.serialize(tokenSeqStream));

card.setCardTitleTokenStreamText(result.tokenSequence.toString());

} catch (IOException e) {

LOG.error("TwitterTokenStream serialization error! Could not serialize card title: "

+ result.tokenSequence);

card.unsetCardTitleTokenStream();

card.unsetCardTitleTokenStreamText();

}

}

if (message.getCardDescription() != null && !message.getCardDescription().isEmpty()) {

String normalizedDesc = NormalizerHelper.normalize(

message.getCardDescription(), message.getLocale(), penguinVersion);

TokenizerResult result = TokenizerHelper.tokenizeTweet(

normalizedDesc, message.getLocale(), penguinVersion);

TokenizedCharSequenceStream tokenSeqStream = new TokenizedCharSequenceStream();

tokenSeqStream.reset(result.tokenSequence);

try {

card.setCardDescriptionTokenStream(streamSerializer.serialize(tokenSeqStream));

card.setCardDescriptionTokenStreamText(result.tokenSequence.toString());

} catch (IOException e) {

LOG.error("TwitterTokenStream serialization error! Could not serialize card description: "

+ result.tokenSequence);

card.unsetCardDescriptionTokenStream();

card.unsetCardDescriptionTokenStreamText();

}

}

return card;

}

/\*\*

\* Builds card features.

\*/

private static void buildCardFeatures(

long tweetId, EarlybirdThriftDocumentBuilder builder, SearchCard2 card) {

if (card == null) {

return;

}

builder

.withTokenStreamField(

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

card.getCardTitleTokenStreamText(),

card.isSetCardTitleTokenStream() ? card.getCardTitleTokenStream() : null)

.withTokenStreamField(

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

card.getCardDescriptionTokenStreamText(),

card.isSetCardDescriptionTokenStream() ? card.getCardDescriptionTokenStream() : null)

.withStringField(

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

card.getCardName())

.withIntField(

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

SearchCardType.cardTypeFromStringName(card.getCardName()).getByteValue());

if (card.getCardLang() != null) {

builder.withStringField(

EarlybirdFieldConstants.EarlybirdFieldConstant.CARD\_LANG.getFieldName(),

card.getCardLang()).withIntField(

EarlybirdFieldConstants.EarlybirdFieldConstant.CARD\_LANG\_CSF.getFieldName(),

ThriftLanguageUtil.getThriftLanguageOf(card.getCardLang()).getValue());

}

if (card.getCardDomain() != null) {

builder.withStringField(

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

card.getCardDomain());

}

if (card.getCardUrl() != null) {

NUM\_TWEETS\_WITH\_CARD\_URL.increment();

if (card.getCardUrl().startsWith("card://")) {

String suffix = card.getCardUrl().replace("card://", "");

if (StringUtils.isNumeric(suffix)) {

NUM\_TWEETS\_WITH\_NUMERIC\_CARD\_URI.increment();

builder.withLongField(

EarlybirdFieldConstants.EarlybirdFieldConstant.CARD\_URI\_CSF.getFieldName(),

Long.parseLong(suffix));

LOG.debug(String.format(

"Good card URL for tweet %s: %s",

tweetId,

card.getCardUrl()));

} else {

NUM\_TWEETS\_WITH\_INVALID\_CARD\_URI.increment();

LOG.debug(String.format(

"Card URL starts with \"card://\" but followed by non-numeric for tweet %s: %s",

tweetId,

card.getCardUrl()));

}

}

}

if (isCardVideo(card)) {

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

builder.addFacetSkipList(

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

}

}

/\*\*

\* Determines if a card is a video.

\*/

private static boolean isCardVideo(@Nullable SearchCard2 card) {

if (card == null) {

return false;

}

return AMPLIFY\_CARD\_NAME.equalsIgnoreCase(card.getCardName())

|| PLAYER\_CARD\_NAME.equalsIgnoreCase(card.getCardName());

}

private void buildSpaceAdminAndTitleFields(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message,

PenguinVersion penguinVersion) {

buildSpaceAdminFields(builder, message.getSpaceAdmins(), penguinVersion);

// build the space title field.

buildTweetTokenizerTokenizedField(

builder,

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

message.getSpaceTitle(),

penguinVersion);

}

private void buildSpaceAdminFields(

EarlybirdThriftDocumentBuilder builder,

Set<TwitterMessageUser> spaceAdmins,

PenguinVersion penguinVersion) {

for (TwitterMessageUser spaceAdmin : spaceAdmins) {

if (spaceAdmin.getScreenName().isPresent()) {

// build screen name (aka handle) fields.

String screenName = spaceAdmin.getScreenName().get();

String normalizedScreenName =

NormalizerHelper.normalizeWithUnknownLocale(screenName, penguinVersion);

builder.withStringField(

EarlybirdFieldConstants.EarlybirdFieldConstant.SPACE\_ADMIN\_FIELD.getFieldName(),

normalizedScreenName);

builder.withWhiteSpaceTokenizedScreenNameField(

EarlybirdFieldConstants

.EarlybirdFieldConstant.TOKENIZED\_SPACE\_ADMIN\_FIELD.getFieldName(),

normalizedScreenName);

if (spaceAdmin.getTokenizedScreenName().isPresent()) {

builder.withCamelCaseTokenizedScreenNameField(

EarlybirdFieldConstants

.EarlybirdFieldConstant.CAMELCASE\_TOKENIZED\_SPACE\_ADMIN\_FIELD.getFieldName(),

screenName,

normalizedScreenName,

spaceAdmin.getTokenizedScreenName().get());

}

}

if (spaceAdmin.getDisplayName().isPresent()) {

buildTweetTokenizerTokenizedField(

builder,

EarlybirdFieldConstants

.EarlybirdFieldConstant.TOKENIZED\_SPACE\_ADMIN\_DISPLAY\_NAME\_FIELD.getFieldName(),

spaceAdmin.getDisplayName().get(),

penguinVersion);

}

}

}

private void buildTweetTokenizerTokenizedField(

EarlybirdThriftDocumentBuilder builder,

String fieldName,

String text,

PenguinVersion penguinVersion) {

if (StringUtils.isNotEmpty(text)) {

Locale locale = LanguageIdentifierHelper

.identifyLanguage(text);

String normalizedText = NormalizerHelper.normalize(

text, locale, penguinVersion);

TokenizerResult result = TokenizerHelper

.tokenizeTweet(normalizedText, locale, penguinVersion);

TokenizedCharSequenceStream tokenSeqStream = new TokenizedCharSequenceStream();

tokenSeqStream.reset(result.tokenSequence);

TokenStreamSerializer streamSerializer =

TweetTokenStreamSerializer.getTweetTokenStreamSerializer();

try {

builder.withTokenStreamField(

fieldName,

result.tokenSequence.toString(),

streamSerializer.serialize(tokenSeqStream));

} catch (IOException e) {

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

}

}

}

}