package com.twitter.tweetypie.util

import com.twitter.dataproducts.enrichments.thriftscala.ProfileGeoEnrichment

import com.twitter.expandodo.thriftscala.\_

import com.twitter.mediaservices.commons.thriftscala.MediaKey

import com.twitter.mediaservices.commons.tweetmedia.thriftscala.\_

import com.twitter.servo.data.Lens

import com.twitter.spam.rtf.thriftscala.SafetyLabel

import com.twitter.tseng.withholding.thriftscala.TakedownReason

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.tweetypie.unmentions.thriftscala.UnmentionData

object TweetLenses {

import Lens.checkEq

def requireSome[A, B](l: Lens[A, Option[B]]): Lens[A, B] =

checkEq[A, B](

a => l.get(a).get,

(a, b) => l.set(a, Some(b))

)

def tweetLens[A](get: Tweet => A, set: (Tweet, A) => Tweet): Lens[Tweet, A] =

checkEq[Tweet, A](get, set)

val id: Lens[Tweet, TweetId] =

tweetLens[TweetId](\_.id, (t, id) => t.copy(id = id))

val coreData: Lens[Tweet, Option[TweetCoreData]] =

tweetLens[Option[TweetCoreData]](\_.coreData, (t, coreData) => t.copy(coreData = coreData))

val requiredCoreData: Lens[Tweet, TweetCoreData] =

requireSome(coreData)

val optUrls: Lens[Tweet, Option[Seq[UrlEntity]]] =

tweetLens[Option[Seq[UrlEntity]]](\_.urls, (t, urls) => t.copy(urls = urls))

val urls: Lens[Tweet, Seq[UrlEntity]] =

tweetLens[Seq[UrlEntity]](\_.urls.toSeq.flatten, (t, urls) => t.copy(urls = Some(urls)))

val optMentions: Lens[Tweet, Option[Seq[MentionEntity]]] =

tweetLens[Option[Seq[MentionEntity]]](\_.mentions, (t, v) => t.copy(mentions = v))

val mentions: Lens[Tweet, Seq[MentionEntity]] =

tweetLens[Seq[MentionEntity]](\_.mentions.toSeq.flatten, (t, v) => t.copy(mentions = Some(v)))

val unmentionData: Lens[Tweet, Option[UnmentionData]] =

tweetLens[Option[UnmentionData]](\_.unmentionData, (t, v) => t.copy(unmentionData = v))

val optHashtags: Lens[Tweet, Option[Seq[HashtagEntity]]] =

tweetLens[Option[Seq[HashtagEntity]]](\_.hashtags, (t, v) => t.copy(hashtags = v))

val hashtags: Lens[Tweet, Seq[HashtagEntity]] =

tweetLens[Seq[HashtagEntity]](\_.hashtags.toSeq.flatten, (t, v) => t.copy(hashtags = Some(v)))

val optCashtags: Lens[Tweet, Option[Seq[CashtagEntity]]] =

tweetLens[Option[Seq[CashtagEntity]]](\_.cashtags, (t, v) => t.copy(cashtags = v))

val cashtags: Lens[Tweet, Seq[CashtagEntity]] =

tweetLens[Seq[CashtagEntity]](\_.cashtags.toSeq.flatten, (t, v) => t.copy(cashtags = Some(v)))

val optMedia: Lens[Tweet, Option[Seq[MediaEntity]]] =

tweetLens[Option[Seq[MediaEntity]]](\_.media, (t, v) => t.copy(media = v))

val media: Lens[Tweet, Seq[MediaEntity]] =

tweetLens[Seq[MediaEntity]](\_.media.toSeq.flatten, (t, v) => t.copy(media = Some(v)))

val mediaKeys: Lens[Tweet, Seq[MediaKey]] =

tweetLens[Seq[MediaKey]](

\_.mediaKeys.toSeq.flatten,

{

case (t, v) => t.copy(mediaKeys = Some(v))

})

val place: Lens[Tweet, Option[Place]] =

tweetLens[Option[Place]](

\_.place,

{

case (t, v) => t.copy(place = v)

})

val quotedTweet: Lens[Tweet, Option[QuotedTweet]] =

tweetLens[Option[QuotedTweet]](

\_.quotedTweet,

{

case (t, v) => t.copy(quotedTweet = v)

})

val selfThreadMetadata: Lens[Tweet, Option[SelfThreadMetadata]] =

tweetLens[Option[SelfThreadMetadata]](

\_.selfThreadMetadata,

{

case (t, v) => t.copy(selfThreadMetadata = v)

})

val composerSource: Lens[Tweet, Option[ComposerSource]] =

tweetLens[Option[ComposerSource]](

\_.composerSource,

{

case (t, v) => t.copy(composerSource = v)

})

val deviceSource: Lens[Tweet, Option[DeviceSource]] =

tweetLens[Option[DeviceSource]](

\_.deviceSource,

{

case (t, v) => t.copy(deviceSource = v)

})

val perspective: Lens[Tweet, Option[StatusPerspective]] =

tweetLens[Option[StatusPerspective]](

\_.perspective,

{

case (t, v) => t.copy(perspective = v)

})

val cards: Lens[Tweet, Option[Seq[Card]]] =

tweetLens[Option[Seq[Card]]](

\_.cards,

{

case (t, v) => t.copy(cards = v)

})

val card2: Lens[Tweet, Option[Card2]] =

tweetLens[Option[Card2]](

\_.card2,

{

case (t, v) => t.copy(card2 = v)

})

val cardReference: Lens[Tweet, Option[CardReference]] =

tweetLens[Option[CardReference]](

\_.cardReference,

{

case (t, v) => t.copy(cardReference = v)

})

val spamLabel: Lens[Tweet, Option[SafetyLabel]] =

tweetLens[Option[SafetyLabel]](

\_.spamLabel,

{

case (t, v) => t.copy(spamLabel = v)

})

val lowQualityLabel: Lens[Tweet, Option[SafetyLabel]] =

tweetLens[Option[SafetyLabel]](

\_.lowQualityLabel,

{

case (t, v) => t.copy(lowQualityLabel = v)

})

val nsfwHighPrecisionLabel: Lens[Tweet, Option[SafetyLabel]] =

tweetLens[Option[SafetyLabel]](

\_.nsfwHighPrecisionLabel,

{

case (t, v) => t.copy(nsfwHighPrecisionLabel = v)

})

val bounceLabel: Lens[Tweet, Option[SafetyLabel]] =

tweetLens[Option[SafetyLabel]](

\_.bounceLabel,

{

case (t, v) => t.copy(bounceLabel = v)

})

val takedownCountryCodes: Lens[Tweet, Option[Seq[String]]] =

tweetLens[Option[Seq[String]]](

\_.takedownCountryCodes,

{

case (t, v) => t.copy(takedownCountryCodes = v)

})

val takedownReasons: Lens[Tweet, Option[Seq[TakedownReason]]] =

tweetLens[Option[Seq[TakedownReason]]](

\_.takedownReasons,

{

case (t, v) => t.copy(takedownReasons = v)

})

val contributor: Lens[Tweet, Option[Contributor]] =

tweetLens[Option[Contributor]](

\_.contributor,

{

case (t, v) => t.copy(contributor = v)

})

val mediaTags: Lens[Tweet, Option[TweetMediaTags]] =

tweetLens[Option[TweetMediaTags]](

\_.mediaTags,

{

case (t, v) => t.copy(mediaTags = v)

})

val mediaTagMap: Lens[Tweet, Map[MediaId, Seq[MediaTag]]] =

tweetLens[Map[MediaId, Seq[MediaTag]]](

\_.mediaTags.map { case TweetMediaTags(tagMap) => tagMap.toMap }.getOrElse(Map.empty),

(t, v) => {

val cleanMap = v.filter { case (\_, tags) => tags.nonEmpty }

t.copy(mediaTags = if (cleanMap.nonEmpty) Some(TweetMediaTags(cleanMap)) else None)

}

)

val escherbirdEntityAnnotations: Lens[Tweet, Option[EscherbirdEntityAnnotations]] =

tweetLens[Option[EscherbirdEntityAnnotations]](

\_.escherbirdEntityAnnotations,

{

case (t, v) => t.copy(escherbirdEntityAnnotations = v)

})

val communities: Lens[Tweet, Option[Communities]] =

tweetLens[Option[Communities]](

\_.communities,

{

case (t, v) => t.copy(communities = v)

})

val tweetypieOnlyTakedownCountryCodes: Lens[Tweet, Option[Seq[String]]] =

tweetLens[Option[Seq[String]]](

\_.tweetypieOnlyTakedownCountryCodes,

{

case (t, v) => t.copy(tweetypieOnlyTakedownCountryCodes = v)

})

val tweetypieOnlyTakedownReasons: Lens[Tweet, Option[Seq[TakedownReason]]] =

tweetLens[Option[Seq[TakedownReason]]](

\_.tweetypieOnlyTakedownReasons,

{

case (t, v) => t.copy(tweetypieOnlyTakedownReasons = v)

})

val profileGeo: Lens[Tweet, Option[ProfileGeoEnrichment]] =

tweetLens[Option[ProfileGeoEnrichment]](

\_.profileGeoEnrichment,

(t, v) => t.copy(profileGeoEnrichment = v)

)

val visibleTextRange: Lens[Tweet, Option[TextRange]] =

tweetLens[Option[TextRange]](

\_.visibleTextRange,

{

case (t, v) => t.copy(visibleTextRange = v)

})

val selfPermalink: Lens[Tweet, Option[ShortenedUrl]] =

tweetLens[Option[ShortenedUrl]](

\_.selfPermalink,

{

case (t, v) => t.copy(selfPermalink = v)

})

val extendedTweetMetadata: Lens[Tweet, Option[ExtendedTweetMetadata]] =

tweetLens[Option[ExtendedTweetMetadata]](

\_.extendedTweetMetadata,

{

case (t, v) => t.copy(extendedTweetMetadata = v)

})

object TweetCoreData {

val userId: Lens[TweetCoreData, UserId] = checkEq[TweetCoreData, UserId](

\_.userId,

{ (c, v) =>

// Pleases the compiler: https://github.com/scala/bug/issues/9171

val userId = v

c.copy(userId = userId)

})

val text: Lens[TweetCoreData, String] = checkEq[TweetCoreData, String](

\_.text,

{ (c, v) =>

// Pleases the compiler: https://github.com/scala/bug/issues/9171

val text = v

c.copy(text = text)

})

val createdAt: Lens[TweetCoreData, TweetId] =

checkEq[TweetCoreData, Long](\_.createdAtSecs, (c, v) => c.copy(createdAtSecs = v))

val createdVia: Lens[TweetCoreData, String] =

checkEq[TweetCoreData, String](

\_.createdVia,

{

case (c, v) => c.copy(createdVia = v)

})

val hasTakedown: Lens[TweetCoreData, Boolean] =

checkEq[TweetCoreData, Boolean](

\_.hasTakedown,

{

case (c, v) => c.copy(hasTakedown = v)

})

val nullcast: Lens[TweetCoreData, Boolean] =

checkEq[TweetCoreData, Boolean](

\_.nullcast,

{

case (c, v) => c.copy(nullcast = v)

})

val nsfwUser: Lens[TweetCoreData, Boolean] =

checkEq[TweetCoreData, Boolean](

\_.nsfwUser,

{

case (c, v) => c.copy(nsfwUser = v)

})

val nsfwAdmin: Lens[TweetCoreData, Boolean] =

checkEq[TweetCoreData, Boolean](

\_.nsfwAdmin,

{

case (c, v) => c.copy(nsfwAdmin = v)

})

val reply: Lens[TweetCoreData, Option[Reply]] =

checkEq[TweetCoreData, Option[Reply]](

\_.reply,

{

case (c, v) => c.copy(reply = v)

})

val share: Lens[TweetCoreData, Option[Share]] =

checkEq[TweetCoreData, Option[Share]](

\_.share,

{

case (c, v) => c.copy(share = v)

})

val narrowcast: Lens[TweetCoreData, Option[Narrowcast]] =

checkEq[TweetCoreData, Option[Narrowcast]](

\_.narrowcast,

{

case (c, v) => c.copy(narrowcast = v)

})

val directedAtUser: Lens[TweetCoreData, Option[DirectedAtUser]] =

checkEq[TweetCoreData, Option[DirectedAtUser]](

\_.directedAtUser,

{

case (c, v) => c.copy(directedAtUser = v)

})

val conversationId: Lens[TweetCoreData, Option[ConversationId]] =

checkEq[TweetCoreData, Option[ConversationId]](

\_.conversationId,

{

case (c, v) => c.copy(conversationId = v)

})

val placeId: Lens[TweetCoreData, Option[String]] =

checkEq[TweetCoreData, Option[String]](

\_.placeId,

{

case (c, v) => c.copy(placeId = v)

})

val geoCoordinates: Lens[TweetCoreData, Option[GeoCoordinates]] =

checkEq[TweetCoreData, Option[GeoCoordinates]](

\_.coordinates,

(c, v) => c.copy(coordinates = v)

)

val trackingId: Lens[TweetCoreData, Option[TweetId]] =

checkEq[TweetCoreData, Option[Long]](

\_.trackingId,

{

case (c, v) => c.copy(trackingId = v)

})

val hasMedia: Lens[TweetCoreData, Option[Boolean]] =

checkEq[TweetCoreData, Option[Boolean]](

\_.hasMedia,

{

case (c, v) => c.copy(hasMedia = v)

})

}

val counts: Lens[Tweet, Option[StatusCounts]] =

tweetLens[Option[StatusCounts]](

\_.counts,

{

case (t, v) => t.copy(counts = v)

})

object StatusCounts {

val retweetCount: Lens[StatusCounts, Option[TweetId]] =

checkEq[StatusCounts, Option[Long]](

\_.retweetCount,

(c, retweetCount) => c.copy(retweetCount = retweetCount)

)

val replyCount: Lens[StatusCounts, Option[TweetId]] =

checkEq[StatusCounts, Option[Long]](

\_.replyCount,

(c, replyCount) => c.copy(replyCount = replyCount)

)

val favoriteCount: Lens[StatusCounts, Option[TweetId]] =

checkEq[StatusCounts, Option[Long]](

\_.favoriteCount,

{

case (c, v) => c.copy(favoriteCount = v)

})

val quoteCount: Lens[StatusCounts, Option[TweetId]] =

checkEq[StatusCounts, Option[Long]](

\_.quoteCount,

{

case (c, v) => c.copy(quoteCount = v)

})

}

val userId: Lens[Tweet, UserId] = requiredCoreData andThen TweetCoreData.userId

val text: Lens[Tweet, String] = requiredCoreData andThen TweetCoreData.text

val createdVia: Lens[Tweet, String] = requiredCoreData andThen TweetCoreData.createdVia

val createdAt: Lens[Tweet, ConversationId] = requiredCoreData andThen TweetCoreData.createdAt

val reply: Lens[Tweet, Option[Reply]] = requiredCoreData andThen TweetCoreData.reply

val share: Lens[Tweet, Option[Share]] = requiredCoreData andThen TweetCoreData.share

val narrowcast: Lens[Tweet, Option[Narrowcast]] =

requiredCoreData andThen TweetCoreData.narrowcast

val directedAtUser: Lens[Tweet, Option[DirectedAtUser]] =

requiredCoreData andThen TweetCoreData.directedAtUser

val conversationId: Lens[Tweet, Option[ConversationId]] =

requiredCoreData andThen TweetCoreData.conversationId

val placeId: Lens[Tweet, Option[String]] = requiredCoreData andThen TweetCoreData.placeId

val geoCoordinates: Lens[Tweet, Option[GeoCoordinates]] =

requiredCoreData andThen TweetCoreData.geoCoordinates

val hasTakedown: Lens[Tweet, Boolean] = requiredCoreData andThen TweetCoreData.hasTakedown

val nsfwAdmin: Lens[Tweet, Boolean] = requiredCoreData andThen TweetCoreData.nsfwAdmin

val nsfwUser: Lens[Tweet, Boolean] = requiredCoreData andThen TweetCoreData.nsfwUser

val nullcast: Lens[Tweet, Boolean] = requiredCoreData andThen TweetCoreData.nullcast

val trackingId: Lens[Tweet, Option[ConversationId]] =

requiredCoreData andThen TweetCoreData.trackingId

val hasMedia: Lens[Tweet, Option[Boolean]] = requiredCoreData andThen TweetCoreData.hasMedia

object CashtagEntity {

val indices: Lens[CashtagEntity, (Short, Short)] =

checkEq[CashtagEntity, (Short, Short)](

t => (t.fromIndex, t.toIndex),

(t, v) => t.copy(fromIndex = v.\_1, toIndex = v.\_2)

)

val text: Lens[CashtagEntity, String] =

checkEq[CashtagEntity, String](\_.text, (t, text) => t.copy(text = text))

}

object HashtagEntity {

val indices: Lens[HashtagEntity, (Short, Short)] =

checkEq[HashtagEntity, (Short, Short)](

t => (t.fromIndex, t.toIndex),

(t, v) => t.copy(fromIndex = v.\_1, toIndex = v.\_2)

)

val text: Lens[HashtagEntity, String] =

checkEq[HashtagEntity, String](\_.text, (t, text) => t.copy(text = text))

}

object MediaEntity {

val indices: Lens[MediaEntity, (Short, Short)] =

checkEq[MediaEntity, (Short, Short)](

t => (t.fromIndex, t.toIndex),

(t, v) => t.copy(fromIndex = v.\_1, toIndex = v.\_2)

)

val mediaSizes: Lens[MediaEntity, collection.Set[MediaSize]] =

checkEq[MediaEntity, scala.collection.Set[MediaSize]](

\_.sizes,

(m, sizes) => m.copy(sizes = sizes)

)

val url: Lens[MediaEntity, String] =

checkEq[MediaEntity, String](

\_.url,

{

case (t, v) => t.copy(url = v)

})

val mediaInfo: Lens[MediaEntity, Option[MediaInfo]] =

checkEq[MediaEntity, Option[MediaInfo]](

\_.mediaInfo,

{

case (t, v) => t.copy(mediaInfo = v)

})

}

object MentionEntity {

val indices: Lens[MentionEntity, (Short, Short)] =

checkEq[MentionEntity, (Short, Short)](

t => (t.fromIndex, t.toIndex),

(t, v) => t.copy(fromIndex = v.\_1, toIndex = v.\_2)

)

val screenName: Lens[MentionEntity, String] =

checkEq[MentionEntity, String](

\_.screenName,

(t, screenName) => t.copy(screenName = screenName)

)

}

object UrlEntity {

val indices: Lens[UrlEntity, (Short, Short)] =

checkEq[UrlEntity, (Short, Short)](

t => (t.fromIndex, t.toIndex),

(t, v) => t.copy(fromIndex = v.\_1, toIndex = v.\_2)

)

val url: Lens[UrlEntity, String] =

checkEq[UrlEntity, String](\_.url, (t, url) => t.copy(url = url))

}

object Contributor {

val screenName: Lens[Contributor, Option[String]] =

checkEq[Contributor, Option[String]](

\_.screenName,

(c, screenName) => c.copy(screenName = screenName)

)

}

object Reply {

val inReplyToScreenName: Lens[Reply, Option[String]] =

checkEq[Reply, Option[String]](

\_.inReplyToScreenName,

(c, inReplyToScreenName) => c.copy(inReplyToScreenName = inReplyToScreenName)

)

val inReplyToStatusId: Lens[Reply, Option[TweetId]] =

checkEq[Reply, Option[TweetId]](

\_.inReplyToStatusId,

(c, inReplyToStatusId) => c.copy(inReplyToStatusId = inReplyToStatusId)

)

}

}