package com.twitter.product\_mixer.component\_library.premarshaller.urt.builder

import com.twitter.product\_mixer.component\_library.premarshaller.urt.builder.UrtCursorUpdater.getCursorByType

import com.twitter.product\_mixer.core.model.marshalling.response.urt.TimelineEntry

import com.twitter.product\_mixer.core.model.marshalling.response.urt.operation.CursorOperation

import com.twitter.product\_mixer.core.model.marshalling.response.urt.operation.CursorType

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

object UrtCursorUpdater {

def getCursorByType(

entries: Seq[TimelineEntry],

cursorType: CursorType

): Option[CursorOperation] = {

entries.collectFirst {

case cursor: CursorOperation if cursor.cursorType == cursorType => cursor

}

}

}

// If a CursorCandidate is returned by a Candidate Source, use this trait to update that Cursor as

// necessary (as opposed to building a new cursor which is done with the UrtCursorBuilder)

trait UrtCursorUpdater[-Query <: PipelineQuery] extends UrtCursorBuilder[Query] { self =>

def getExistingCursor(entries: Seq[TimelineEntry]): Option[CursorOperation] = {

getCursorByType(entries, self.cursorType)

}

def update(query: Query, entries: Seq[TimelineEntry]): Seq[TimelineEntry] = {

if (includeOperation(query, entries)) {

getExistingCursor(entries)

.map { existingCursor =>

// Safe .get because includeOperation() is shared in this context

// build() method creates a new CursorOperation. We copy over the `idToReplace`

// from the existing cursor.

val newCursor =

build(query, entries).get

.copy(idToReplace = existingCursor.idToReplace)

entries.filterNot(\_ == existingCursor) :+ newCursor

}.getOrElse(entries)

} else entries

}

}