package com.twitter.unified\_user\_actions.adapter.client\_event

import com.twitter.clientapp.thriftscala.ItemType

import com.twitter.clientapp.thriftscala.LogEvent

import com.twitter.clientapp.thriftscala.{Item => LogEventItem}

import com.twitter.unified\_user\_actions.thriftscala.\_

object ClientEventEngagement {

object TweetFav extends BaseClientEvent(ActionType.ClientTweetFav)

/\*\*

\* This is fired when a user unlikes a liked(favorited) tweet

\*/

object TweetUnfav extends BaseClientEvent(ActionType.ClientTweetUnfav)

/\*\*

\* This is "Send Reply" event to indicate publishing of a reply Tweet as opposed to clicking

\* on the reply button to initiate a reply Tweet (captured in ClientTweetClickReply).

\* The difference between this and the ServerTweetReply are:

\* 1) ServerTweetReply already has the new Tweet Id, 2) A sent reply may be lost during transfer

\* over the wire and thus may not end up with a follow-up ServerTweetReply.

\*/

object TweetReply extends BaseClientEvent(ActionType.ClientTweetReply)

/\*\*

\* This is the "send quote" event to indicate publishing of a quote tweet as opposed to clicking

\* on the quote button to initiate a quote tweet (captured in ClientTweetClickQuote).

\* The difference between this and the ServerTweetQuote are:

\* 1) ServerTweetQuote already has the new Tweet Id, 2) A sent quote may be lost during transfer

\* over the wire and thus may not ended up with a follow-up ServerTweetQuote.

\*/

object TweetQuote extends BaseClientEvent(ActionType.ClientTweetQuote)

/\*\*

\* This is the "retweet" event to indicate publishing of a retweet.

\*/

object TweetRetweet extends BaseClientEvent(ActionType.ClientTweetRetweet)

/\*\*

\* "action = reply" indicates that a user expressed the intention to reply to a Tweet by clicking

\* the reply button. No new tweet is created in this event.

\*/

object TweetClickReply extends BaseClientEvent(ActionType.ClientTweetClickReply)

/\*\*

\* Please note that the "action == quote" is NOT the create quote Tweet event like what

\* we can get from TweetyPie.

\* It is just click on the "quote tweet" (after clicking on the retweet button there are 2 options,

\* one is "retweet" and the other is "quote tweet")

\*

\* Also checked the CE (BQ Table), the `item.tweet\_details.quoting\_tweet\_id` is always NULL but

\* `item.tweet\_details.retweeting\_tweet\_id`, `item.tweet\_details.in\_reply\_to\_tweet\_id`, `item.tweet\_details.quoted\_tweet\_id`

\* could be NON-NULL and UUA would just include these NON-NULL fields as is. This is also checked in the unit test.

\*/

object TweetClickQuote extends BaseClientEvent(ActionType.ClientTweetClickQuote)

/\*\*

\* Refer to go/cme-scribing and go/interaction-event-spec for details.

\* Fired on the first tick of a track regardless of where in the video it is playing.

\* For looping playback, this is only fired once and does not reset at loop boundaries.

\*/

object TweetVideoPlaybackStart

extends BaseVideoClientEvent(ActionType.ClientTweetVideoPlaybackStart)

/\*\*

\* Refer to go/cme-scribing and go/interaction-event-spec for details.

\* Fired when playback reaches 100% of total track duration.

\* Not valid for live videos.

\* For looping playback, this is only fired once and does not reset at loop boundaries.

\*/

object TweetVideoPlaybackComplete

extends BaseVideoClientEvent(ActionType.ClientTweetVideoPlaybackComplete)

/\*\*

\* Refer to go/cme-scribing and go/interaction-event-spec for details.

\* This is fired when playback reaches 25% of total track duration. Not valid for live videos.

\* For looping playback, this is only fired once and does not reset at loop boundaries.

\*/

object TweetVideoPlayback25 extends BaseVideoClientEvent(ActionType.ClientTweetVideoPlayback25)

object TweetVideoPlayback50 extends BaseVideoClientEvent(ActionType.ClientTweetVideoPlayback50)

object TweetVideoPlayback75 extends BaseVideoClientEvent(ActionType.ClientTweetVideoPlayback75)

object TweetVideoPlayback95 extends BaseVideoClientEvent(ActionType.ClientTweetVideoPlayback95)

/\*\*

\* Refer to go/cme-scribing and go/interaction-event-spec for details.

\* This if fired when the video has been played in non-preview

\* (i.e. not autoplaying in the timeline) mode, and was not started via auto-advance.

\* For looping playback, this is only fired once and does not reset at loop boundaries.

\*/

object TweetVideoPlayFromTap extends BaseVideoClientEvent(ActionType.ClientTweetVideoPlayFromTap)

/\*\*

\* Refer to go/cme-scribing and go/interaction-event-spec for details.

\* This is fired when 50% of the video has been on-screen and playing for 10 consecutive seconds

\* or 95% of the video duration, whichever comes first.

\* For looping playback, this is only fired once and does not reset at loop boundaries.

\*/

object TweetVideoQualityView extends BaseVideoClientEvent(ActionType.ClientTweetVideoQualityView)

object TweetVideoView extends BaseVideoClientEvent(ActionType.ClientTweetVideoView)

object TweetVideoMrcView extends BaseVideoClientEvent(ActionType.ClientTweetVideoMrcView)

object TweetVideoViewThreshold

extends BaseVideoClientEvent(ActionType.ClientTweetVideoViewThreshold)

object TweetVideoCtaUrlClick extends BaseVideoClientEvent(ActionType.ClientTweetVideoCtaUrlClick)

object TweetVideoCtaWatchClick

extends BaseVideoClientEvent(ActionType.ClientTweetVideoCtaWatchClick)

/\*\*

\* This is fired when a user clicks on "Undo retweet" after re-tweeting a tweet

\*

\*/

object TweetUnretweet extends BaseClientEvent(ActionType.ClientTweetUnretweet)

/\*\*

\* This is fired when a user clicks on a photo attached to a tweet and the photo expands to fit

\* the screen.

\*/

object TweetPhotoExpand extends BaseClientEvent(ActionType.ClientTweetPhotoExpand)

/\*\*

\* This is fired when a user clicks on a card, a card could be a photo or video for example

\*/

object CardClick extends BaseCardClientEvent(ActionType.ClientCardClick)

object CardOpenApp extends BaseCardClientEvent(ActionType.ClientCardOpenApp)

object CardAppInstallAttempt extends BaseCardClientEvent(ActionType.ClientCardAppInstallAttempt)

object PollCardVote extends BaseCardClientEvent(ActionType.ClientPollCardVote)

/\*\*

\* This is fired when a user clicks on a profile mention inside a tweet.

\*/

object TweetClickMentionScreenName

extends BaseClientEvent(ActionType.ClientTweetClickMentionScreenName) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] =

(

ceItem.id,

logEvent.eventDetails.flatMap(

\_.targets.flatMap(\_.find(\_.itemType.contains(ItemType.User))))) match {

case (Some(tweetId), Some(target)) =>

(target.id, target.name) match {

case (Some(profileId), Some(profileHandle)) =>

Some(

Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(tweetId, ceItem, logEvent.eventNamespace)

.copy(tweetActionInfo = Some(

TweetActionInfo.ClientTweetClickMentionScreenName(

ClientTweetClickMentionScreenName(

actionProfileId = profileId,

handle = profileHandle

))))))

case \_ => None

}

case \_ => None

}

}

/\*\*

\* These are fired when user follows/unfollows a Topic. Please see the comment in the

\* ClientEventAdapter namespace matching to see the subtle details.

\*/

object TopicFollow extends BaseTopicClientEvent(ActionType.ClientTopicFollow)

object TopicUnfollow extends BaseTopicClientEvent(ActionType.ClientTopicUnfollow)

/\*\*

\* This is fired when the user clicks the "x" icon next to the topic on their timeline,

\* and clicks "Not interested in {TOPIC}" in the pop-up prompt

\* Alternatively, they can also click "See more" button to visit the topic page, and click "Not interested" there.

\*/

object TopicNotInterestedIn extends BaseTopicClientEvent(ActionType.ClientTopicNotInterestedIn)

/\*\*

\* This is fired when the user clicks the "Undo" button after clicking "x" or "Not interested" on a Topic

\* which is captured in ClientTopicNotInterestedIn

\*/

object TopicUndoNotInterestedIn

extends BaseTopicClientEvent(ActionType.ClientTopicUndoNotInterestedIn)

/\*\*

\* This is fired when a user clicks on "This Tweet's not helpful" flow in the caret menu

\* of a Tweet result on the Search Results Page

\*/

object TweetNotHelpful extends BaseClientEvent(ActionType.ClientTweetNotHelpful)

/\*\*

\* This is fired when a user clicks Undo after clicking on

\* "This Tweet's not helpful" flow in the caret menu of a Tweet result on the Search Results Page

\*/

object TweetUndoNotHelpful extends BaseClientEvent(ActionType.ClientTweetUndoNotHelpful)

object TweetReport extends BaseClientEvent(ActionType.ClientTweetReport) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] = {

for {

actionTweetId <- ceItem.id

} yield {

Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(

actionTweetId = actionTweetId,

ceItem = ceItem,

ceNamespaceOpt = logEvent.eventNamespace)

.copy(tweetActionInfo = Some(

TweetActionInfo.ClientTweetReport(

ClientTweetReport(

isReportTweetDone =

logEvent.eventNamespace.flatMap(\_.action).exists(\_.contains("done")),

reportFlowId = logEvent.reportDetails.flatMap(\_.reportFlowId)

)

))))

}

}

}

/\*\*

\* Not Interested In (Do Not like) event

\*/

object TweetNotInterestedIn extends BaseClientEvent(ActionType.ClientTweetNotInterestedIn)

object TweetUndoNotInterestedIn extends BaseClientEvent(ActionType.ClientTweetUndoNotInterestedIn)

/\*\*

\* This is fired when a user FIRST clicks the "Not interested in this Tweet" button in the caret menu of a Tweet

\* then clicks "This Tweet is not about {TOPIC}" in the subsequent prompt

\* Note: this button is hidden unless a user clicks "Not interested in this Tweet" first.

\*/

object TweetNotAboutTopic extends BaseClientEvent(ActionType.ClientTweetNotAboutTopic)

/\*\*

\* This is fired when a user clicks "Undo" immediately after clicking "This Tweet is not about {TOPIC}",

\* which is captured in TweetNotAboutTopic

\*/

object TweetUndoNotAboutTopic extends BaseClientEvent(ActionType.ClientTweetUndoNotAboutTopic)

/\*\*

\* This is fired when a user FIRST clicks the "Not interested in this Tweet" button in the caret menu of a Tweet

\* then clicks "This Tweet isn't recent" in the subsequent prompt

\* Note: this button is hidden unless a user clicks "Not interested in this Tweet" first.

\*/

object TweetNotRecent extends BaseClientEvent(ActionType.ClientTweetNotRecent)

/\*\*

\* This is fired when a user clicks "Undo" immediately after clicking "his Tweet isn't recent",

\* which is captured in TweetNotRecent

\*/

object TweetUndoNotRecent extends BaseClientEvent(ActionType.ClientTweetUndoNotRecent)

/\*\*

\* This is fired when a user clicks "Not interested in this Tweet" button in the caret menu of a Tweet

\* then clicks "Show fewer tweets from" in the subsequent prompt

\* Note: this button is hidden unless a user clicks "Not interested in this Tweet" first.

\*/

object TweetSeeFewer extends BaseClientEvent(ActionType.ClientTweetSeeFewer)

/\*\*

\* This is fired when a user clicks "Undo" immediately after clicking "Show fewer tweets from",

\* which is captured in TweetSeeFewer

\*/

object TweetUndoSeeFewer extends BaseClientEvent(ActionType.ClientTweetUndoSeeFewer)

/\*\*

\* This is fired when a user click "Submit" at the end of a "Report Tweet" flow

\* ClientTweetReport = 1041 is scribed by HealthClient team, on the client side

\* This is scribed by spamacaw, on the server side

\* They can be joined on reportFlowId

\* See https://confluence.twitter.biz/pages/viewpage.action?spaceKey=HEALTH&title=Understanding+ReportDetails

\*/

object TweetReportServer extends BaseClientEvent(ActionType.ServerTweetReport) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] =

for {

actionTweetId <- ceItem.id

} yield Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(

actionTweetId = actionTweetId,

ceItem = ceItem,

ceNamespaceOpt = logEvent.eventNamespace)

.copy(tweetActionInfo = Some(

TweetActionInfo.ServerTweetReport(

ServerTweetReport(

reportFlowId = logEvent.reportDetails.flatMap(\_.reportFlowId),

reportType = logEvent.reportDetails.flatMap(\_.reportType)

)

))))

}

/\*\*

\* This is fired when a user clicks Block in a Profile page

\* A Profile can also be blocked when a user clicks Block in the menu of a Tweet, which

\* is captured in ClientTweetBlockAuthor

\*/

object ProfileBlock extends BaseProfileClientEvent(ActionType.ClientProfileBlock)

/\*\*

\* This is fired when a user clicks unblock in a pop-up prompt right after blocking a profile

\* in the profile page or clicks unblock in a drop-down menu in the profile page.

\*/

object ProfileUnblock extends BaseProfileClientEvent(ActionType.ClientProfileUnblock)

/\*\*

\* This is fired when a user clicks Mute in a Profile page

\* A Profile can also be muted when a user clicks Mute in the menu of a Tweet, which

\* is captured in ClientTweetMuteAuthor

\*/

object ProfileMute extends BaseProfileClientEvent(ActionType.ClientProfileMute)

/\*

\* This is fired when a user clicks "Report User" action from user profile page

\* \*/

object ProfileReport extends BaseProfileClientEvent(ActionType.ClientProfileReport)

// This is fired when a user profile is open in a Profile page

object ProfileShow extends BaseProfileClientEvent(ActionType.ClientProfileShow)

object ProfileClick extends BaseProfileClientEvent(ActionType.ClientProfileClick) {

/\*\*

\* ClientTweetClickProfile would emit 2 events, 1 with item type Tweet and 1 with item type User

\* Both events will go to both actions (the actual classes). For ClientTweetClickProfile,

\* item type of Tweet will filter out the event with item type User. But for ClientProfileClick,

\* because we need to include item type of User, then we will also include the event of TweetClickProfile

\* if we don't do anything here. This override ensures we don't include tweet author clicks events in ProfileClick

\*/

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] =

if (logEvent.eventDetails

.flatMap(\_.items).exists(items => items.exists(\_.itemType.contains(ItemType.Tweet)))) {

None

} else {

super.getUuaItem(ceItem, logEvent)

}

}

/\*\*

\* This is fired when a user follows a profile from the

\* profile page / people module and people tab on the Search Results Page / sidebar on the Home page

\* A Profile can also be followed when a user clicks follow in the

\* caret menu of a Tweet / follow button on hovering on profile avatar,

\* which is captured in ClientTweetFollowAuthor

\*/

object ProfileFollow extends BaseProfileClientEvent(ActionType.ClientProfileFollow) {

/\*\*

\* ClientTweetFollowAuthor would emit 2 events, 1 with item type Tweet and 1 with item type User

\* Both events will go to both actions (the actual classes). For ClientTweetFollowAuthor,

\* item type of Tweet will filter out the event with item type User. But for ClientProfileFollow,

\* because we need to include item type of User, then we will also include the event of TweetFollowAuthor

\* if we don't do anything here. This override ensures we don't include tweet author follow events in ProfileFollow

\*/

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] =

if (logEvent.eventDetails

.flatMap(\_.items).exists(items => items.exists(\_.itemType.contains(ItemType.Tweet)))) {

None

} else {

super.getUuaItem(ceItem, logEvent)

}

}

/\*\*

\* This is fired when a user clicks Follow in the caret menu of a Tweet or hovers on the avatar of the tweet author

\* and clicks on the Follow button. A profile can also be followed by clicking the Follow button on the Profile

\* page and confirm, which is captured in ClientProfileFollow.

\* The event emits two items, one of user type and another of tweet type, since the default implementation of

\* BaseClientEvent only looks for Tweet type, the other item is dropped which is the expected behaviour

\*/

object TweetFollowAuthor extends BaseClientEvent(ActionType.ClientTweetFollowAuthor) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] = {

for {

actionTweetId <- ceItem.id

} yield {

Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(

actionTweetId = actionTweetId,

ceItem = ceItem,

ceNamespaceOpt = logEvent.eventNamespace)

.copy(tweetActionInfo = Some(

TweetActionInfo.ClientTweetFollowAuthor(

ClientTweetFollowAuthor(

ClientEventCommonUtils.getTweetAuthorFollowSource(logEvent.eventNamespace))

))))

}

}

}

/\*\*

\* This is fired when a user clicks Unfollow in the caret menu of a Tweet or hovers on the avatar of the tweet author

\* and clicks on the Unfollow button. A profile can also be unfollowed by clicking the Unfollow button on the Profile

\* page and confirm, which will be captured in ClientProfileUnfollow.

\* The event emits two items, one of user type and another of tweet type, since the default implementation of

\* BaseClientEvent only looks for Tweet type, the other item is dropped which is the expected behaviour

\*/

object TweetUnfollowAuthor extends BaseClientEvent(ActionType.ClientTweetUnfollowAuthor) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] = {

for {

actionTweetId <- ceItem.id

} yield {

Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(

actionTweetId = actionTweetId,

ceItem = ceItem,

ceNamespaceOpt = logEvent.eventNamespace)

.copy(tweetActionInfo = Some(

TweetActionInfo.ClientTweetUnfollowAuthor(

ClientTweetUnfollowAuthor(

ClientEventCommonUtils.getTweetAuthorUnfollowSource(logEvent.eventNamespace))

))))

}

}

}

/\*\*

\* This is fired when a user clicks Block in the caret menu of a Tweet to block the profile

\* that authors this Tweet. A profile can also be blocked in the Profile page, which is captured

\* in ClientProfileBlock

\*/

object TweetBlockAuthor extends BaseClientEvent(ActionType.ClientTweetBlockAuthor)

/\*\*

\* This is fired when a user clicks unblock in a pop-up prompt right after blocking an author

\* in the drop-down menu of a tweet

\*/

object TweetUnblockAuthor extends BaseClientEvent(ActionType.ClientTweetUnblockAuthor)

/\*\*

\* This is fired when a user clicks Mute in the caret menu of a Tweet to mute the profile

\* that authors this Tweet. A profile can also be muted in the Profile page, which is captured

\* in ClientProfileMute

\*/

object TweetMuteAuthor extends BaseClientEvent(ActionType.ClientTweetMuteAuthor)

/\*\*

\* This is fired when a user clicks on a Tweet to open the Tweet details page. Note that for

\* Tweets in the Notification Tab product surface, a click can be registered differently

\* depending on whether the Tweet is a rendered Tweet (a click results in ClientTweetClick)

\* or a wrapper Notification (a click results in ClientNotificationClick).

\*/

object TweetClick extends BaseClientEvent(ActionType.ClientTweetClick)

/\*\*

\* This is fired when a user clicks to view the profile page of another user from a Tweet

\*/

object TweetClickProfile extends BaseClientEvent(ActionType.ClientTweetClickProfile)

/\*\*

\* This is fired when a user clicks on the "share" icon on a Tweet to open the share menu.

\* The user may or may not proceed and finish sharing the Tweet.

\*/

object TweetClickShare extends BaseClientEvent(ActionType.ClientTweetClickShare)

/\*\*

\* This is fired when a user clicks "Copy link to Tweet" in a menu appeared after hitting

\* the "share" icon on a Tweet OR when a user selects share\_via -> copy\_link after long-click

\* a link inside a tweet on a mobile device

\*/

object TweetShareViaCopyLink extends BaseClientEvent(ActionType.ClientTweetShareViaCopyLink)

/\*\*

\* This is fired when a user clicks "Send via Direct Message" after

\* clicking on the "share" icon on a Tweet to open the share menu.

\* The user may or may not proceed and finish Sending the DM.

\*/

object TweetClickSendViaDirectMessage

extends BaseClientEvent(ActionType.ClientTweetClickSendViaDirectMessage)

/\*\*

\* This is fired when a user clicks "Bookmark" after

\* clicking on the "share" icon on a Tweet to open the share menu.

\*/

object TweetShareViaBookmark extends BaseClientEvent(ActionType.ClientTweetShareViaBookmark)

/\*\*

\* This is fired when a user clicks "Remove Tweet from Bookmarks" after

\* clicking on the "share" icon on a Tweet to open the share menu.

\*/

object TweetUnbookmark extends BaseClientEvent(ActionType.ClientTweetUnbookmark)

/\*\*

\* This event is fired when the user clicks on a hashtag in a Tweet.

\*/

object TweetClickHashtag extends BaseClientEvent(ActionType.ClientTweetClickHashtag) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] = for {

actionTweetId <- ceItem.id

} yield Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(

actionTweetId = actionTweetId,

ceItem = ceItem,

ceNamespaceOpt = logEvent.eventNamespace)

.copy(tweetActionInfo = logEvent.eventDetails

.map(

\_.targets.flatMap(\_.headOption.flatMap(\_.name))

) // fetch the first item in the details and then the name will have the hashtag value with the '#' sign

.map { hashtagOpt =>

TweetActionInfo.ClientTweetClickHashtag(

ClientTweetClickHashtag(hashtag = hashtagOpt)

)

}))

}

/\*\*

\* This is fired when a user clicks "Bookmark" after clicking on the "share" icon on a Tweet to

\* open the share menu, or when a user clicks on the 'bookmark' icon on a Tweet (bookmark icon

\* is available to ios only as of March 2023).

\* TweetBookmark and TweetShareByBookmark log the same events but serve for individual use cases.

\*/

object TweetBookmark extends BaseClientEvent(ActionType.ClientTweetBookmark)

/\*\*

\* This is fired when a user clicks on a link in a tweet.

\* The link could be displayed as a URL or embedded

\* in a component such as an image or a card in a tweet.

\*/

object TweetOpenLink extends BaseClientEvent(ActionType.ClientTweetOpenLink) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] =

for {

actionTweetId <- ceItem.id

} yield Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(

actionTweetId = actionTweetId,

ceItem = ceItem,

ceNamespaceOpt = logEvent.eventNamespace)

.copy(tweetActionInfo = Some(

TweetActionInfo.ClientTweetOpenLink(

ClientTweetOpenLink(url = logEvent.eventDetails.flatMap(\_.url))

))))

}

/\*\*

\* This is fired when a user takes a screenshot.

\* This is available for only mobile clients.

\*/

object TweetTakeScreenshot extends BaseClientEvent(ActionType.ClientTweetTakeScreenshot) {

override def getUuaItem(

ceItem: LogEventItem,

logEvent: LogEvent

): Option[Item] =

for {

actionTweetId <- ceItem.id

} yield Item.TweetInfo(

ClientEventCommonUtils

.getBasicTweetInfo(

actionTweetId = actionTweetId,

ceItem = ceItem,

ceNamespaceOpt = logEvent.eventNamespace)

.copy(tweetActionInfo = Some(

TweetActionInfo.ClientTweetTakeScreenshot(

ClientTweetTakeScreenshot(percentVisibleHeight100k = ceItem.percentVisibleHeight100k)

))))

}

/\*\*

\* This is fired when a user clicks the "This Tweet isn't relevant" button in a prompt displayed

\* after clicking "This Tweet's not helpful" in search result page or "Not Interested in this Tweet"

\* in the home timeline page.

\* Note: this button is hidden unless a user clicks "This Tweet isn't relevant" or

\* "This Tweet's not helpful" first

\*/

object TweetNotRelevant extends BaseClientEvent(ActionType.ClientTweetNotRelevant)

/\*\*

\* This is fired when a user clicks "Undo" immediately after clicking "this Tweet isn't relevant",

\* which is captured in TweetNotRelevant

\*/

object TweetUndoNotRelevant extends BaseClientEvent(ActionType.ClientTweetUndoNotRelevant)

/\*\*

\* This is fired when a user is logged out and follows a profile from the

\* profile page / people module from web.

\* One can only try to follow from web, iOS and Android do not support logged out browsing

\*/

object ProfileFollowAttempt extends BaseProfileClientEvent(ActionType.ClientProfileFollowAttempt)

/\*\*

\* This is fired when a user is logged out and favourite a tweet from web.

\* One can only try to favourite from web, iOS and Android do not support logged out browsing

\*/

object TweetFavoriteAttempt extends BaseClientEvent(ActionType.ClientTweetFavoriteAttempt)

/\*\*

\* This is fired when a user is logged out and Retweet a tweet from web.

\* One can only try to favourite from web, iOS and Android do not support logged out browsing

\*/

object TweetRetweetAttempt extends BaseClientEvent(ActionType.ClientTweetRetweetAttempt)

/\*\*

\* This is fired when a user is logged out and reply on tweet from web.

\* One can only try to favourite from web, iOS and Android do not support logged out browsing

\*/

object TweetReplyAttempt extends BaseClientEvent(ActionType.ClientTweetReplyAttempt)

/\*\*

\* This is fired when a user is logged out and clicks on login button.

\* Currently seem to be generated only on [m5, LiteNativeWrapper] as of Jan 2023.

\*/

object CTALoginClick extends BaseCTAClientEvent(ActionType.ClientCTALoginClick)

/\*\*

\* This is fired when a user is logged out and login window is shown.

\*/

object CTALoginStart extends BaseCTAClientEvent(ActionType.ClientCTALoginStart)

/\*\*

\* This is fired when a user is logged out and login is successful.

\*/

object CTALoginSuccess extends BaseCTAClientEvent(ActionType.ClientCTALoginSuccess)

/\*\*

\* This is fired when a user is logged out and clicks on signup button.

\*/

object CTASignupClick extends BaseCTAClientEvent(ActionType.ClientCTASignupClick)

/\*\*

\* This is fired when a user is logged out and signup is successful.

\*/

object CTASignupSuccess extends BaseCTAClientEvent(ActionType.ClientCTASignupSuccess)

/\*\*

\* This is fired when a user opens a Push Notification.

\* Refer to https://confluence.twitter.biz/pages/viewpage.action?pageId=161811800

\* for Push Notification scribe details

\*/

object NotificationOpen extends BasePushNotificationClientEvent(ActionType.ClientNotificationOpen)

/\*\*

\* This is fired when a user clicks on a notification in the Notification Tab.

\* Refer to go/ntab-urt-scribe for Notification Tab scribe details.

\*/

object NotificationClick

extends BaseNotificationTabClientEvent(ActionType.ClientNotificationClick)

/\*\*

\* This is fired when a user taps the "See Less Often" caret menu item of a notification in

\* the Notification Tab.

\* Refer to go/ntab-urt-scribe for Notification Tab scribe details.

\*/

object NotificationSeeLessOften

extends BaseNotificationTabClientEvent(ActionType.ClientNotificationSeeLessOften)

/\*\*

\* This is fired when a user closes or swipes away a Push Notification.

\* Refer to https://confluence.twitter.biz/pages/viewpage.action?pageId=161811800

\* for Push Notification scribe details

\*/

object NotificationDismiss

extends BasePushNotificationClientEvent(ActionType.ClientNotificationDismiss)

/\*\*

\* This is fired when a user clicks on a typeahead suggestion(queries, events, topics, users)

\* in a drop-down menu of a search box or a tweet compose box.

\*/

object TypeaheadClick extends BaseSearchTypeaheadEvent(ActionType.ClientTypeaheadClick)

/\*\*

\* This is a generic event fired when the user submits feedback on a prompt.

\* Some examples include Did You Find It Prompt and Tweet Relevance on Search Results Page.

\*/

object FeedbackPromptSubmit

extends BaseFeedbackSubmitClientEvent(ActionType.ClientFeedbackPromptSubmit)

object AppExit extends BaseUASClientEvent(ActionType.ClientAppExit)

}