package com.twitter.tweetypie.serverutil

/\*\*

\* Parse a device source into an OAuth app id. This mapping is

\* neccesary when you need to request information about a client from

\* a service that only knows about clients in terms of oauthIds.

\*

\* This happens either by parsing out an explicit "oauth:" app id or

\* using a mapping from old non oauth clientIds like "web" and "sms"

\* to oauthIds that have retroactively been assigned to those clients.

\* If the legacy id cannot be found in the map and it's a non-numeric

\* string, it's converted to the oauthId for twitter.com.

\*

\* Tweets with non oauth clientIds are still being created because

\* thats how the monorail creates them. We also need to be able to

\* process any app id string that is in old tweet data.

\*

\*/

object DeviceSourceParser {

/\*\*

\* The oauth id for twitter.com. Also used as a default oauth id for

\* other clients without their own

\*/

val Web = 268278L

/\*\*

\* The OAuth app ids for known legacy device sources.

\*/

val legacyMapping: Map[String, Long] = Map[String, Long](

"web" -> Web,

"tweetbutton" -> 6219130L,

"keitai\_web" -> 38366L,

"sms" -> 241256L

)

/\*\*

\* Attempt to convert a client application id String into an OAuth

\* id.

\*

\* The string must consist of the characters "oauth:" followed by a

\* non-negative, decimal long. The text is case-insensitive, and

\* whitespace at the beginning or end is ignored.

\*

\* We want to accept input as liberally as possible, because if we

\* fail to do that here, it will get counted as a "legacy app id"

\*/

val parseOAuthAppId: String => Option[Long] = {

// Case-insensitive, whitespace insensitive. The javaWhitespace

// character class is consistent with Character.isWhitespace, but is

// sadly different from \s. It will likely not matter in the long

// run, but this accepts more inputs and is easier to test (because

// we can use isWhitespace)

val OAuthAppIdRe = """(?i)\p{javaWhitespace}\*oauth:(\d+)\p{javaWhitespace}\*""".r

\_ match {

case OAuthAppIdRe(digits) =>

// We should only get NumberFormatException when the number is

// larger than a Long, because the regex will rule out all of

// the other invalid cases.

try Some(digits.toLong)

catch { case \_: NumberFormatException => None }

case \_ =>

None

}

}

/\*\*

\* Attempt to convert a client application id String into an OAuth id or legacy identifier without

\* any fallback behavior.

\*/

val parseStrict: String => Option[Long] =

appIdStr =>

parseOAuthAppId(appIdStr)

.orElse(legacyMapping.get(appIdStr))

/\*\*

\* Return true if a string can be used as a valid client application id or legacy identifier

\*/

val isValid: String => Boolean = appIdStr => parseStrict(appIdStr).isDefined

/\*\*

\* Build a parser that converts device sources to OAuth app ids,

\* including performing the legacy mapping.

\*/

val parseAppId: String => Option[Long] = {

val IsNumericRe = """-?[0-9]+""".r

appIdStr =>

parseStrict(appIdStr)

.orElse {

appIdStr match {

// We just fail the lookup if the app id looks like it's

// numeric.

case IsNumericRe() => None

case \_ => Some(Web)

}

}

}

}