\_ = require('underscore')

request = require('request')

HttpsProxyAgent = require('https-proxy-agent')

options =

countryBias: "us" #more likely to find addresses in this country. Think of this as you where you are searching "from" to find results around you. (use ISO 3166-1 country code)

countryMatch: null #match results in this country only. (ISO 3166-1 country code)

key: null #optional google api key (if used will submit requests over https)

proxy: null #optional proxy address

exports.setOptions = (opts) ->

\_.extend(options, opts)

matchUnknownType = (known, unknown) ->

compare = (prop) =>

if known[prop] and unknown[prop]

if known[prop].toLowerCase() == unknown[prop].toLowerCase()

return true

if unknown.generated and unknown[prop+'Abbr']

return known[prop].toLowerCase() == unknown[prop+'Abbr'].toLowerCase()

else if known.generated and known[prop+'Abbr']

return known[prop+'Abbr'].toLowerCase() == unknown[prop].toLowerCase()

else

return false

return !known[prop] and !unknown[prop]

if known.isObject and unknown.isObject

return compare('city') && compare('state') && compare('country')

else if known.isObject and not unknown.isObject

#unknown was provided as a string, and now we must check if the provided address is indeed one of the ones returned

props = ['streetNumber', 'street', 'city', 'state', 'country', 'postalCode']

otherAddress = unknown.toString().toLowerCase();

if known.toString() == otherAddress

return true

foundProps = 0

haveProps = 0

find = (val) ->

val = val.toLowerCase()

oldlen = otherAddress.length

otherAddress = otherAddress.replace(new RegExp("\\b"+val+"\\b", "i"), "")

if oldlen != otherAddress.length

foundProps++

return true

return false

for prop in props

value = known[prop]

if value != undefined

found = find(value)

if not found and prop in ["state", "country", "street"] and known[prop+"Abbr"] != undefined

found = find(known[prop+"Abbr"])

if not found and prop == "country" and value.toLowerCase() == "united states"

found = find("usa")

if not found and prop == "street"

value = value.replace(/( street)/i, ' st')

found = find(value)

if not found

value = value.replace(/( road)/i, ' rd')

find(value)

if not found and prop == "postalCode"

haveProps-- #these arent always specified. if the rest of the address matches we dont care about this

haveProps++

otherAddress = otherAddress.replace(/[ ,]/g, '')

#console.log("found:"+foundProps+" have:"+haveProps+" left: ["+otherAddress+"]")

return foundProps == haveProps and otherAddress.length == 0

else

return known.toString().toLowerCase() == unknown.toString().toLowerCase()

addressMatch =

streetAddress: [{location\_type: "ROOFTOP", types: ["street\_address"], exact: true}, {location\_type: "RANGE\_INTERPOLATED", types: ["street\_address"], exact: false}]

route: [{location\_type: "GEOMETRIC\_CENTER", types: ["route"], exact: true}]

city: [{location\_type: "APPROXIMATE", types: [ "locality", "political" ], exact: true}]

state: [{location\_type: "APPROXIMATE", types: [ "administrative\_area\_level\_1", "political" ], exact: true}]

country: [{location\_type: "APPROXIMATE", types: [ "country", "political" ], exact: true}]

unknown: [{location\_type: "unknown", types: ["unknown"], exact: true}] #wont match anything in the response.

exports.match = matchType = {}

\_.each(addressMatch, (list, name) ->

matchType[name] = name

)

###

Address object that provides useful methods. Create a new one by

1. passing a map with these props: {street:'123 main st', city: 'boston', state: 'MA'|'massachussetts', country: 'US'|'United States'}

None of the props are required, but chances are you wont have a valid address if you omit any of them (except for state)

2. passing a string containing an address (the address class does not parse the string into parts)

3. passing a result object from a google geocoding response. ie: geoResponse.results[0]

The validator.validate callback will return to you these objects, except they will have all or some of the following properties:

streetNumber: '100'

street: 'North Main St'

streetAbbr: 'N Main St'

city: 'Boston'

state: 'Massachussetts'

stateAbbr: 'MA'

country: 'United States'

countryAbbr: 'US'

postalCode: 02114

location: {lat: 43.233332, lon: 23.2222243}

Methods:

toString(useCountryAbbr, useStateAbbr, useStreetAbbr) - returns a string representing the address. currently geared towards North American addresses

useCountryAbbr = [optional] default: true - the resulting address string should use country abbr, not the full country name

useStateAbbr = [optional] default: true - the resulting address string should use state abbr, not the full state name

useStreetAbbr = [optional] default: false - the resulting address string should use street name abbr, not the full street name

Note: the abbriviated values will only be used if they are available. The Address objects returned to you from the validate callback may have these available.

equals(anotherAddress) - check if 2 addresses are probably\* the same. IT DOES NOT CHECK STREET NAME/NUMBER

###

exports.Address = class Address

matchType: matchType.unknown

exactMatch: null #can only be set on a @generated address

constructor: (address, @isObject=false, @generated=false) ->

if \_.isObject(address) #this gives you higher accuracy because we can compare resulting address parts to the input's address parts and see if its they are the same or not

@isObject = true

if address.address\_components #the address is parsed from a response from google geocoding

@generated = true

location =

lat: address.geometry?.location?.lat

lon: address.geometry?.location?.lng

#figure out the match type

@exactMatch = not address.partial\_match

\_.each(addressMatch, (list, name) =>

\_.each(list, (obj) =>

if(obj.location\_type == address.geometry.location\_type and \_.difference(obj.types, address.types).length == 0)

@matchType = name

if not obj.exact

@exactMatch = false

)

)

getComponent = @componentFinder(address.address\_components)

[x, streetNum] = getComponent('street\_number')

[streetAbbr, street] = getComponent('route')

[x, city] = getComponent('locality')

[stateAbbr, state] = getComponent('administrative\_area\_level\_1')

[countryAbbr, country] = getComponent('country')

[postalCode, x] = getComponent('postal\_code')

[postalCodePrefix, x] = getComponent('postal\_code\_prefix')

[colloquialArea, x] = getComponent('colloquial\_area')

[sublocality, x] = getComponent('sublocality')

address =

streetNumber: streetNum

street: street

streetAbbr: streetAbbr

city: city

state: state

stateAbbr: stateAbbr

country: country

countryAbbr: countryAbbr

postalCode: postalCode

postalCodePrefix: postalCodePrefix

colloquialArea: colloquialArea

sublocality: sublocality

location: location

\_.each(address, (val, key) =>

this[key] = val

)

else

@addressStr = address

componentFinder: (components) ->

return (type) ->

it = \_.find(components, (c) ->

return \_.contains(c.types, type)

)

return [it?.short\_name, it?.long\_name]

toString: (useCountryAbbr=true, useStateAbbr=true, useStreetAbbr=false) ->

return @addressStr if not @isObject

arr = []

stateVal = if useStateAbbr and @generated then 'stateAbbr' else 'state'

countryVal = if useCountryAbbr and @generated then 'countryAbbr' else 'country'

streetVal = if useStreetAbbr and @generated then 'streetAbbr' else 'street'

for prop in [streetVal, 'city', stateVal, countryVal]

arr.push(this[prop]) if this[prop]

str = arr.join(', ')

if @streetNumber

str = "#{this.streetNumber} #{str}"

return str

###

validate an input address.

inputAddr: validator.Address object or map with 'street', 'city', 'state', 'country' keys, or string address

cb: function(err, validAddresses, inexactMatches, geocodingResponse)

err - something went wrong calling the google api

validAddresses - list of Address objects. These are exact matches to your input, and will have proper spelling, caps etc. Its best that you use this instead of what you had

inexactMatches - list of Address objects. Incomplete addresses or addresses that do not match your input address. useful for 'did you mean?' type UIs

geocodingResponse - the json object that i got from google API

###

defaultMatchType = matchType.streetAddress

exports.validate = (inputAddr, addressType=defaultMatchType, cb) ->

if arguments.length == 2

cb = addressType

addressType = defaultMatchType

inputAddress = if inputAddr instanceof Address then inputAddr else new Address(inputAddr)

qs = {'sensor':false, 'address': inputAddress.toString(), region: options.countryBias, language: options.language}

if options.countryMatch

qs.components = "country:#{options.countryMatch}"

protocol = 'http'

if options.key

qs.key = options.key

protocol = 'https'

opts =

json: true,

url: "#{protocol}://maps.googleapis.com/maps/api/geocode/json"

method: 'GET'

qs: qs

if options.proxy

if options.key

opts.agent = new HttpsProxyAgent(options.proxy);

else

opts.proxy = options.proxy

request(opts, (err, response, body) ->

return cb(err, null, null, body) if err

return cb(new Error("Google geocode API returned status code of #{response.statusCode}"), [], [], body) if response.statusCode isnt 200

return cb(new Error("Google returned error: #{body.status} - #{body.error\_message}"), [], [], body) if body.status.toLowerCase() isnt "ok"

return cb(null, [], [], body) if body.results.length == 0

validAddresses = []

inexactMatches = []

\_.each(body.results, (result) ->

address = new Address(result)

if addressType == matchType.unknown

if matchUnknownType(address, inputAddress)

validAddresses.push(address)

else

inexactMatches.push(address)

else if addressType == address.matchType

if address.exactMatch

validAddresses.push(address)

else

inexactMatches.push(address)

)

cb(null, validAddresses, inexactMatches, body)