/\*\*

\* life360

\*

\* BTRIAL DISTANCE AND SLEEP PATCH 29-12-2017

\* Updated Code to handle distance from, and sleep functionality

\*

\* TMLEAFS REFRESH PATCH 06-12-2016 V1.1

\* Updated Code to match Smartthings updates 12-05-2017 V1.2

\* Added updateMember function that pulls all usefull information Life360 provides for webCoRE use V2.0

\*

\* Copyright 2014 Jeff's Account

\*

\* Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except

\* in compliance with the License. You may obtain a copy of the License at:

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software distributed under the License is distributed

\* on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License

\* for the specific language governing permissions and limitations under the License.

\*

\*/

definition(

name: "Life360 (Connect)",

namespace: "tmleafs",

author: "tmleafs",

description: "Life360 Service Manager",

category: "SmartThings Labs",

iconUrl: "https://s3.amazonaws.com/smartapp-icons/Partner/life360.png",

iconX2Url: "https://s3.amazonaws.com/smartapp-icons/Partner/life360@2x.png",

oauth: [displayName: "Life360", displayLink: "Life360"],

singleInstance: true

) {

appSetting "clientId"

appSetting "clientSecret"

}

preferences {

page(name: "Credentials", title: "Life360 Authentication", content: "authPage", nextPage: "listCirclesPage", install: false)

page(name: "listCirclesPage", title: "Select Life360 Circle", nextPage: "listPlacesPage", content: "listCircles", install: false)

page(name: "listPlacesPage", title: "Select Life360 Place", nextPage: "listUsersPage", content: "listPlaces", install: false)

page(name: "listUsersPage", title: "Select Life360 Users", content: "listUsers", install: true)

}

// page(name: "Credentials", title: "Enter Life360 Credentials", content: "getCredentialsPage", nextPage: "listCirclesPage", install: false)

// page(name: "page3", title: "Select Life360 Users", content: "listUsers")

mappings {

path("/placecallback") {

action: [

POST: "placeEventHandler",

GET: "placeEventHandler"

]

}

path("/receiveToken") {

action: [

POST: "receiveToken",

GET: "receiveToken"

]

}

}

def authPage()

{

log.debug "authPage()"

def description = "Life360 Credentials Already Entered."

def uninstallOption = false

if (app.installationState == "COMPLETE")

uninstallOption = true

if(!state.life360AccessToken)

{

log.debug "about to create access token"

createAccessToken()

description = "Click to enter Life360 Credentials."

def redirectUrl = oauthInitUrl()

return dynamicPage(name: "Credentials", title: "Life360", nextPage:"listCirclesPage", uninstall: uninstallOption, install:false) {

section {

href url:redirectUrl, style:"embedded", required:false, title:"Life360", description:description

}

}

}

else

{

listCircles()

}

}

def receiveToken() {

state.life360AccessToken = params.access\_token

def html = """

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=640">

<title>Withings Connection</title>

<style type="text/css">

@font-face {

font-family: 'Swiss 721 W01 Thin';

src: url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-thin-webfont.eot');

src: url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-thin-webfont.eot?#iefix') format('embedded-opentype'),

url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-thin-webfont.woff') format('woff'),

url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-thin-webfont.ttf') format('truetype'),

url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-thin-webfont.svg#swis721\_th\_btthin') format('svg');

font-weight: normal;

font-style: normal;

}

@font-face {

font-family: 'Swiss 721 W01 Light';

src: url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-light-webfont.eot');

src: url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-light-webfont.eot?#iefix') format('embedded-opentype'),

url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-light-webfont.woff') format('woff'),

url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-light-webfont.ttf') format('truetype'),

url('https://s3.amazonaws.com/smartapp-icons/Partner/fonts/swiss-721-light-webfont.svg#swis721\_lt\_btlight') format('svg');

font-weight: normal;

font-style: normal;

}

.container {

width: 560px;

padding: 40px;

/\*background: #eee;\*/

text-align: center;

}

img {

vertical-align: middle;

}

img:nth-child(2) {

margin: 0 30px;

}

p {

font-size: 2.2em;

font-family: 'Swiss 721 W01 Thin';

text-align: center;

color: #666666;

padding: 0 40px;

margin-bottom: 0;

}

/\*

p:last-child {

margin-top: 0px;

}

\*/

span {

font-family: 'Swiss 721 W01 Light';

}

</style>

</head>

<body>

<div class="container">

<img src="https://s3.amazonaws.com/smartapp-icons/Partner/life360@2x.png" alt="Life360 icon" />

<img src="https://s3.amazonaws.com/smartapp-icons/Partner/support/connected-device-icn%402x.png" alt="connected device icon" />

<img src="https://s3.amazonaws.com/smartapp-icons/Partner/support/st-logo%402x.png" alt="SmartThings logo" />

<p>Your Life360 Account is now connected to SmartThings!</p>

<p>Click 'Done' to finish setup.</p>

</div>

</body>

</html>

"""

render contentType: 'text/html', data: html

}

def oauthInitUrl()

{

log.debug "oauthInitUrl"

def stcid = getSmartThingsClientId();

// def oauth\_url = "https://api.life360.com/v3/oauth2/authorize?client\_id=pREqugabRetre4EstetherufrePumamExucrEHuc&response\_type=token&redirect\_uri=http%3A%2F%2Fwww.smartthings.com"

state.oauthInitState = UUID.randomUUID().toString()

def oauthParams = [

response\_type: "token",

client\_id: stcid,

redirect\_uri: buildRedirectUrl()

]

return "https://api.life360.com/v3/oauth2/authorize?" + toQueryString(oauthParams)

}

String toQueryString(Map m) {

return m.collect { k, v -> "${k}=${URLEncoder.encode(v.toString())}" }.sort().join("&")

}

def getSmartThingsClientId() {

return "pREqugabRetre4EstetherufrePumamExucrEHuc"

}

def getServerUrl() { getApiServerUrl() }

def buildRedirectUrl()

{

log.debug "buildRedirectUrl"

// /api/token/:st\_token/smartapps/installations/:id/something

return serverUrl + "/api/token/${state.accessToken}/smartapps/installations/${app.id}/receiveToken"

}

//

// This method is no longer used - was part of the initial username/password based authentication that has now been replaced

// by the full OAUTH web flow

//

def getCredentialsPage() {

dynamicPage(name: "Credentials", title: "Enter Life360 Credentials", nextPage: "listCirclesPage", uninstall: true, install:false)

{

section("Life 360 Credentials ...") {

input "username", "text", title: "Life360 Username?", multiple: false, required: true

input "password", "password", title: "Life360 Password?", multiple: false, required: true, autoCorrect: false

}

}

}

//

// This method is no longer used - was part of the initial username/password based authentication that has now been replaced

// by the full OAUTH web flow

//

def getCredentialsErrorPage(String message) {

dynamicPage(name: "Credentials", title: "Enter Life360 Credentials", nextPage: "listCirclesPage", uninstall: true, install:false)

{

section("Life 360 Credentials ...") {

input "username", "text", title: "Life360 Username?", multiple: false, required: true

input "password", "password", title: "Life360 Password?", multiple: false, required: true, autoCorrect: false

paragraph "${message}"

}

}

}

def testLife360Connection() {

if (state.life360AccessToken)

true

else

false

}

//

// This method is no longer used - was part of the initial username/password based authentication that has now been replaced

// by the full OAUTH web flow

//

def initializeLife360Connection() {

def oauthClientId = appSettings.clientId

def oauthClientSecret = appSettings.clientSecret

initialize()

def username = settings.username

def password = settings.password

// Base 64 encode the credentials

def basicCredentials = "${oauthClientId}:${oauthClientSecret}"

def encodedCredentials = basicCredentials.encodeAsBase64().toString()

// call life360, get OAUTH token using password flow, save

// curl -X POST -H "Authorization: Basic cFJFcXVnYWJSZXRyZTRFc3RldGhlcnVmcmVQdW1hbUV4dWNyRUh1YzptM2ZydXBSZXRSZXN3ZXJFQ2hBUHJFOTZxYWtFZHI0Vg=="

// -F "grant\_type=password" -F "username=jeff@hagins.us" -F "password=tondeleo" https://api.life360.com/v3/oauth2/token.json

def url = "https://api.life360.com/v3/oauth2/token.json"

def postBody = "grant\_type=password&" +

"username=${username}&"+

"password=${password}"

def result = null

try {

httpPost(uri: url, body: postBody, headers: ["Authorization": "Basic ${encodedCredentials}" ]) {response ->

result = response

}

if (result.data.access\_token) {

state.life360AccessToken = result.data.access\_token

return true;

}

log.info "Life360 initializeLife360Connection, response=${result.data}"

return false;

}

catch (e) {

log.error "Life360 initializeLife360Connection, error: $e"

return false;

}

}

def listCircles(){

// understand whether to present the Uninstall option

def uninstallOption = false

if (app.installationState == "COMPLETE")

uninstallOption = true

// get connected to life360 api

if (testLife360Connection()) {

// now pull back the list of Life360 circles

// curl -X GET -H "Authorization: Bearer MmEzODQxYWQtMGZmMy00MDZhLWEwMGQtMTIzYmYxYzFmNGU3" https://api.life360.com/v3/circles.json

def url = "https://api.life360.com/v3/circles.json"

def result = null

httpGet(uri: url, headers: ["Authorization": "Bearer ${state.life360AccessToken}" ]) {response ->

result = response

}

log.debug "Circles=${result.data}"

def circles = result.data.circles

if (circles.size > 1) {

return (

dynamicPage(name: "listCirclesPage", title: "Life360 Circles", nextPage: null, uninstall: uninstallOption, install:false) {

section("Select Life360 Circle:") {

input "circle", "enum", multiple: false, required:true, title:"Life360 Circle: ", options: circles.collectEntries{[it.id, it.name]}

}

}

)

}

else {

state.circle = circles[0].id

return (listPlaces())

}

}

else {

getCredentialsErrorPage("Invalid Usernaname or password.")

}

}

def listPlaces() {

// understand whether to present the Uninstall option

def uninstallOption = false

if (app.installationState == "COMPLETE")

uninstallOption = true

if (!state?.circle)

state.circle = settings.circle

// call life360 and get the list of places in the circle

def url = "https://api.life360.com/v3/circles/${state.circle}/places.json"

def result = null

httpGet(uri: url, headers: ["Authorization": "Bearer ${state.life360AccessToken}" ]) {response ->

result = response

}

log.debug "Places=${result.data}"

def places = result.data.places

state.places = places

// If there is a place called "Home" use it as the default

def defaultPlace = places.find{it.name=="Home"}

def defaultPlaceId

if (defaultPlace) {

defaultPlaceId = defaultPlace.id

log.debug "Place = $defaultPlace.name, Id=$defaultPlace.id"

}

dynamicPage(name: "listPlacesPage", title: "Life360 Places", nextPage: null, uninstall: uninstallOption, install:false) {

section("Select Life360 Place to Match Current Location:") {

paragraph "Please select the ONE Life360 Place that matches your SmartThings location: ${location.name}"

input "place", "enum", multiple: false, required:true, title:"Life360 Places: ", options: places.collectEntries{[it.id, it.name]}, defaultValue: defaultPlaceId

}

}

}

def listUsers() {

// understand whether to present the Uninstall option

def uninstallOption = false

if (app.installationState == "COMPLETE")

uninstallOption = true

if (!state?.circle)

state.circle = settings.circle

// call life360 and get list of users (members)

def url = "https://api.life360.com/v3/circles/${state.circle}/members.json"

def result = null

httpGet(uri: url, headers: ["Authorization": "Bearer ${state.life360AccessToken}" ]) {response ->

result = response

}

log.debug "Members=${result.data}"

// save members list for later

def members = result.data.members

state.members = members

// build preferences page

dynamicPage(name: "listUsersPage", title: "Life360 Users", nextPage: null, uninstall: uninstallOption, install:true) {

section("Select Life360 Users to Import into SmartThings:") {

input "users", "enum", multiple: true, required:true, title:"Life360 Users: ", options: members.collectEntries{[it.id, it.firstName+" "+it.lastName]}

}

}

}

def installed() {

if (!state?.circle)

state.circle = settings.circle

log.debug "In installed() method."

// log.debug "Members: ${state.members}"

// log.debug "Users: ${settings.users}"

settings.users.each {memberId->

// log.debug "Find by Member Id = ${memberId}"

def member = state.members.find{it.id==memberId}

// log.debug "After Find Attempt."

// log.debug "Member Id = ${member.id}, Name = ${member.firstName} ${member.lastName}, Email Address = ${member.loginEmail}"

// log.debug "External Id=${app.id}:${member.id}"

// create the device

if (member) {

def childDevice = addChildDevice("tmleafs", "Life360 User", "${app.id}.${member.id}",null,[name:member.firstName, completedSetup: true])

// save the memberId on the device itself so we can find easily later

// childDevice.setMemberId(member.id)

if (childDevice)

{

// log.debug "Child Device Successfully Created"

generateInitialEvent (member, childDevice)

// build the icon name form the L360 Avatar URL

// URL Format: https://www.life360.com/img/user\_images/b4698717-1f2e-4b7a-b0d4-98ccfb4e9730/Maddie\_Hagins\_51d2eea2019c7.jpeg

// SmartThings Icon format is: L360.b4698717-1f2e-4b7a-b0d4-98ccfb4e9730.Maddie\_Hagins\_51d2eea2019c7

try {

// build the icon name from the avatar URL

log.debug "Avatar URL = ${member.avatar}"

def urlPathElements = member.avatar.tokenize("/")

def fileElements = urlPathElements[5].tokenize(".")

// def icon = "st.Lighting.light1"

def icon="l360.${urlPathElements[4]}.${fileElements[0]}"

log.debug "Icon = ${icon}"

// set the icon on the device

childDevice.setIcon("presence","present",icon)

childDevice.setIcon("presence","not present",icon)

childDevice.save()

}

catch (e) { // do nothing

log.debug "Error = ${e}"

}

}

}

}

createCircleSubscription()

}

def createCircleSubscription() {

// delete any existing webhook subscriptions for this circle

//

// curl -X DELETE https://webhook.qa.life360.com/v3/circles/:circleId/webhook.json

log.debug "Remove any existing Life360 Webhooks for this Circle."

def deleteUrl = "https://api.life360.com/v3/circles/${state.circle}/webhook.json"

try { // ignore any errors - there many not be any existing webhooks

httpDelete (uri: deleteUrl, headers: ["Authorization": "Bearer ${state.life360AccessToken}" ]) {response ->

result = response}

}

catch (e) {

log.debug (e)

}

// subscribe to the life360 webhook to get push notifications on place events within this circle

// POST /circles/:circle\_id/places/webooks

// Params: hook\_url

log.debug "Create a new Life360 Webhooks for this Circle."

createAccessToken() // create our own OAUTH access token to use in webhook url

def hookUrl = "${serverUrl}/api/smartapps/installations/${app.id}/placecallback?access\_token=${state.accessToken}".encodeAsURL()

def url = "https://api.life360.com/v3/circles/${state.circle}/webhook.json"

def postBody = "url=${hookUrl}"

def result = null

try {

httpPost(uri: url, body: postBody, headers: ["Authorization": "Bearer ${state.life360AccessToken}" ]) {response ->

result = response}

} catch (e) {

log.debug (e)

}

// response from this call looks like this:

// {"circleId":"41094b6a-32fc-4ef5-a9cd-913f82268836","userId":"0d1db550-9163-471b-8829-80b375e0fa51","clientId":"11",

// "hookUrl":"https://testurl.com"}

log.debug "Response = ${response}"

if (result.data?.hookUrl) {

log.debug "Webhook creation successful. Response = ${result.data}"

}

}

def updated() {

if (!state?.circle)

state.circle = settings.circle

log.debug "In updated() method."

// log.debug "Members: ${state.members}"

// log.debug "Users: ${settings.users}"

// loop through selected users and try to find child device for each

settings.users.each {memberId->

def externalId = "${app.id}.${memberId}"

// find the appropriate child device based on my app id and the device network id

def deviceWrapper = getChildDevice("${externalId}")

if (!deviceWrapper) { // device isn't there - so we need to create

// log.debug "Find by Member Id = ${memberId}"

def member = state.members.find{it.id==memberId}

// log.debug "After Find Attempt."

// log.debug "External Id=${app.id}:${member.id}"

// create the device

def childDevice = addChildDevice("tmleafs", "Life360 User", "${app.id}.${member.id}",null,[name:member.firstName, completedSetup: true])

// childDevice.setMemberId(member.id)

if (childDevice)

{

// log.debug "Child Device Successfully Created"

generateInitialEvent (member, childDevice)

// build the icon name form the L360 Avatar URL

// URL Format: https://www.life360.com/img/user\_images/b4698717-1f2e-4b7a-b0d4-98ccfb4e9730/Maddie\_Hagins\_51d2eea2019c7.jpeg

// SmartThings Icon format is: L360.b4698717-1f2e-4b7a-b0d4-98ccfb4e9730.Maddie\_Hagins\_51d2eea2019c7

try {

// build the icon name from the avatar URL

log.debug "Avatar URL = ${member.avatar}"

def urlPathElements = member.avatar.tokenize("/")

def icon="l360.${urlPathElements[4]}.${urlPathElements[5]}"

// set the icon on the device

childDevice.setIcon("presence","present",icon)

childDevice.setIcon("presence","not present",icon)

childDevice.save()

}

catch (e) { // do nothing

log.debug "Error = ${e}"

}

}

}

else {

// log.debug "Find by Member Id = ${memberId}"

def member = state.members.find{it.id==memberId}

generateInitialEvent (member, deviceWrapper)

}

}

// Now remove any existing devices that represent users that are no longer selected

def childDevices = getAllChildDevices()

log.debug "Child Devices = ${childDevices}"

childDevices.each {childDevice->

log.debug "Child = ${childDevice}, DNI=${childDevice.deviceNetworkId}"

// def childMemberId = childDevice.getMemberId()

def splitStrings = childDevice.deviceNetworkId.split("\\.")

log.debug "Strings = ${splitStrings}"

def childMemberId = splitStrings[1]

log.debug "Child Member Id = ${childMemberId}"

log.debug "Settings.users = ${settings.users}"

if (!settings.users.find{it==childMemberId}) {

deleteChildDevice(childDevice.deviceNetworkId)

def member = state.members.find {it.id==memberId}

if (member)

state.members.remove(member)

}

}

}

def generateInitialEvent (member, childDevice) {

runEvery1Minute(updateMembers)

// lets figure out if the member is currently "home" (At the place)

try { // we are going to just ignore any errors

log.info "Life360 generateInitialEvent($member, $childDevice)"

def place = state.places.find{it.id==settings.place}

if (place) {

def memberLatitude = new Float (member.location.latitude)

def memberLongitude = new Float (member.location.longitude)

def memberAddress1 = member.location.address1

def memberLocationName = member.location.name

def placeLatitude = new Float (place.latitude)

def placeLongitude = new Float (place.longitude)

def placeRadius = new Float (place.radius)

log.debug "Member Location = ${memberLatitude}/${memberLongitude}"

log.debug "Place Location = ${placeLatitude}/${placeLongitude}"

log.debug "Place Radius = ${placeRadius}"

def distanceAway = haversine(memberLatitude, memberLongitude, placeLatitude, placeLongitude)\*1000 // in meters

log.debug "Distance Away = ${distanceAway}"

boolean isPresent = (distanceAway <= placeRadius)

log.info "Life360 generateInitialEvent, member: ($memberLatitude, $memberLongitude), place: ($placeLatitude, $placeLongitude), radius: $placeRadius, dist: $distanceAway, present: $isPresent"

// log.debug "External Id=${app.id}:${member.id}"

// def childDevice2 = getChildDevice("${app.id}.${member.id}")

// log.debug "Child Device = ${childDevice2}"

def address1

def address2

def speed

def speedmeters

def speedMPH

def speedKPH

if(member.location.address1 == null || member.location.address1 == "")

address1 = "No Data"

else

address1 = member.location.address1

if(member.location.address2 == null || member.location.address2 == "")

address2 = "No Data"

else

address2 = member.location.address2

//Covert 0 1 to False True

def charging = member.location.charge == "0" ? "false" : "true"

def moving = member.location.inTransit == "0" ? "false" : "true"

def driving = member.location.isDriving == "0" ? "false" : "true"

def wifi = member.location.wifiState == "0" ? "false" : "true"

//Fix Iphone -1 speed

if(member.location.speed.toFloat() == -1){

speed = 0

speed = speed.toFloat()}

else

speed = member.location.speed.toFloat()

if(speed > 0 ){

speedmeters = speed.toDouble().round(2)

speedMPH = speedmeters.toFloat() \* 2.23694

speedMPH = speedMPH.toDouble().round(2)

speedKPH = speedmeters.toFloat() \* 3.6

speedKPH = speedKPH.toDouble().round(2)

}else{

speedmeters = 0

speedMPH = 0

speedKPH = 0

}

def battery = Math.round(member.location.battery.toDouble())

def latitude = member.location.latitude.toFloat()

def longitude = member.location.longitude.toFloat()

//Sent data

childDevice?.extraInfo(address1,address2,battery,charging,member.location.endTimestamp,moving,driving,latitude,longitude,member.location.since,speedmeters,speedMPH,speedKPH,wifi)

//childDevice?.extraInfo(member.location.address1,member.location.address2,member.location.battery,member.location.charge,member.location.endTimestamp,member.location.inTransit,member.location.isDriving,member.location.latitude,member.location.longitude,member.location.since,member.location.speed,member.location.wifiState)

childDevice?.generatePresenceEvent(isPresent, distanceAway)

// log.debug "After generating presence event."

}

}

catch (e) {

// eat it

}

}

def initialize() {

// TODO: subscribe to attributes, devices, locations, etc.

}

def haversine(lat1, lon1, lat2, lon2) {

def R = 6372.8

// In kilometers

def dLat = Math.toRadians(lat2 - lat1)

def dLon = Math.toRadians(lon2 - lon1)

lat1 = Math.toRadians(lat1)

lat2 = Math.toRadians(lat2)

def a = Math.sin(dLat / 2) \* Math.sin(dLat / 2) + Math.sin(dLon / 2) \* Math.sin(dLon / 2) \* Math.cos(lat1) \* Math.cos(lat2)

def c = 2 \* Math.asin(Math.sqrt(a))

def d = R \* c

return(d)

}

def placeEventHandler() {

log.info "Life360 placeEventHandler: params=$params"

log.info "Life360 placeEventHandler: settings.place=$settings.place"

// the POST to this end-point will look like:

// POST http://test.com/webhook?circleId=XXXX&placeId=XXXX&userId=XXXX&direction=arrive

def circleId = params?.circleId

def placeId = params?.placeId

def userId = params?.userId

def direction = params?.direction

def timestamp = params?.timestamp

if (placeId == settings.place) {

def presenceState = (direction=="in")

def externalId = "${app.id}.${userId}"

// find the appropriate child device based on my app id and the device network id

def deviceWrapper = getChildDevice("${externalId}")

// invoke the generatePresenceEvent method on the child device

if (deviceWrapper) {

deviceWrapper.generatePresenceEvent(presenceState, 0)

log.debug "Life360 event raised on child device: ${externalId}"

}

else {

log.warn "Life360 couldn't find child device associated with inbound Life360 event."

}

}

}

def refresh() {

listCircles()

listPlaces()

listUsers()

updated()

}

def updateMembers(){

if (!state?.circle)

state.circle = settings.circle

def url = "https://api.life360.com/v3/circles/${state.circle}/members.json"

def result = null

httpGet(uri: url, headers: ["Authorization": "Bearer ${state.life360AccessToken}" ]) {response ->

result = response

}

//log.debug "Latest Members=${result.data}"

def members = result.data.members

state.members = members

settings.users.each {memberId->

//log.debug "appid $app.id memberid $memberId"

def externalId = "${app.id}.${memberId}"

//log.debug "ExternalId = $externalId"

def member = state.members.find{it.id==memberId}

//log.debug "member = $member"

// find the appropriate child device based on my app id and the device network id

def deviceWrapper = getChildDevice("${externalId}")

def address1

def address2

def speed

def speedMetric

def speedMiles

def speedKm

if(member.location.address1 == null || member.location.address1 == "")

address1 = "No Data"

else

address1 = member.location.address1

if(member.location.address2 == null || member.location.address2 == "")

address2 = "No Data"

else

address2 = member.location.address2

//Covert 0 1 to False True

def charging = member.location.charge == "0" ? "false" : "true"

def moving = member.location.inTransit == "0" ? "false" : "true"

def driving = member.location.isDriving == "0" ? "false" : "true"

def wifi = member.location.wifiState == "0" ? "false" : "true"

//Fix Iphone -1 speed

if(member.location.speed.toFloat() == -1){

speed = 0

speed = speed.toFloat()}

else

speed = member.location.speed.toFloat()

if(speed > 0 ){

speedMetric = speed.toDouble().round(2)

speedMiles = speedMetric.toFloat() \* 2.23694

speedMiles = speedMiles.toDouble().round(2)

speedKm = speedMetric.toFloat() \* 3.6

speedKm = speedKm.toDouble().round(2)

}else{

speedMetric = 0

speedMiles = 0

speedKm = 0

}

def battery = Math.round(member.location.battery.toDouble())

def latitude = member.location.latitude.toFloat()

def longitude = member.location.longitude.toFloat()

//log.debug "extrainfo = Address 1 = $address1 | Address 2 = $address2 | Battery = $battery | Charging = $charging | Last Checkin = $member.location.endTimestamp | Moving = $moving | Driving = $driving | Latitude = $latitude | Longitude = $longitude | Since = $member.location.since | Speedmeters = $speedMetric | SpeedMPH = $speedMiles | SpeedKPH = $speedKm | Wifi = $wifi"

deviceWrapper.extraInfo(address1,address2,battery,charging,member.location.endTimestamp,moving,driving,latitude,longitude,member.location.since,speedMetric,speedMiles,speedKm,wifi)

def place = state.places.find{it.id==settings.place}

if (place) {

def memberLatitude = new Float (member.location.latitude)

def memberLongitude = new Float (member.location.longitude)

def memberAddress1 = member.location.address1

def memberLocationName = member.location.name

def placeLatitude = new Float (place.latitude)

def placeLongitude = new Float (place.longitude)

//Radius is 20% larger if currently present, otherwise normal radius

def placeRadius = (deviceWrapper.currentValue('presence').toLowerCase() == 'present') ? new Float(place.radius) \* 1.20 : new Float(place.radius)

//def placeRadius = new Float (place.radius)

//log.debug "Member Location = ${memberLatitude}/${memberLongitude}"

//log.debug "Place Location = ${placeLatitude}/${placeLongitude}"

//log.debug "Place Radius = ${placeRadius}"

def distanceAway = haversine(memberLatitude, memberLongitude, placeLatitude, placeLongitude)\*1000 // in meters

//log.debug "Distance Away = ${distanceAway}"

boolean isPresent = (distanceAway <= placeRadius)

log.info "Life360 Update member ($member.firstName): ($memberLatitude, $memberLongitude), place: ($placeLatitude, $placeLongitude), radius: $placeRadius, dist: $distanceAway, present: $isPresent"

deviceWrapper.generatePresenceEvent(isPresent, distanceAway)

}

}

}