Android

Step 1: Updated my dependency in build.gradle

implementation(libs.play.services.location)

```
dependencies {
    implementation 'com.google.android.gms:play-services-location:21.3.0'
}
```

step 2: Add permissions to be used in AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission
android:name="android.permission.FOREGROUND_SERVICE_LOCATION" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission
android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission
android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS" />
```

Step 3: Create an interface class like following

```
interface MyLocationClient {
    fun getLocationUpdates(interval: Long): Flow<Location>

    class AnyException(message: String): Exception()
}
```

Step 4: Now create a class with MyDefaultLocationClient.kt,and check permission function

```
interface MyLocationClient {
    fun getLocationUpdates(interval: Long): Flow<Location>

    class AnyException(message: String) : Exception()
}


class MyDefaultLocationClient(
    private val context: Context,
    private val client: FusedLocationProviderClient
) : MyLocationClient {
    @SuppressLint("MissingPermission")
    @OptIn(ExperimentalCoroutinesApi::class)
    override fun getLocationUpdates(interval: Long): Flow<Location> {
        return callbackFlow {


            //-------------------------------------------------------------
-------------------------------------------
            /// check permissions
            if (!context.hasLocationPermissions()) {
                throw MyLocationClient.AnyException("Missing Permissions
for Location")
            }
            val locationManager =
context.getSystemService(Context.LOCATION_SERVICE) as LocationManager
            val isGpsEnabled =
locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)
            val isNetworkEnabled =
locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER)
            if (!isGpsEnabled && !isNetworkEnabled) {
                throw MyLocationClient.AnyException("GPS is disabled")
            }
            //-------------------------------------------------------------
-------------------------------------------



            //-------------------------------------------------------------
-------------------------------------------
            /// create location tracker
            val request = LocationRequest.Builder(interval)
                .setMinUpdateDistanceMeters(1f)
                .setIntervalMillis(1L)
                .setWaitForAccurateLocation(false)
                .setPriority(100)
                .build()
            //-------------------------------------------------------------
-------------------------------------------



            //-------------------------------------------------------------
-------------------------------------------
            //location callback
            val locationCallback = object : LocationCallback() {
                override fun onLocationResult(p0: LocationResult) {
                    super.onLocationResult(p0)
                    p0.locations.lastOrNull()?.let {
                        Log.i("", p0.locations.toString())
                        Counter(context).incrementFunctionCount()

        //here you can add the task
                        //for notifications
                        launch { send(it) }

                    }
                }
            }
            //-------------------------------------------------------------
-------------------------------------------


            //request location updates
            client.requestLocationUpdates(request, locationCallback,
Looper.getMainLooper())


            //remove location updates
            awaitClose { client.removeLocationUpdates(locationCallback) }

        }
    }


    //check permissions
    private fun Context.hasLocationPermissions(): Boolean {
        return ContextCompat.checkSelfPermission(
            this,
            android.Manifest.permission.ACCESS_FINE_LOCATION
        ) == PackageManager.PERMISSION_GRANTED
                &&
                ContextCompat.checkSelfPermission(
                    this,
                    android.Manifest.permission.ACCESS_COARSE_LOCATION
                ) == PackageManager.PERMISSION_GRANTED
    }

}
```

Step 5: Now define a Service class with name: LocationTrackingService.kt

and
Step 6: Define your notification channel in Application class

```
package com.example.ios_android_background.myBackgroundLocationTracker




class LocationTrackingService: Service() {

    private val serviceScope = CoroutineScope(SupervisorJob() +
Dispatchers.IO)
    private lateinit var  myLocationClient: MyLocationClient

    override fun onBind(p0: Intent?): IBinder? { return null }



    override fun onCreate() {
        super.onCreate()

        //[1]create notification channel
        createNotificationChannel()



        //[2]create location client
        myLocationClient = MyDefaultLocationClient(applicationContext,
LocationServices.getFusedLocationProviderClient(applicationContext))



    }

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int):
Int {
        when(intent?.action){
            ACTION_START -> start()
            ACTION_STOP -> stop()
        }
        return super.onStartCommand(intent, flags, startId)
    }

    private fun start(){
        Log.i("","Start Service")
        print("Start Service")


        //-----------------------------------------------------------------
------------------------------------
        //[]edit notification content
        val notification = NotificationCompat.Builder(this, "location")
            .setContentTitle("Tracking Location")
            .setContentText("Location: null")
            .setSmallIcon(R.drawable.ic_launcher_background)
            .setOngoing(true).setPriority(100)
        val notificationManager =
getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        //-------------------------------------------------------------------
------------------------------------







        //-------------------------------------------------------------------
------------------------------------
        // create location tracker and get location updates
        myLocationClient.getLocationUpdates(1L)
            .catch { e -> e.printStackTrace() }
            .onEach {
                val lat = it.latitude.toString()
                val long = it.longitude.toString()
                val updateNotification =
notification.setContentText("Location: ($lat, $long)")
                notificationManager.notify(1, updateNotification.build())



        //here add the task




            }
            .launchIn(serviceScope)
        //-------------------------------------------------------------------
------------------------------------








        startForeground(1, notification.build())
    }

    private fun stop(){
        Log.i("","stop service")
        stopForeground(true)
        stopSelf()
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.i("","onDestroy")
        serviceScope.cancel()
    }

    companion object{
        const val ACTION_START = "ACTION_START"
        const val ACTION_STOP = "ACTION_STOP"
    }


    private fun createNotificationChannel(){
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel("location", "location",
NotificationManager.IMPORTANCE_HIGH)
            val notificationManager =
getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
            notificationManager.createNotificationChannel(channel)



        }
    }
}
```

Step 7: add service in manifest.xml

```
<service
android:name=".myBackgroundLocationTracker.LocationTrackingService"
    android:foregroundServiceType="location"
    android:exported="false"
    tools:ignore="ForegroundServicePermission"
    />
```

Step 8: Ask for Location permission in your mainActivity.kt

```
ActivityCompat.requestPermissions(
    this,
    arrayOf(
        android.Manifest.permission.ACCESS_FINE_LOCATION,
        android.Manifest.permission.ACCESS_COARSE_LOCATION,
        android.Manifest.permission.POST_NOTIFICATIONS,
    ),
    111
)
```

Step 9: Create channel and start the service

```
private val CHANNEL = "myLocationTracking"
MethodChannel(flutterEngine.dartExecutor.binaryMessenger, CHANNEL).setMethodCallHandler { call, result ->



  if (call.method == "startService") {

    try {
      val intent = Intent(this, LocationTrackingService::class.java)
      startService(intent)
      Intent(applicationContext, LocationTrackingService::class.java).apply {
        action = LocationTrackingService.ACTION_START
        startService(this)
      }
    }catch (e:Exception){ Log.i("",e.toString()) }

    result.success(1)
  }

  else {
    result.notImplemented()
  }
}
```

Step 10: invoke the start method from your ui

```
var platform = const MethodChannel("myLocationTracking");

startService() async {
  try {
    await platform.invokeMethod('startService');

  } on PlatformException catch (e) {
    print("Failed to get count: '${e.message}'.");
    return 0;
  }
}
```

Ios

Permissions and info.plist

```
<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
<string>We need your location to provide services at all times.</string>
<key>NSLocationAlwaysUsageDescription</key>
<string>We need your location to track your position in the
background.</string>
<key>NSLocationWhenInUseUsageDescription</key>
<string>We need your location to provide services while the app is in
use.</string>
<key>UIApplicationSupportsIndirectInputEvents</key>
<true/>
<key>UIBackgroundModes</key>
<array>
    <string>location</string>
    <string>processing</string>
    <string>fetch</string>
</array>
```

Step 1: Create Class and extend CLLocationManagerDelegate

```swift
import BackgroundTasks
import Flutter
import UIKit
import CoreLocation

class LocationManager: NSObject, CLLocationManagerDelegate
{



    let locationManager = CLLocationManager()

    override init() {
        super.init()




        locationManager.delegate = self
        requestLocationPermission()
        locationManager.allowsBackgroundLocationUpdates=true
        locationManager.desiredAccuracy = kCLLocationAccuracyBest
        locationManager.pausesLocationUpdatesAutomatically = false
        locationManager.activityType = .fitness



        print("here")




        locationManager.showsBackgroundLocationIndicator = true

        startSignificantChangeUpdates()
        startTrackingLocation()
    }

    // Request permission from the user
    func requestLocationPermission() {

        print("requestLocationPermission")
        locationManager.requestAlwaysAuthorization()  // Request "Always"
authorization
    }

    // Start updating the location
    func startTrackingLocation() {
        print("startTrackingLocation")
        locationManager.startUpdatingLocation()
    }

    // Stop updating the location
    func stopTrackingLocation() {
        print("stopTrackingLocation")
        locationManager.stopUpdatingLocation()
    }

    // CLLocationManagerDelegate method - called when a new location is
received
    func locationManager(_ manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
        print("here....")
        guard let location = locations.last else { return }
        print("New location: \(location.coordinate.latitude),
\(location.coordinate.longitude)")



        //here add the task

        // Here you can send the location data to your server, update the
UI, etc.
    }

    // Handle any errors
    func locationManager(_ manager: CLLocationManager, didFailWithError
error: Error) {
        print("Location update failed with error:
\(error.localizedDescription)")
    }

    func startSignificantChangeUpdates() {
        locationManager.startMonitoringSignificantLocationChanges()
    }



    func locationManagerDidChangeAuthorization(_ manager:
CLLocationManager) {


            switch manager.authorizationStatus {
            case .authorizedAlways:
                print("Location authorized: Always")

                manager.startUpdatingLocation()
                startSignificantChangeUpdates()
                startTrackingLocation()
            case .authorizedWhenInUse:
                print("Location authorized: When In Use")
                // You can prompt the user to allow "Always" for background
tracking
            case .denied, .restricted:
                print("Location permission denied or restricted")
                // Handle the case where the user has denied location
services
            case .notDetermined:
                print("Location permission not determined yet")
            @unknown default:
                break
            }
```

```
}


}
```

Step 2: create channel and put in it method to start tracking in your AppDelegate.swift

```
  var locationManager: LocationManager?

//start service
  let controller = window?.rootViewController as! FlutterViewController
  let nativeChannel = FlutterMethodChannel(name:
"myLocationTracking",binaryMessenger: controller.binaryMessenger)
  nativeChannel.setMethodCallHandler {
      (call: FlutterMethodCall, result: @escaping FlutterResult) in
            if call.method == "startService" {
                locationManager = LocationManager()
                locationManager?.startTrackingLocation()

                result(1)
            } else {
                result(FlutterMethodNotImplemented)
            }
        }
```

Step 3: invoke the start method from your ui

```
var platform = const MethodChannel("myLocationTracking");

startService() async {
  try {
    await platform.invokeMethod('startService');

  } on PlatformException catch (e) {
    print("Failed to get count: '${e.message}'.");
    return 0;
  }
}
```