## Why use StepCounter Pro?

Step counting in Unity should be straightforward, but in reality, it often isn't. Inconsistent behavior between devices, especially in how and if they count background steps, causes developers to create convoluted workarounds for this seemingly trivial task. Furthermore there are no good ways to know when the steps were taken. This means developers only really have full control over the steps taken while the app is open and active.

`StepCounter Pro` aims to fix these issues and make step counting as easy as it should be. `StepCounter Pro` tracks the users steps in the background in a battery efficient way, even while the app is closed and even after device restart. This gives you complete access to the users step history the past 7 days on iOS and 10 days on Android. In addition, the plugin provides helpful methods for dealing with permissions and settings.

`StepCounter Pro` provides a unified interface for step counting functionality across Android and iOS platforms. It abstracts the platform-specific implementations, enabling developers to easily integrate step counting into their cross-platform applications built with Unity. The plugin handles permission requests, platform detection, and interaction with the native step counting APIs on each platform.

## Requirements

- **Unity**: 2021.3+ (Confirmed on versions, 2021.3.22, 2022.3.38, 6000.0.12)

- **Android**: Requires Google Play Services. This may affect apps distributed outside Google Play.
- **Android**: Requires minimum API level 23 or higher
- **iOS**: Requires iOS 11 or higher

### Known limitations

- Step count is limited to the last 7 days on iOS and 10 days on Android.
- Steps previous to installing app and enabling plugin are not available on Android.

---

## Getting started

In this example, we'll create an instance of the `StepCounterPro` class in the `Start()` method of a GameObject, retrieve today's step count and handle permissions.

### Step 1: Configuration

If you have not already, create a new project, download and import the plugin through the Asset Store or Package Manager.

**Android:** Set `Minimum API Level` to `Android 8.0 'Oreo' (API level 26)`

### Step 2: Create a GameObject:

Create an empty GameObject in your scene and create a new c# script. In the `Start` method, create a `StepCounterRequest` object. This object is the gateway to all of `StepCounter Pro's` functionality.

```
using Repforge.StepCounterPro;

public class Example : MonoBehaviour {

        void Start()
    {
            StepCounterRequest request = new StepCounterRequest();
    }

...
```

### Step 3: Finding today's step count

All queries consist of a start date and an end date. `StepCounter Pro` provides several helpful methods to intuitively set these up. To find the steps since a given time, e.g. midnight, we can use the `Since(fromDate DateTime)` method.

We also need to define what to do with the result. The `OnQuerySuccess(Action<int>)` lets us provide a callback function which takes in the step count as an int parameter . In this example we use a simple lambda expression to print out the value.

Finally, to execute the query we call the `Execute()` method. Since the query parameter methods return back the instance they can be chained together like this:

```
public class Example : MonoBehaviour {

        void Start()
    {
            StepCounterRequest request = new StepCounterRequest();
            request
```

```
                        .Since(DateTime.Today)
                        .OnQuerySuccess((stepCount) => Debug.Log("Steps today: " + stepCount))
                        .Execute();
    }
```

### Step 4: Handling permissions

Before accessing step counter data, the user must grant permission for the app to access motion/activity data. If the user grants the permission, the query will continue to execute as normal. In the event that the permission was denied, the `OnPermissionDenied` callback will be called and the query will not execute. We can use this callback to show a message to the user explaining why the permission is required.

```
    void Start()
    {
            StepCounterRequest request = new StepCounterRequest();
            request
                    .Since(DateTime.Today)
                    .OnQuerySuccess((stepCount) => Debug.Log("Steps today: " + stepCount))
                    .OnPermissionDenied(() => permissionCanvas.gameObject.SetActive(true))
                    .Execute();
    }
```

You can also use the `RequestPermission` method to manually prompt the user before accessing the step counter as shown below.

```
StepCounterRequest permissionRequest = new StepCounterRequest();
permissionRequest
        .OnPermissionGranted(OnPermissionGranted)
        .OnPermissionDenied(OnPermissionDenied)
        .RequestPermission();
```

If the user has denied the request, you may need to ask them to grant the permissions through the app settings. This can be done using the following method:

```
StepCounterRequest request = new StepCounterRequest();
request.OpenAppSettings();
```

### Step 5: Build the App:

Since the app relies on native plugins, you need to build it to view the step count. After running it for the first time on Android, the step count may appear as 0 or a low number, as data from before the sensor was enabled is not available. On iOS, the step count data from the past 7 days is available immediately.

**For a pre-made example, see the demo scenes within the demo folder**

---

## Setting up the demo

1. Open Demo1 or Demo2
2. If prompted, install TextMeshPro
3. Depending on your input system, you may have to add the default input modules to the `EventManager` GameObject within the scene. Do this for both scenes.

4. **Android:** Set `Minumum Api Level` to `API level 26` in player settings

---

## Important concepts

### Permissions

**When to request permissions**
In order to use the step counter, the user must grant the app permission to access motion/activity data. It is best practice to only ask for permission when you actually need the data. This promotes trust by making it easier for the user to understand how your app intends to use the data.

Executing a valid `StepCounterRequest` query using `Execute()` requests permissions automatically. However step data recorded before the app was installed and the step counter enabled won't be available on Android. If historical step data is important to your application we recommend enabling the step counter early using `Enable()`. It is still important to clearly communicate why your app requires motion data before requesting permissions.

On iOS, the last 7 days of step data are accessible regardless of when the step counter was enabled. Depending on your needs, you may want to avoid querying pre-enablement data to ensure consistency between platforms.

On Android, the user may remove permissions at any time through the app settings. You can handle this event in the `OnPermissionDenied` callback.

**Dealing with denied permissions**
On iOS devices, a permission request is only shown to the user once. If the user denies the request, or removes the permission through the app settings, the request will be automatically denied. Similarly on Android, the user may choose to not be asked again. On some Android versions this happens automatically if the permission is denied more than once.

If step counting is crucial for your app to function, you can ask the user to enable the permission in the app settings. To redirect the user to the app settings simply call the `OpenAppSettings()` method. Always explain to the user why the permission is required before calling `OpenAppSettings()`. Note that on iOS, the app will restart if changes are made to its privacy settings.

**Usage description (iOS)**
The message displayed to the user when requesting permission needs be defined in the `Info.plist` and must explain to the user why the app needs access to motion data. If not already set, `StepCounter Pro` will automatically add a default description during the build phase. The description can be configured in the **config file** within the plugin's folder.

### Enabling

The `Enable()` method is called automatically before executing a query. The first time the enable method is called, the plugin will subscribe to the background collection of step data. On Android, any data collected previous to this will not be available. To start counting steps earlier in your app's life-cycle (or to avoid the additional overhead on the first query) you may call the `Enable()` method manually. Calling `Enable()` requires permissions to be granted.

A possible way of enabling the step counter:

```
void Start() {
    StepCounterRequest request = new StepCounterRequest();
    request
            .OnPermissionGranted(() => request.Enable())
                .OnPermissionDenied(() => Debug.Log("Permission was denied. Cannot count steps"))
            .RequestPermission();
}
```

**Background step recording**
In contrast to the default Unity step counter, `StepCounter Pro` has access to the user's steps even while the app is closed. This also includes steps taken after rebooting the device. The steps are recorded in a battery efficient manner on both iOS and Android.

## Queries

A major strength of `StepCounter Pro` is that it allows you to query the users step data up to 7 days back in time on iOS and up to 10 days on Android.

All queries comprise of a `DateTime startDate` and a `DateTime endDate`.

**DateTime**
The `DateTime` has many handy features to make creating queries easy. Here are some examples

```
DateTime.Today;               // Today at 00:00:00.000
DateTime.Today.AddDays(-1);   // Yesterday at 00:00:00.000
DateTime.Now;                 // Time right now
DateTime.Now.AddMinutes(-5);  // Time 5 minutes ago
```

**Setting time frames for query**
The `From`, `To`, `Between`, and `Since` methods are used to set the start and end dates for the query. Both the start and end dates must be set, so either use `From` in combination with `To` or `Between` to explicitly set the start and end dates, or use the `Since` method to automatically set the end date to `DateTime.Now`.

The example below shows the same exact query, but using the different methods.

```
StepCounterRequest request = new StepCounterRequest();
// The following queries are identical
request.From(DateTime.Today).To(DateTime.Now).OnQuerySuccess(...).Execute();
request.Between(DateTime.Today, DateTime.Now).OnQuerySuccess(...).Execute();
request.Since(DateTime.Today).OnQuerySuccess(...).Execute();
```

---

# StepCounterRequest Class Documentation

The `StepCounterRequest` class represents a request to query step count data and to perform other step counter related actions from the platform specific step counter bridge.

## Constructors

### `StepCounterRequest()`

Initializes a new instance of the `StepCounterRequest` class.

## Methods

### `From(DateTime fromDate)`

Sets the start date for the step count query.

- **Parameters:**
  - `fromDate` (DateTime): The start date of the query period.
- **Returns:** The current `StepCounterRequest` instance.

### `To(DateTime toDate)`

Sets the end date for the step count query.

- **Parameters:**
  - `toDate` (DateTime): The end date of the query period.
- **Returns:** The current `StepCounterRequest` instance.

### `Between(DateTime fromDate, DateTime toDate)`

Sets the start and end dates for the step count query.

- **Parameters:**
  - `fromDate` (DateTime): The start date of the query period.
  - `toDate` (DateTime): The end date of the query period.
- **Returns:** The current `StepCounterRequest` instance.

### `Since(DateTime fromDate)`

Sets the start date and uses the current date as the end date for the step count query.

- **Parameters:**
  - `fromDate` (DateTime): The start date of the query period.
- **Returns:** The current `StepCounterRequest` instance.

### `OnQuerySuccess(Action<int> onQuerySuccess)`

Sets the callback to be invoked when the step count query is successful.

- **Parameters:**
  - `onQuerySuccess` (Action<int>): The callback action that receives the step count as an integer
- Returns: The current `StepCounterRequest` instance..

### `OnError(Action<string> onError)`

Sets the callback to be invoked when an error occurs during the step count query.

- **Parameters:**
  - `onError` (Action<string>): The callback action that receives the error message as a string.
- **Returns:** The current `StepCounterRequest` instance.

### `OnPermissionDenied(Action onPermissionDenied)`

Sets the callback to be invoked when the user denies the required permission for step counting.

- **Parameters:**
  - `onPermissionDenied` (Action): The callback action to be invoked when permission is denied.
- **Returns:** The current `StepCounterRequest` instance.

### `OnPermissionGranted(Action onPermissionGranted)`

Sets the callback to be invoked when the user grants the required permission for step counting.

- **Parameters:**
  - `onPermissionGranted` (Action): The callback action to be invoked when permission is granted.
- **Returns:** The current `StepCounterRequest` instance.

### `Execute()`

Executes the step count query with the provided parameters and callbacks.

### `OpenAppSettings()`

Opens the app settings to allow the user to manually grant permissions.

### `HasPermission()`

Checks whether the necessary permission for step counting has been granted.

- **Returns:** `true` if the permission is granted; otherwise, `false`.

### `IsEnabled()`

Checks whether the step counter functionality is enabled on the device.

- **Returns:** `true` if the step counter is enabled; otherwise, `false`.

### `isStepCounterAvailible()`

Checks whether the step counter feature is available on the current device.

- **Returns:** `true` if the step counter is available; otherwise, `false`.

### `RequestPermission()`

Requests the necessary permissions to use the step counter.

- **Remarks:**
  - If permission is already granted, the `onPermissionGranted` callback will be invoked immediately without prompting the user.
  - If permission was previously denied on iOS or the user selected "Do not ask again" on Android, the `onPermissionDenied` callback will be invoked without prompting the user.

- If the step counter is crucial for your application, it is recommended to guide the user to the app's settings using `OpenAppSettings()`.