android / **platform_frameworks_base**
mirrored from https://android.googlesource.com/platform/frameworks/base.git

Branch: master **platform_frameworks_base** / core / java / com / android / internal / os / **BatterySipper.java** Find file Copy path

Adam Lesinski Track BLE scans in BatteryStats 9f55cc7 on 2 Feb 2016

1 contributor

183 lines (167 sloc) 5.74 KB Raw Blame History

```
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * 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.
 */
package com.android.internal.os;

import android.os.BatteryStats.Uid;

/**
 * Contains power usage of an application, system service, or hardware type.
 */
public class BatterySipper implements Comparable<BatterySipper> {
    public int userId;
    public Uid uidObj;
    public double totalPowerMah;
    public DrainType drainType;

    /**
     * Generic usage time in milliseconds.
     */
    public long usageTimeMs;

    /**
     * Generic power usage in mAh.
     */
    public double usagePowerMah;

    // Subsystem usage times.
    public long cpuTimeMs;
    public long gpsTimeMs;
    public long wifiRunningTimeMs;
    public long cpuFgTimeMs;
    public long wakeLockTimeMs;
    public long cameraTimeMs;
    public long flashlightTimeMs;
    public long bluetoothRunningTimeMs;

    public long mobileRxPackets;
    public long mobileTxPackets;
    public long mobileActive;
    public int mobileActiveCount;
    public double mobilemspp;         // milliseconds per packet
    public long wifiRxPackets;
    public long wifiTxPackets;
    public long mobileRxBytes;
    public long mobileTxBytes;
```

```
58      public long wifiRxBytes;
59      public long wifiTxBytes;
60      public long btRxBytes;
61      public long btTxBytes;
62      public double percent;
63      public double noCoveragePercent;
64      public String[] mPackages;
65      public String packageWithHighestDrain;
66
67      // Measured in mAh (milli-ampere per hour).
68      // These are included when summed.
69      public double wifiPowerMah;
70      public double cpuPowerMah;
71      public double wakeLockPowerMah;
72      public double mobileRadioPowerMah;
73      public double gpsPowerMah;
74      public double sensorPowerMah;
75      public double cameraPowerMah;
76      public double flashlightPowerMah;
77      public double bluetoothPowerMah;
78
79      public enum DrainType {
80          IDLE,
81          CELL,
82          PHONE,
83          WIFI,
84          BLUETOOTH,
85          FLASHLIGHT,
86          SCREEN,
87          APP,
88          USER,
89          UNACCOUNTED,
90          OVERCOUNTED,
91          CAMERA
92      }
93
94      public BatterySipper(DrainType drainType, Uid uid, double value) {
95          this.totalPowerMah = value;
96          this.drainType = drainType;
97          uidObj = uid;
98      }
99
100     public void computeMobilemspp() {
101         long packets = mobileRxPackets+mobileTxPackets;
102         mobilemspp = packets > 0 ? (mobileActive / (double)packets) : 0;
103     }
104
105     @Override
106     public int compareTo(BatterySipper other) {
107         // Over-counted always goes to the bottom.
108         if (drainType != other.drainType) {
109             if (drainType == DrainType.OVERCOUNTED) {
110                 // This is "larger"
111                 return 1;
112             } else if (other.drainType == DrainType.OVERCOUNTED) {
113                 return -1;
114             }
115         }
116         // Return the flipped value because we want the items in descending order
117         return Double.compare(other.totalPowerMah, totalPowerMah);
118     }
119
120     /**
121      * Gets a list of packages associated with the current user
122      */
123     public String[] getPackages() {
124         return mPackages;
125     }
126
127     public int getUid() {
128         // Bail out if the current sipper is not an App sipper.
129         if (uidObj == null) {
130             return 0;
131         }
132         return uidObj.getUid();
```

```
133     }
134
135     /**
136      * Add stats from other to this BatterySipper.
137      */
138     public void add(BatterySipper other) {
139         totalPowerMah += other.totalPowerMah;
140         usageTimeMs += other.usageTimeMs;
141         usagePowerMah += other.usagePowerMah;
142         cpuTimeMs += other.cpuTimeMs;
143         gpsTimeMs += other.gpsTimeMs;
144         wifiRunningTimeMs += other.wifiRunningTimeMs;
145         cpuFgTimeMs += other.cpuFgTimeMs;
146         wakeLockTimeMs += other.wakeLockTimeMs;
147         cameraTimeMs += other.cameraTimeMs;
148         flashlightTimeMs += other.flashlightTimeMs;
149         bluetoothRunningTimeMs += other.bluetoothRunningTimeMs;
150         mobileRxPackets += other.mobileRxPackets;
151         mobileTxPackets += other.mobileTxPackets;
152         mobileActive += other.mobileActive;
153         mobileActiveCount += other.mobileActiveCount;
154         wifiRxPackets += other.wifiRxPackets;
155         wifiTxPackets += other.wifiTxPackets;
156         mobileRxBytes += other.mobileRxBytes;
157         mobileTxBytes += other.mobileTxBytes;
158         wifiRxBytes += other.wifiRxBytes;
159         wifiTxBytes += other.wifiTxBytes;
160         btRxBytes += other.btRxBytes;
161         btTxBytes += other.btTxBytes;
162         wifiPowerMah += other.wifiPowerMah;
163         gpsPowerMah += other.gpsPowerMah;
164         cpuPowerMah += other.cpuPowerMah;
165         sensorPowerMah += other.sensorPowerMah;
166         mobileRadioPowerMah += other.mobileRadioPowerMah;
167         wakeLockPowerMah += other.wakeLockPowerMah;
168         cameraPowerMah += other.cameraPowerMah;
169         flashlightPowerMah += other.flashlightPowerMah;
170         bluetoothPowerMah += other.bluetoothPowerMah;
171     }
172
173     /**
174      * Sum all the powers and store the value into `value`.
175      * @return the sum of all the power in this BatterySipper.
176      */
177     public double sumPower() {
178         return totalPowerMah = usagePowerMah + wifiPowerMah + gpsPowerMah + cpuPowerMah +
179                 sensorPowerMah + mobileRadioPowerMah + wakeLockPowerMah + cameraPowerMah +
180                 flashlightPowerMah + bluetoothPowerMah;
181     }
182 }
```