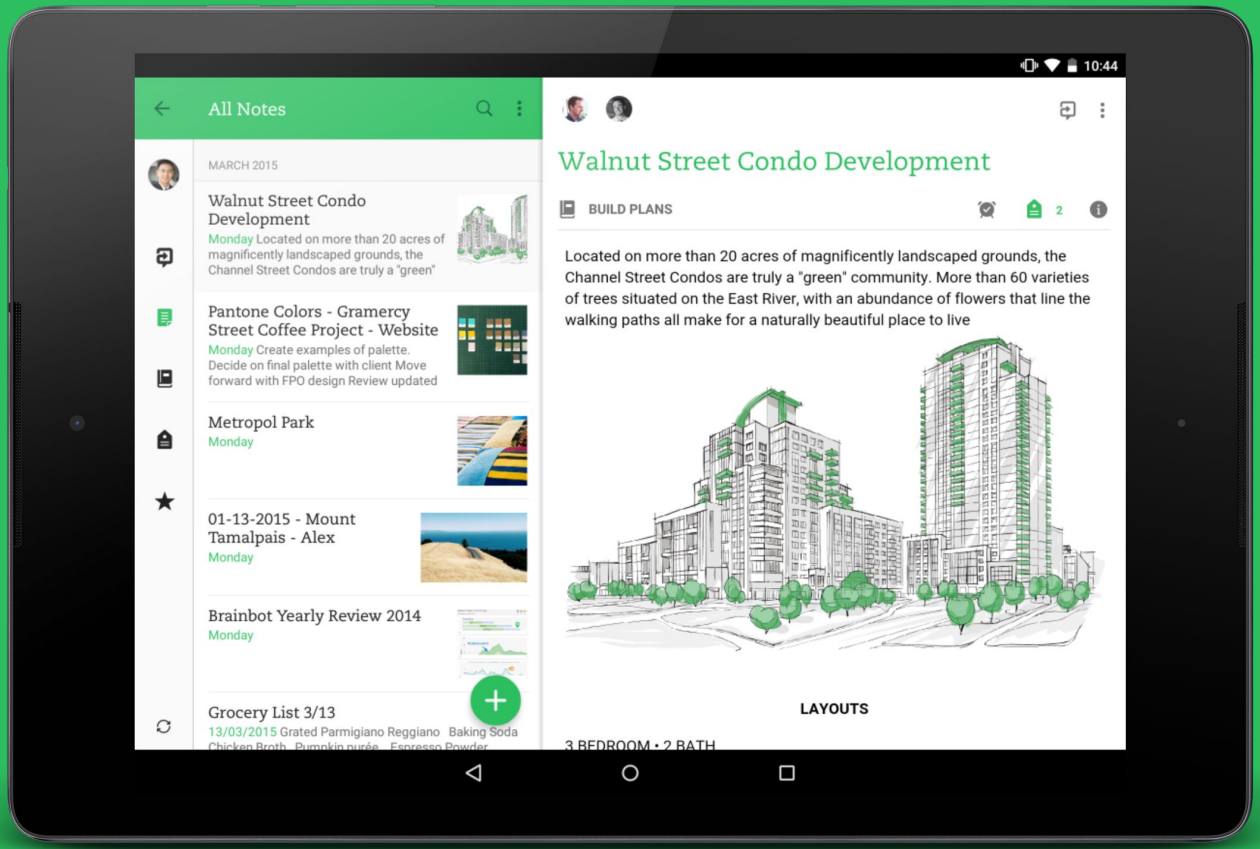
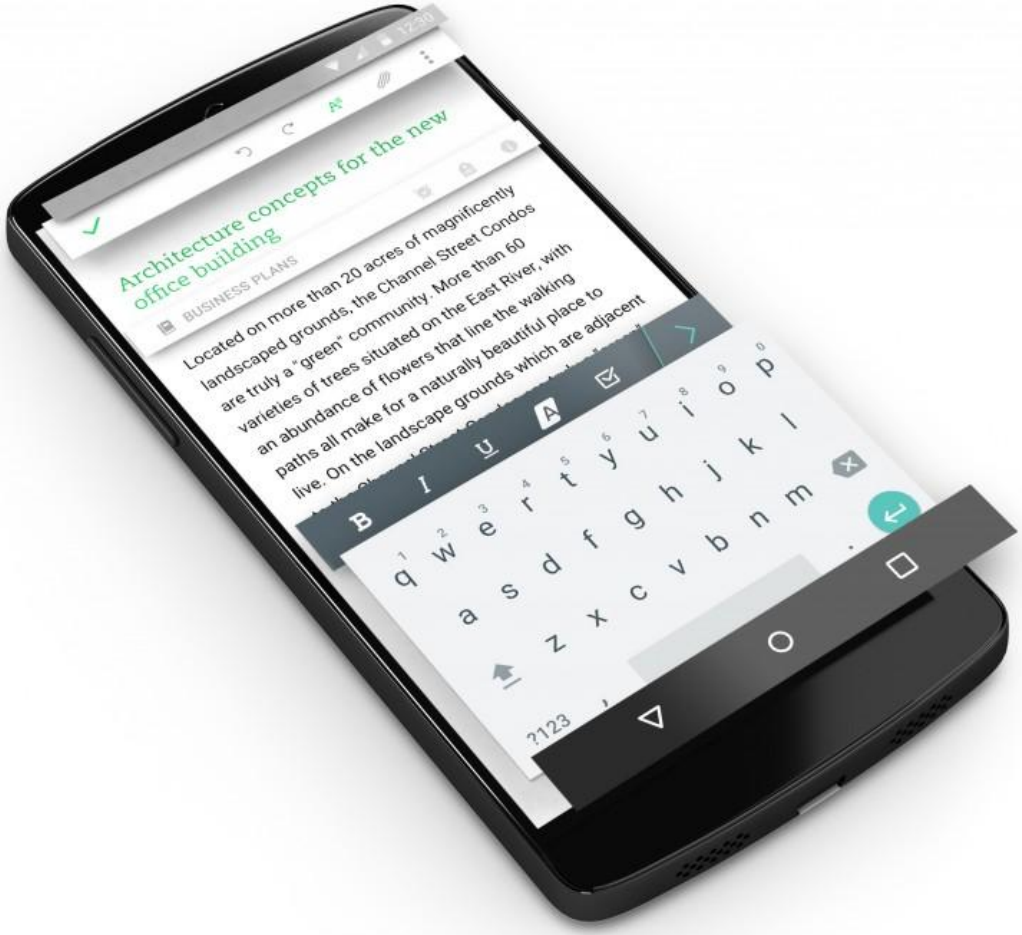
Schedule background jobs at the right time

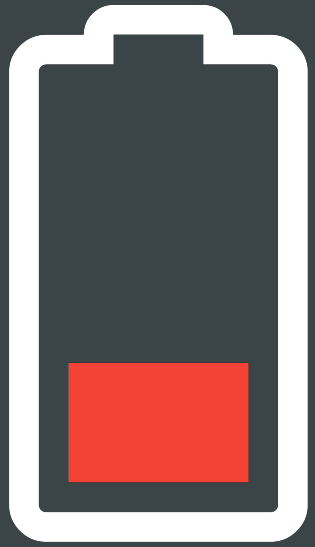
Ralf Wondratschek
2017-03-14

EVERNOTE


10:44
All Notes
MARCH 2015
Walnut Street Condo Development
Monday Located on more than 20 acres of magnificently landscaped grounds, the Channel Street Condos are truly a "green"
Pantone Colors - Gramercy Street Coffee Project - Website
Monday Create examples of palette. Decide on final palette with client Move forward with FPO design Review updated
Metropol Park
Monday
01-13-2015 - Mount Tamalpais - Alex
Monday
Brainbot Yearly Review 2014
Monday
Grocery List 3/13
13/03/2015 Grated Parmigiano Reggiano Baking Soda
Chicken Broth Pumpkin purée Espresso Powder
Walnut Street Condo Development
BUILD PLANS
2
Located on more than 20 acres of magnificently landscaped grounds, the Channel Street Condos are truly a "green" community. More than 60 varieties of trees situated on the East River, with an abundance of flowers that line the walking paths all make for a naturally beautiful place to live
LAYOUTS
3 BEDROOM • 2 BATH


Architecture concepts for the new office building
BUSINESS PLANS
Located on more than 20 acres of magnificently landscaped grounds, the Channel Street Condos are truly a "green" community. More than 60 varieties of trees situated on the East River, with an abundance of flowers that line the walking paths all make for a naturally beautiful place to live. On the landscape grounds which are adjacent

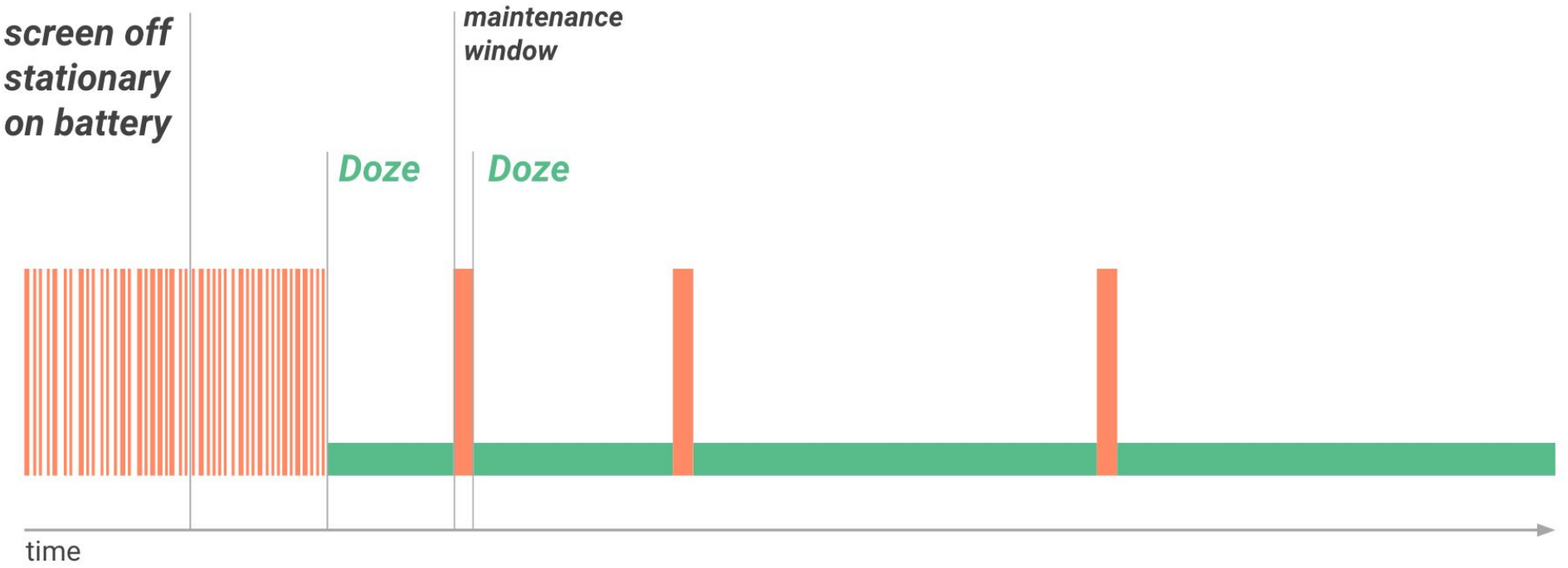
Motivation



Motivation

- Every app has some kind of repeated work
- Don’t waste battery unnecessarily
- App should still work correctly even in Doze mode or in App Standby
- Scheduling repeated work is quite a headache with 3 different APIs all doing similar things

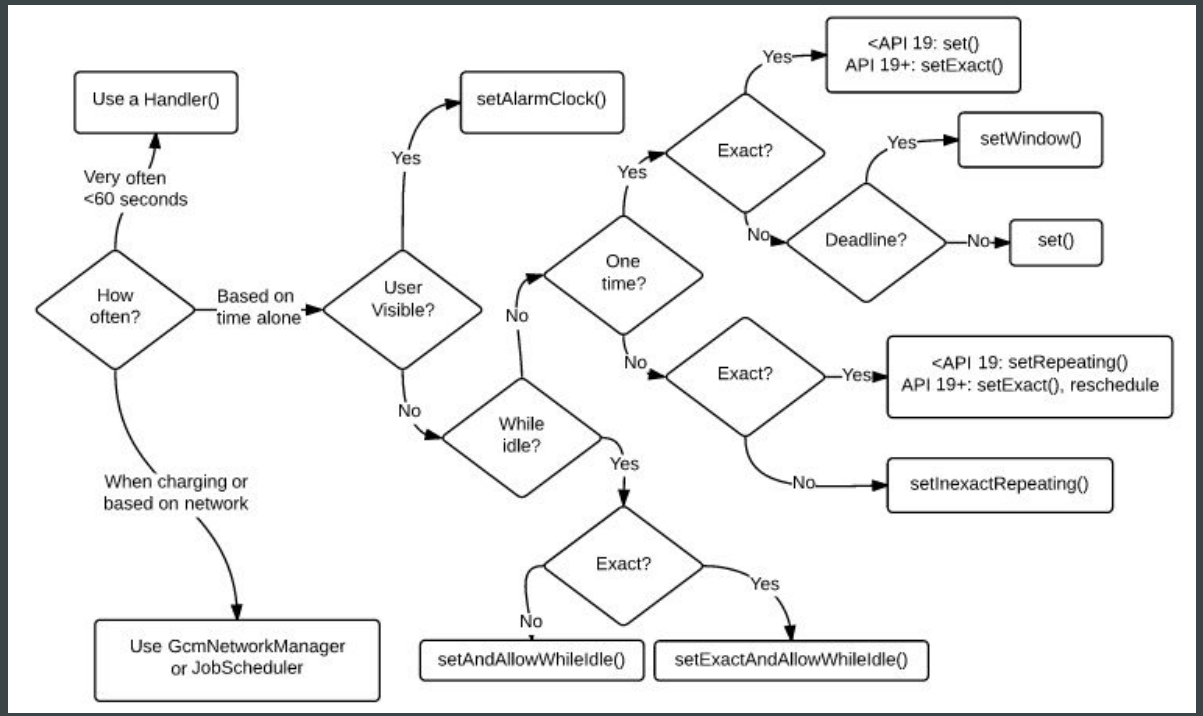
Doze & App Standby



https://developer.android.com/images/training/doze.png

All APIs in a nutshell

AlarmManager



https://plus.google.com/+AndroidDevelopers/posts/GdNrQciPwqo

AlarmManager

+ Available on all devices
+ Easy to send broadcast to start a service delayed

- API behavior differs between platform versions
- A lot of boilerplate
- Device state ignored
- …

JobScheduler

+ Easy to use with fluent API
+ Respects device state

- Only on API 21+ (some features only API 24+)
- Platform bugs
- Still a lot of boilerplate code

GCM Network Manager

+ Similar API like JobScheduler
+ minSdkVersion 9

- Part of Google’s Play Services SDK
- Can’t be used without Play Services being pre-installed (most Chinese devices)
- API contains some gotchas

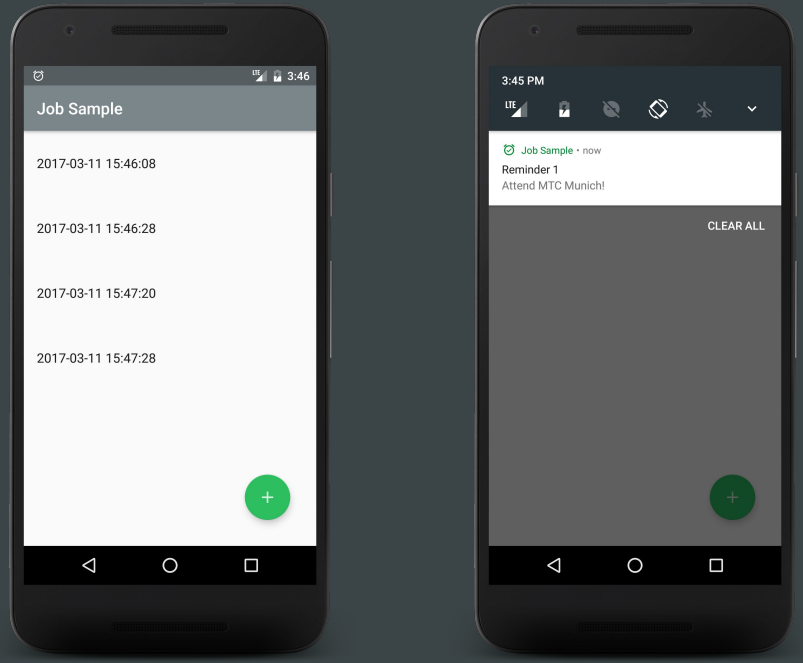
(Firebase JobDispatcher)

+ Wrapper for job scheduling engines

- Setup is difficult
- Library not maintained
- Only GCM NetworkManager supported
- Many open issues

→ I would not recommend using it

Demo - Reminder app



https://github.com/vRallev/job-sample

Requirements

- Show the reminder in a notification
- Sync reminders with the server
- Support all devices
- Be a good citizen

Reminder job

- Must be exact
- Can have multiple at the same time
- AlarmManager is the correct API

Reminder job

```
<receiver
    android:name=".reminder.ReminderReceiver"
    android:exported="false"/>
```

```
public class ReminderReceiver extends BroadcastReceiver {

    private static final String EXTRA_ID = "EXTRA_ID";

    @Override
    public void onReceive(Context context, Intent intent) {
        int id = intent.getIntExtra(EXTRA_ID, -1);
        if (id < 0) {
            return;
        }

        Reminder reminder = ReminderEngine.instance().getReminderById(id);
        if (reminder != null) {
            ReminderEngine.instance().showReminder(reminder);
        }
    }
}
```

API 14-18

```
public static void schedule(Context context, Reminder reminder) {
    Intent intent = new Intent(context, ReminderReceiver.class);
    intent.putExtra(EXTRA_ID, reminder.getId());

    int requestCode = reminder.getId();
    int flags = PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_ONE_SHOT;

    PendingIntent pendingIntent = PendingIntent
            .getBroadcast(context, requestCode, intent, flags);

    AlarmManager manager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
    manager.set(AlarmManager.RTC_WAKEUP, reminder.getTimestamp(), pendingIntent);
}
```

API 14-18

```
public static void schedule(Context context, Reminder reminder) {
    Intent intent = new Intent(context, ReminderReceiver.class);
    intent.putExtra(EXTRA_ID, reminder.getId());

    int requestCode = reminder.getId();
    int flags = PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_ONE_SHOT;

    PendingIntent pendingIntent = PendingIntent
            .getBroadcast(context, requestCode, intent, flags);

    AlarmManager manager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
    manager.set(AlarmManager.RTC_WAKEUP, reminder.getTimestamp(), pendingIntent);
}
```

API 19-22

```
public static void schedule(Context context, Reminder reminder) {
    Intent intent = new Intent(context, ReminderReceiver.class);
    intent.putExtra(EXTRA_ID, reminder.getId());

    int requestCode = reminder.getId();
    int flags = PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_ONE_SHOT;

    PendingIntent pendingIntent = PendingIntent
            .getBroadcast(context, requestCode, intent, flags);

    AlarmManager manager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
    manager.set(AlarmManager.RTC_WAKEUP, reminder.getTimestamp(), pendingIntent);
}
```

API 19-22

```
public static void schedule(Context context, Reminder reminder) {
    Intent intent = new Intent(context, ReminderReceiver.class);
    intent.putExtra(EXTRA_ID, reminder.getId());

    int requestCode = reminder.getId();
    int flags = PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_ONE_SHOT;

    PendingIntent pendingIntent = PendingIntent
            .getBroadcast(context, requestCode, intent, flags);

    AlarmManager manager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
    manager.setExact(AlarmManager.RTC_WAKEUP, reminder.getTimestamp(), pendingIntent);
}
```

API 23-25

```
public static void schedule(Context context, Reminder reminder) {
    Intent intent = new Intent(context, ReminderReceiver.class);
    intent.putExtra(EXTRA_ID, reminder.getId());

    int requestCode = reminder.getId();
    int flags = PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_ONE_SHOT;

    PendingIntent pendingIntent = PendingIntent
            .getBroadcast(context, requestCode, intent, flags);

    AlarmManager manager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
    manager.setExact(AlarmManager.RTC_WAKEUP, reminder.getTimestamp(), pendingIntent);
}
```

API 23-25

```
public static void schedule(Context context, Reminder reminder) {
    Intent intent = new Intent(context, ReminderReceiver.class);
    intent.putExtra(EXTRA_ID, reminder.getId());

    int requestCode = reminder.getId();
    int flags = PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_ONE_SHOT;

    PendingIntent pendingIntent = PendingIntent
            .getBroadcast(context, requestCode, intent, flags);

    AlarmManager manager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
    manager.setExactAndAllowWhileIdle(AlarmManager.RTC_WAKEUP,
                reminder.getTimestamp(), pendingIntent);
}
```

Reminder job

```
public static void schedule(Context context, Reminder reminder) {
    // …
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        manager.setExactAndAllowWhileIdle(AlarmManager.RTC_WAKEUP, when, pendingIntent);

    } else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
        manager.setExact(AlarmManager.RTC_WAKEUP, when, pendingIntent);

    } else {
        manager.set(AlarmManager.RTC_WAKEUP, when, pendingIntent);
    }
}
```

Sync job

- Can be inexact, but must be repeating
- Run job only on an unmetered network and only if device is charging
- Don’t wake up the device to sync data
- JobScheduler and GCM Network Manager work best, AlarmManager is fallback

API 21-23

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>

<application … >
   <service
       android:name=".sync.SyncJob"
       android:permission="android.permission.BIND_JOB_SERVICE"/>
</application>
```

API 21-23

```
public class SyncJob extends JobService {

    @Override
    public boolean onStartJob(JobParameters params) {
        // called on main thread
        new Thread(() -> {
            try {
                new SyncEngine().syncReminders();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                jobFinished(params, false); // don't forget to call
            }
        }).start();

        return true; // we have background work
    }

    @Override
    public boolean onStopJob(JobParameters params) {
        return false; // don't reschedule
    }
}
```

API 21-23

```
private static final int JOB_ID = 1;

public static void schedule(Context context) {
   long interval = TimeUnit.HOURS.toMillis(6);

   JobInfo jobInfo = new JobInfo.Builder(JOB_ID, new ComponentName(context, SyncJob.class))
           .setRequiresCharging(true)
           .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
           .setPersisted(true)
           .setPeriodic(interval)
           .build();

   JobScheduler jobScheduler = (JobScheduler) context
                .getSystemService(Context.JOB_SCHEDULER_SERVICE);
   jobScheduler.schedule(jobInfo);
}
```

API 21-23

```
private static final int JOB_ID = 1;

public static void schedule(Context context) {
   long interval = TimeUnit.HOURS.toMillis(6);

   JobInfo jobInfo = new JobInfo.Builder(JOB_ID, new ComponentName(context, SyncJob.class))
           .setRequiresCharging(true)
           .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
           .setPersisted(true)
           .setPeriodic(interval)
           .build();

   JobScheduler jobScheduler = (JobScheduler) context
                   .getSystemService(Context.JOB_SCHEDULER_SERVICE);
   jobScheduler.schedule(jobInfo);
}
```

API 24-25

```
private static final int JOB_ID = 1;

public static void schedule(Context context) {
   long interval = TimeUnit.HOURS.toMillis(6);
   long flex = TimeUnit.HOURS.toMillis(3);

   JobInfo jobInfo = new JobInfo.Builder(JOB_ID, new ComponentName(context, SyncJob.class))
           .setRequiresCharging(true)
           .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
           .setPersisted(true)
           .setPeriodic(interval, flex)
           .build();

   JobScheduler jobScheduler = (JobScheduler) context
           .getSystemService(Context.JOB_SCHEDULER_SERVICE);
   jobScheduler.schedule(jobInfo);
}
```

Flex parameter explained

Flex parameter explained

With Google Play Services

```
<service
   android:name=".sync.SyncJob"
   android:exported="true"
   android:permission="com.google.android.gms.permission.BIND_NETWORK_TASK_SERVICE">
   <intent-filter>
       <action android:name="com.google.android.gms.gcm.ACTION_TASK_READY"/>
   </intent-filter>
</service>
```

With Google Play Services

```
public class SyncJob extends GcmTaskService {

    @Override
    public int onRunTask(TaskParams taskParams) {
        try {
            new SyncEngine().syncReminders();
            return GcmNetworkManager.RESULT_SUCCESS;

        } catch (IOException e) {
            e.printStackTrace();
            return GcmNetworkManager.RESULT_FAILURE;
        }
    }
}
```

With Google Play Services

```
private static final String TAG = "SyncJob";

public void schedule(Context context) {

   long interval = TimeUnit.HOURS.toMillis(6);
   long flex = TimeUnit.HOURS.toMillis(3);

   PeriodicTask task = new PeriodicTask.Builder()
           .setTag(TAG)
           .setService(SyncJob.class)
           .setRequiresCharging(true)
           .setRequiredNetwork(Task.NETWORK_STATE_UNMETERED)
           .setPersisted(true)
           .setUpdateCurrent(true)
           .setPeriod(interval / 1_000)
           .setFlex(flex / 1_000)
           .build();

   GcmNetworkManager.getInstance(context).schedule(task);
}
```

With Google Play Services

```
private static final String TAG = "SyncJob";

public void schedule(Context context) {

    long interval = TimeUnit.HOURS.toMillis(6);
    long flex = TimeUnit.HOURS.toMillis(3);

    PeriodicTask task = new PeriodicTask.Builder()
            .setTag(TAG)
            .setService(SyncJob.class)
            .setRequiresCharging(true)
            .setRequiredNetwork(Task.NETWORK_STATE_UNMETERED)
            .setPersisted(true)
            .setUpdateCurrent(true)
            .setPeriod(interval / 1_000)
            .setFlex(flex / 1_000)
            .build();

    GcmNetworkManager.getInstance(context).schedule(task);
}
```

API 14-19

- Use AlarmManager
- As much fun as the reminder job…

Sync job

```
public void schedule(Context context) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
        scheduleWithJobScheduler24(context);

    } else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        scheduleWithJobScheduler21(context);

    } else if (GoogleApiAvailability.getInstance().isGooglePlayServicesAvailable(context)
            == ConnectionResult.SUCCESS) {
        scheduleWithGcmNetworkManager(context);

    } else {
        scheduleWithAlarmManager(context);
    }
}
```

Conclusion

- A lot of boilerplate to achieve something that should be simple

~~Conclusion~~

- A lot of boilerplate to achieve something that should be simple
- It should be much easier

Fork me on GitHub

github.com/evernote/android-job

android-job

- Single API to use the JobSchduler, GCM Network Manager and AlarmManager
- All API 24 features supported
- Automatically chooses best API to run a job
- Less boilerplate, e.g. no manifest entry needed
- No reflection used
- Maintained and improved continuously
- … and more

API

```
private void scheduleAdvancedJob() {
   PersistableBundleCompat extras = new PersistableBundleCompat();
   extras.putString("key", "Hello world");

   int jobId = new JobRequest.Builder(DemoSyncJob.TAG)
           .setExecutionWindow(30_000L, 40_000L)
           .setExact(30_000L)
           .setPeriodic(TimeUnit.HOURS.toMillis(3))
           .setBackoffCriteria(5_000L, JobRequest.BackoffPolicy.EXPONENTIAL)
           .setRequiresCharging(true)
           .setRequiresDeviceIdle(false)
           .setRequiredNetworkType(JobRequest.NetworkType.CONNECTED)
           .setRequirementsEnforced(true)
           .setExtras(extras)
           .setPersisted(true)
           .setUpdateCurrent(true)
           .build()
           .schedule();
}
```

Setup

```
dependencies {
    compile 'com.evernote:android-job:1.1.7'
}
```

```
public class App extends Application {

    @Override
    public void onCreate() {
        super.onCreate();

        JobManager.create(this).addJobCreator(new ReminderJobCreator());
    }
}
```

Setup

```
public class ReminderJobCreator implements JobCreator {

    @Override
    public Job create(String tag) {
        switch (tag) {
            case ReminderJob.TAG:
                return new ReminderJob();

            case SyncJob.TAG:
                return new SyncJob();

            default:
                return null;
        }
    }
}
```

Reminder job

```
public class ReminderJob extends Job {

    public static final String TAG = "ReminderJob";
    private static final String EXTRA_ID = "EXTRA_ID";

    @NonNull
    @Override
    protected Result onRunJob(Params params) {
        int id = params.getExtras().getInt(EXTRA_ID, -1);
        Reminder reminder = ReminderEngine.instance().getReminderById(id);
        if (reminder == null) {
            return Result.FAILURE;
        }
        ReminderEngine.instance().showReminder(reminder);
        return Result.SUCCESS;
    }
}
```

Reminder job

```
public static int schedule(@NonNull Reminder reminder) {
   PersistableBundleCompat extras = new PersistableBundleCompat();
   extras.putInt(EXTRA_ID, reminder.getId());

   long time = Math.max(1L, reminder.getTimestamp() - System.currentTimeMillis());

   return new JobRequest.Builder(TAG)
           .setExact(time)
           .setExtras(extras)
           .setPersisted(true)
           .setUpdateCurrent(false)
           .build()
           .schedule();
}
```

Reminder job

```
public static int schedule(@NonNull Reminder reminder) {
    PersistableBundleCompat extras = new PersistableBundleCompat();
    extras.putInt(EXTRA_ID, reminder.getId());

    long time = Math.max(1L, reminder.getTimestamp() - System.currentTimeMillis());

    return new JobRequest.Builder(TAG)
            .setExact(time)
            .setExtras(extras)
            .setPersisted(true)
            .setUpdateCurrent(false)
            .build()
            .schedule();
}
```

Sync job

```
public class SyncJob extends Job {

    public static final String TAG = "SyncJob";

    @NonNull
    @Override
    protected Result onRunJob(Params params) {
        new SyncEngine().syncReminders();
        return Result.SUCCESS;
    }
}
```

Sync job

```
public static int schedule() {
    Set<JobRequest> jobRequests = JobManager.instance().getAllJobRequestsForTag(TAG);
    if (!jobRequests.isEmpty()) {
        return jobRequests.iterator().next().getJobId();
    }

    long interval = TimeUnit.HOURS.toMillis(6); // every 6 hours
    long flex = TimeUnit.HOURS.toMillis(3); // wait 3 hours before job runs again

    return new JobRequest.Builder(TAG)
            .setPeriodic(interval, flex)
            .setPersisted(true)
            .setUpdateCurrent(true)
            .setRequiredNetworkType(JobRequest.NetworkType.UNMETERED)
            .setRequiresCharging(true)
            .setRequirementsEnforced(true)
            .build()
            .schedule();
}
```

Sync job

```
public static int schedule() {
    Set<JobRequest> jobRequests = JobManager.instance().getAllJobRequestsForTag(TAG);
    if (!jobRequests.isEmpty()) {
        return jobRequests.iterator().next().getJobId();
    }

    long interval = TimeUnit.HOURS.toMillis(6); // every 6 hours
    long flex = TimeUnit.HOURS.toMillis(3); // wait 3 hours before job runs again

    return new JobRequest.Builder(TAG)
            .setPeriodic(interval, flex)
            .setPersisted(true)
            .setUpdateCurrent(true)
            .setRequiredNetworkType(JobRequest.NetworkType.UNMETERED)
            .setRequiresCharging(true)
            .setRequirementsEnforced(true)
            .build()
            .schedule();
}
```

Sync job

```
public static int schedule() {
    Set<JobRequest> jobRequests = JobManager.instance().getAllJobRequestsForTag(TAG);
    if (!jobRequests.isEmpty()) {
        return jobRequests.iterator().next().getJobId();
    }

    long interval = TimeUnit.HOURS.toMillis(6); // every 6 hours
    long flex = TimeUnit.HOURS.toMillis(3); // wait 3 hours before job runs again

    return new JobRequest.Builder(TAG)
            .setPeriodic(interval, flex)
            .setPersisted(true)
            .setUpdateCurrent(true)
            .setRequiredNetworkType(JobRequest.NetworkType.UNMETERED)
            .setRequiresCharging(true)
            .setRequirementsEnforced(true)
            .build()
            .schedule();
}
```

Sync job

```
public static int schedule() {
    Set<JobRequest> jobRequests = JobManager.instance().getAllJobRequestsForTag(TAG);
    if (!jobRequests.isEmpty()) {
        return jobRequests.iterator().next().getJobId();
    }

    long interval = TimeUnit.HOURS.toMillis(6); // every 6 hours
    long flex = TimeUnit.HOURS.toMillis(3); // wait 3 hours before job runs again

    return new JobRequest.Builder(TAG)
            .setPeriodic(interval, flex)
            .setPersisted(true)
            .setUpdateCurrent(true)
            .setRequiredNetworkType(JobRequest.NetworkType.UNMETERED)
            .setRequiresCharging(true)
            .setRequirementsEnforced(true)
            .build()
            .schedule();
}
```

Sync job

```
public static int schedule() {
    Set<JobRequest> jobRequests = JobManager.instance().getAllJobRequestsForTag(TAG);
    if (!jobRequests.isEmpty()) {
        return jobRequests.iterator().next().getJobId();
    }

    long interval = TimeUnit.HOURS.toMillis(6); // every 6 hours
    long flex = TimeUnit.HOURS.toMillis(3); // wait 3 hours before job runs again

    return new JobRequest.Builder(TAG)
            .setPeriodic(interval, flex)
            .setPersisted(true)
            .setUpdateCurrent(true)
            .setRequiredNetworkType(JobRequest.NetworkType.UNMETERED)
            .setRequiresCharging(true)
            .setRequirementsEnforced(true)
            .build()
            .schedule();
}
```

Who is using it?

FAQ

Which API is used?

- Exact job → AlarmManager
- API 21+ → JobScheduler
- Play Services installed → GcmNetworkManager
- Else → AlarmManager

FAQ

What happens after a reboot?

- All jobs are rescheduled if necessary

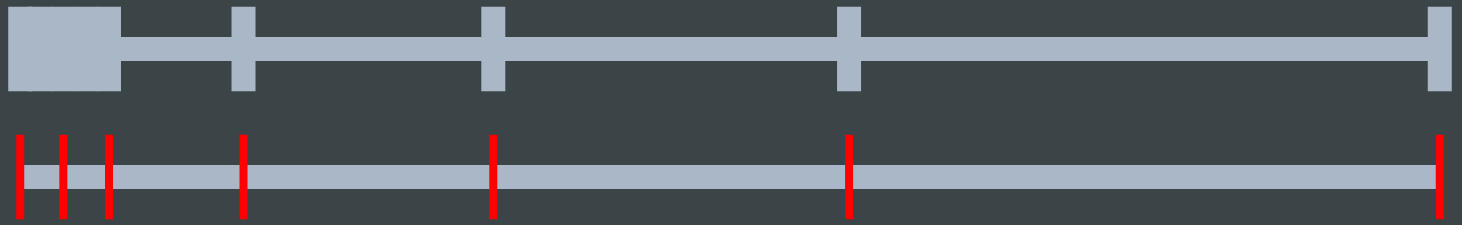
  → even after a force close

  → even after the Play Services were updated

FAQ

My periodic jobs aren't running as expected.

- Keep Doze and the system in mind



android-job

Tips & Tricks

- Look at the samples in the README
- Read the FAQ
- Uncertain? Having problems?
  → Create an issue

Conclusion

- Android APIs force you to
  - do the same thing multiple times
  - write a lot of boilerplate
  - catch many gotchas
- android-job to the rescue

Schedule background jobs at the right time

google.com/+vRallev
twitter.com/vRallev