Discovering the

Android API - Part 1

03 JANUARY 2017 on Android, Tutorial

We are excited to kick off our *Discovering the Android API* series. The Android team at *AutSoft* is always following the cutting edge technologies and is participating in teaching Android development for more than 8 years at *Budapest University of Technology and Economics*. We have a tradition at our company to share our knowledge every week with each other, and now we start this post series where we share different interesting APIs that are part of the Android platform.

Nowadays there are a lot of libraries that we use on the Android platform, but sometimes it is really worth to check the capability of the existing Android API that has a several, not well known, but very useful classes and methods.
It is also very important to know that the Android API is extending the Java SE API which is already very huge, according to statistics, the *Java SE 8* has 217 packages with 4240 classes, while *Java SE 7* has 209 packages with 4024 classes.

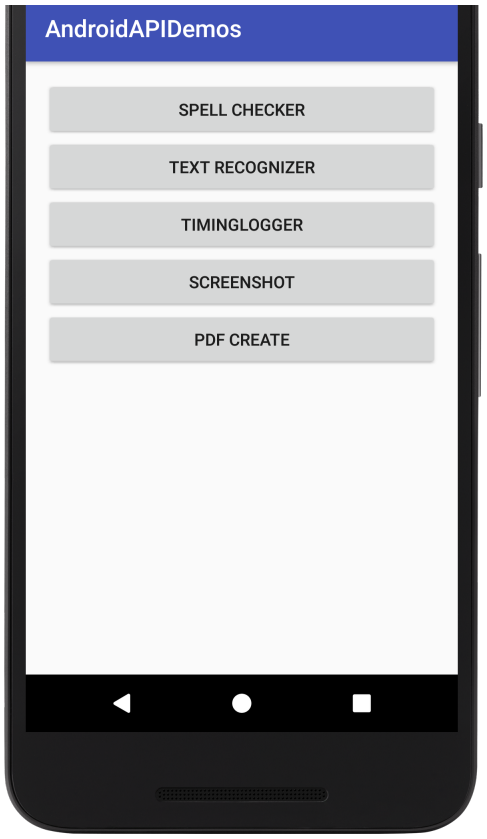
Java Development Kits	Codename	Release	Packages	Classes
Java SE 8 with JDK 1.8.0	---	2014	217	4,240
Java SE 7 with JDK 1.7.0	Dolphin	2011	209	4,024

Java SE 6 with JDK 1.6.0	Mustang	2006	203	3,793
Java 2 SE 5.0 with JDK 1.5.0	Tiger	2004	166	3,279
Java 2 SE with SDK 1.4.0	Merlin	2002	135	2,991
Java 2 SE with SDK 1.3	Kestrel	2000	76	1,842
Java 2 with SDK 1.2	Playground	1998	59	1,520
Development Kit 1.1	---	1997	23	504
Development Kit 1.0	Oak	1996	8	212

source: Java 8 Pocket Guide book by Robert Liguori, Patricia Liguor

In this series, our goal is to discover the not widely known APIs of the Android platform from many different perspectives. We show examples and use cases for the APIs and we also publish the demo under the following repository:
https://github.com/peekler/GDG

The demo application contains always each feature/API demo on a separate *Activity* that can be accessed from the main screen:



No. 1 - Spell Checker

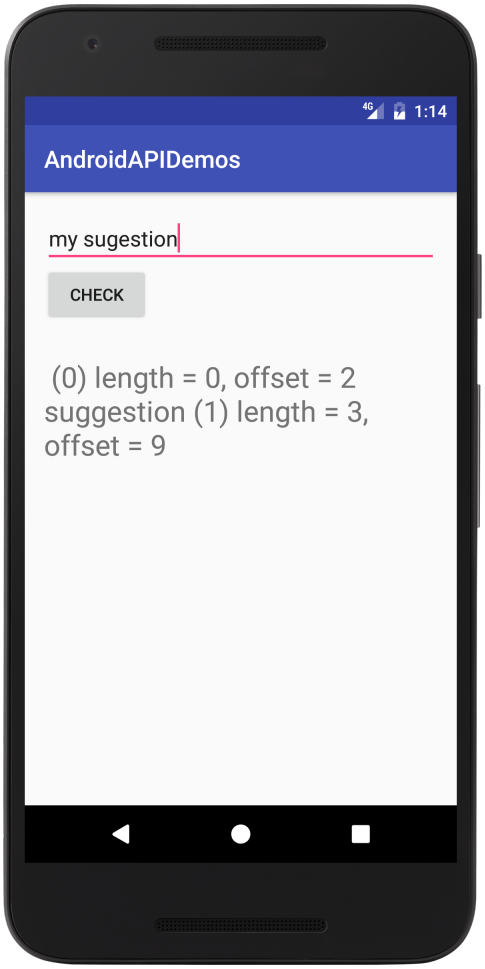
The Android API contains a *Spell Checker API* that can be accessed by anyone via the TextServicesManager class from API level 14 and from Jelly Bean (API level 16) it is able to check even sentences.

The usage is very simple, via the TextServicesManager a SpellCheckerSession can be created that allows setting the locale and some initial parameters:

```
TextServicesManager tsm = (TextServicesManager) getSystemServic
SpellCheckerSession spellCheckerSession = tsm.newSpellCheckerSes
```

The result can be retrieved by implementing the `SpellCheckerSessionListener` interface in the `onGetSuggestions(SentenceSuggestionsInfo[] sentenceSuggestionsInfos)` / `onGetSentenceSuggestions(SentenceSuggestionsInfo[] sentenceSuggestionsInfos)` methods.

The result is stored in the SentenceSuggestionsInfo array where the correct texts, the offset and all relevant information is stored.



A simple usage can be checked in this demo SpellCheckerActivity.

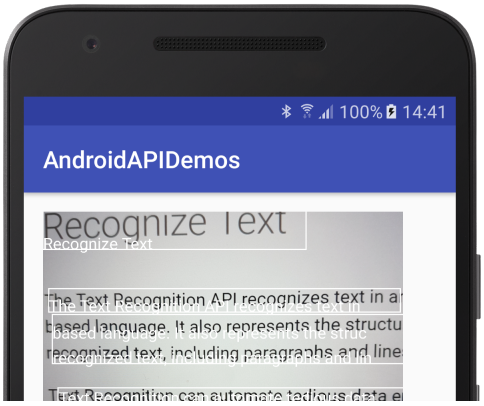
No. 2 - Text Recognizer

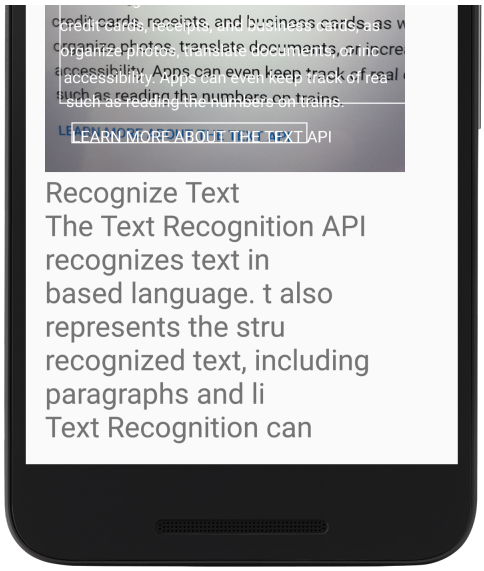
The next demo is a visual Text Recognizer from the *Google Play Services Vision API* that can be easily included in the project with a gradle dependency. Please note that you should never import the whole *Play Services*, because it is very large and typically only a small part of it is used. Here you can check the different *Play Services API dependecies*: https://developers.google.com/android/guides/setup

The Vision API contains three APIs:

- Face recognition
- Barcode scanning
- Text recognition

Next, we introduce how easy it is to use the Text Recognizer API.





In order to use the API, first the *Play Services Vision* has to be imported in the *build.gradle*:

```
compile 'com.google.android.gms:play-services-vision:10.0.1'
```

The usage is very simple, a TextRecognizer object is required to start the process:

```
TextRecognizer textRecognizer = new TextRecognizer.Builder(conte
```

For processing the results, a Detector.Processor interface has to be implemented. The detected texts arrive in a TextBlock array, from where the text coordinates and strings can be retrieved.

```
public class OcrDetectorProcessor implements Detector.Processor<
```

```
    @Override
    public void receiveDetections(Detector.Detections<TextBlock>
        ...
        SparseArray<TextBlock> items = detections.getDetectedIte
        ...
    }

    @Override
    public void release() {
    }
}
```

In order to use the TextRecognizer properly, a custom view is required that has a SurfaceView which can show the camera preview screen.
A simple demo can be found in the OCRActivity demo and the helper classes under the ocr package.

No. 3 - TimingLogger

In Android there is a special TimingLogger class that can be used easily to measure elapsed time between log messages in a sequence like this:

```
D/TAG_MYJOB: MyJob: begin
D/TAG_MYJOB: MyJob:      2002 ms, Phase 1 ready
D/TAG_MYJOB: MyJob:      2002 ms, Phase 2 ready
D/TAG_MYJOB: MyJob:      2001 ms, Phase 3 ready
D/TAG_MYJOB: MyJob: end, 6005 ms
```

In order to start the log process, the TimingLogger has to be initialized first:

```
TimingLogger timings = new TimingLogger("TAG_MYJOB", "MyJob");
```

Then with the *addSplit(...)* method a log entry can be created:

```
timings.addSplit("Phase 1 ready");
```

The log message is not displayed immediately in *LogCat* only after calling the *dumpToLog()* method:

```
timings.dumpToLog();
```

In order to use the `TimingLogger`, it has to be enabled for the TAG you use via adb shell:

```
setprop log.tag.TAG_MYJOB VERBOSE
```

A simple demo and usage example can be found in the TimingLoggerActivity.

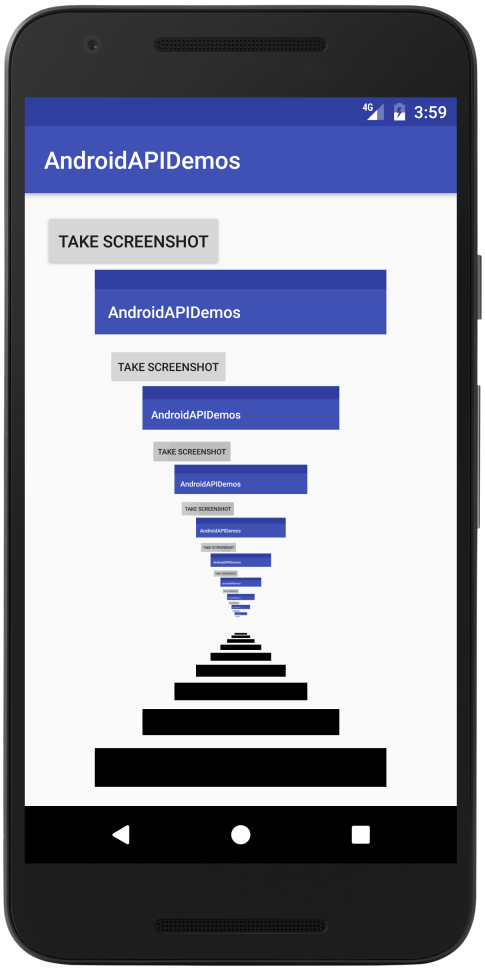
No. 4 - Taking screenshots

In several situations, it is very useful to take a screenshot from the phone or emulator. There are existing libraries for this like Falcon or from API level 21 it is possible to use the MediaProjection API even to stream the screen content and

system audio in real-time.

However sometimes it is enough to use the standard Android API for simply getting the content of the screen as a `Bitmap` via `getWindow()`:

```
View viewRoot = getWindow().getDecorView().getRootView();
viewRoot.setDrawingCacheEnabled(true);
Bitmap screenShotAsBitmap = Bitmap.createBitmap(viewRoot.getDraw
viewRoot.setDrawingCacheEnabled(false);
// use screenShotAsBitmap as you need
```



A demo can be checked in the ScreenCaptureActivity.

No. 5 - PDF Creation API

From API level 19 a PDF Creation API is available in Android that enables generating a PDF document from native Android content.

The key is the PDFDocument class that represents a PDF document, first a new Page has to be created with the `PDFDocument`'s `startPage([pageInfo])` method that requires a `PageInfo` object that we can create via a builder: `new PdfDocument.PageInfo.Builder(w,h,pageNum).create()`;

The following example method creates a *demo.pdf* file in the file system and puts the current screen content in the document:

```
public void createPdfFromCurrentScreen() {
    new Thread() {
        public void run() {
            // Get the directory for the app's private pictures
            final File file = new File(
                    Environment.getExternalStorageDirectory(), "

            if (file.exists ()) {
                file.delete ();
            }

            FileOutputStream out = null;
```

```
            try {
                out = new FileOutputStream(file);

                PdfDocument document = new PdfDocument();
                Point windowSize = new Point();
                getWindowManager().getDefaultDisplay().getSize(w
                PdfDocument.PageInfo pageInfo =
                        new PdfDocument.PageInfo.Builder(
                        windowSize.x, windowSize.y, 1).create();
                PdfDocument.Page page = document.startPage(pageI
                View content = getWindow().getDecorView();
                content.draw(page.getCanvas());
                document.finishPage(page);
                document.writeTo(out);
                document.close();
                out.flush();

                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(PDFCreateActivity.this, "
                    }
                });
            } catch (Exception e) {
                Log.d("TAG_PDF", "File was not created: "+e.getM
            } finally {
                try {
                    out.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }.start();
}
```

Thank you for reading our post, and stay tuned for the next part!

Péter Ekler

Péter Ekler works as a senior lecturer at BME AUT. He is responsible for Android, Mobile Platforms and Business Intelligence. He works as Android and backend team lead at AutSoft.

Budapest

Share this post

READ THIS NEXT

Filtering duplicates from a table in SQL server

It is a known problem in the world of multitiered application development that we want to apply some new...

YOU MIGHT ENJOY

Now you can use the Bottom Navigation View in the Design Support Library

UPDATE 2016.10.26. Added a paragraph after the list of available attributes for the control to mention how...