



Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,  
Arbeitsgruppe Software Engineering

# Scala on Android: problems and solutions

20. Juli 2015

Niklas Klein  
4387959  
niklas.klein@fu-berlin.de

Betreut durch Prof. Dr. Lutz Prechelt

## Abstract

The aim of this bachelor's thesis is to promote and advance the *Scala* programming language on the *Android* platform as an alternative to *Java*. Enjoying the benefits of *Scala on Android* is often circumvented by the numerous obstacles that its integration into the *Android* platform entails. Solving these issues is not impossible, but difficult—due to the lack of documentation.

For this very reason, this thesis is dedicated to the creation of a comprehensive, web-based documentation which simplifies working with *Scala on Android*. At first, it is necessary to identify the obstacles that developers are facing with the technology. By means of qualitative researching methods, developers are interviewed in order to reveal the particular problems they have been confronted with. The next step then is to create a sophisticated web-based documentation that provides guidance through the challenging topics of *Scala on Android*.

As a result of these efforts, the finished documentation is publicly available at <http://scala-on-android.taig.io/>.

## Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

20. Juli 2015

*Niklas Klein*

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Revealing the problems</b>	<b>5</b>
2.1	Researching technique . . . . .	5
2.2	Finding interview partners . . . . .	5
2.3	Preparing the interviews . . . . .	6
2.3.1	Interview outline . . . . .	6
2.3.2	Interview types . . . . .	7
2.3.3	Interview schedule . . . . .	8
<b>3</b>	<b>Conducting the interviews</b>	<b>11</b>
<b>4</b>	<b>Evaluating the interviews</b>	<b>12</b>
4.1	Identified problems . . . . .	12
4.2	Out of scope . . . . .	18
<b>5</b>	<b>Interview reliability</b>	<b>20</b>
<b>6</b>	<b>Outlining the documentation characteristics</b>	<b>22</b>
<b>7</b>	<b>Creating the documentation</b>	<b>25</b>
7.1	Collaboration and hosting . . . . .	25
7.2	Static page generator . . . . .	26
7.3	Implementation . . . . .	28
<b>8</b>	<b>Conclusion and Feedback</b>	<b>40</b>
<b>9</b>	<b>Scala on Android on the web</b>	<b>43</b>

# 1 Introduction

*Scala* is a programming language for the Java Virtual Machine (JVM) which “fuses object-oriented and functional programming” [2, p. 1] paradigms. It has plenty of similarities to the better known *Java* programming language and can seamlessly interact with its code [2, p. 2]. But with *Scala*’s bias towards immutable data structures, a uniform object model (every value is an object) [2, p. 3] and pattern matching [2, p. 13], the language allows the developer to be more expressive, leading to more concise code.

Applications for the *Android* platform are written in *Java* and do then get compiled to JVM byte-code. In a next step the byte-code is converted to an *Android*-specific format and is now ready for execution on the Android Runtime (ART), *Android*’s JVM implementation. Since compiled *Scala* code is also just ordinary JVM byte-code, the thought of converting it to the *Android* format stands to reason. The idea of bringing the advantages of *Scala* to the *Android* platform has already led to a variety of community developed tools that simplify this process, making it a considerable alternative to *Java*. Unfortunately, substituting the technologies is still not as easy as one might hope. It does in fact introduce some obstacles which are hard, but not impossible, to overcome.

Getting *Scala on Android* to work is primarily a matter of configuration which is unsatisfactorily documented, yet to make the procedure more accessible, this thesis is dedicated to creating the missing documentation with the goal to make the technology worth considering for a broader audience.

## 2 Revealing the problems

Having previously worked with *Scala on Android* intensively, I gained a rough overview of the difficulties that a developer faces when dealing with the technology. But in order to broaden this picture it was necessary to collect additional data from developers that also have experiences in this field.

### 2.1 Researching technique

Before actually reaching out to developers I had to learn about the researching methods that are available for this purpose, namely qualitative and quantitative research. Quantitative research is a suitable technique when “researchers have clear ideas about the type of information they want to access and about the purpose and aims of their research” [1, p. 72] whereas qualitative methods are used when “investigators are interested in understanding the perceptions of participants” [1, p. 72]. Since my goal was to discover difficulties with *Scala on Android* development, which I did not find during my work with the technology, qualitative research was obviously more suitable. Also the small size of the *Scala on Android* community would have made obtaining a reasonable sample size for quantitative research quite a challenge.

Since the development community is scattered across the globe I had to think about ways to find and reach out to potential peers in order to conduct a remote interview. This circumstance entailed the side benefit to automatically “reach a sample population that is in geographically diverse locations” [1, p. 82].

### 2.2 Finding interview partners

I planned to promote my intentions on relevant community platforms (such as *reddit/r/scala*, the *Scala* and the *Scala on Android* mailing lists) hoping to reach a diverse target group. If this approach had failed, I would have intended to contact developers that are actively committing in this area on *GitHub* directly.

My postings on the community portals enjoyed a surprisingly great popularity and enabled me to win five individuals over, willing to tell me about

their development experiences. At that time it was not possible for me to estimate whether five interviews would suffice to fully exhaust the topic. I therefore intended to elaborate on this issue when the interviews were completed, searching for further interview partners if it seemed necessary.

## 2.3 Preparing the interviews

### 2.3.1 Interview outline

As explained in *Qualitative research methods for the social sciences* I started by creating an interview *outline* which “lists all the broad categories” [1, p. 72] that are relevant to the study. The outline helped me to get a better idea of the information I wanted to obtain and established a basis where I could afterwards derive the actual interview questions from.

When talking to developers about *Scala on Android* I planned to collect data about the following topics (with a brief explanation of the particular purpose):

#### Background in software development

Getting an idea of the development stacks the interviewee is comfortable with and also what led to the decision of giving *Scala on Android* a try

#### Experiences with Scala and Android

Finding out if and how the technologies were used before they were brought together on the *Android* platform with the intention to get a clear picture about the importance of prior knowledge

#### Experiences with Scala on Android

The core of the conversation; which tools have been used, what were the use cases, as well as the overall satisfaction and experiences in detail

#### Opinions on Scala vs. Java on Android

Retrospectively, get the interviewees evaluations whether switching to *Scala* was worth the hassle and if they would stick to the technology in future projects

### 2.3.2 Interview types

There are numerous classifications for the structure of an interview available, however L. Berg identified the *standardized*, the *unstandardized* and the *semistandardized* interview types [1, p. 68] to which I refer to hereafter.

The different interview types vary primarily in the strictness of their schedule. An interview schedule is the catalogue of questions that the researcher uses as a guideline through the interview process. Depending on the interview type, this schedule does not only contain the actual questions, but also information about their order, optional questions to deepen certain topics or even temporal milestones.

A standardized interview is formally structured and offers “each subject approximately the same stimulus so that the responses to the questions, ideally, will be comparable” [1, p. 69]. It is suitable for studies where researchers “have fairly solid ideas about the things they want to uncover during the interview” [1, p. 69] while leaving little room to discover anything beyond that.

In contrast to the standardized interview, the unstandardized alternative does not rely on a questions schedule at all. The idea behind this strategy is that it is impossible to “know in advance what all the necessary questions are” [1, p. 70] and that the interviewer must instead come up with appropriate follow-up questions in the particular situations. This approach is useful to reveal information “when researchers are unfamiliar with respondents’ life styles [...] or customs” [1, p. 70].

A semistandardized interview is located somewhere between the strictly scheduled and completely unscheduled interview types. A researcher who conducts a semistandardized interview is equipped with a rough question schedule and has the freedom to digress and to go deeper into certain topics in order to gain further information which may be valuable to the research [1, p. 70].

I settled with the semistandardized interviewing technique for my research. A standardized interview seemed inappropriate as I aimed to learn about different perceptions and to gather insights which I missed during my exposure to *Scala on Android*. Furthermore, I wanted to rely on some sort of schedule because I did not conduct interviews before and felt uncomfortable with the idea of coming up with ad-hoc questions as practiced in an unstandardized interview.



### 2.3.3 Interview schedule

I intended to prepare an interview which should take about one hour to conduct [1, p. 82-83]. Below is the question schedule that I developed on the basis of the interview outline. The questions are grouped in distinct topics which are first explained. Reasons why I decided for this particular order or what kind of information I planned to obtain are added where applicable.

Questions on the second indentation level served me as a reminder to digress deeper into certain topics but were not necessarily asked (“scripted questions” [1, p. 92]).

#### 1. Introduction

Introducing myself and the intentions of this interview to the interviewee, not disclosing my opinions (“breaking the ice” [1, p. 83]). Then ask the respondents to introduce themselves and to talk about their background in software development and whether they have any questions about the process or background of this interview [1, p. 83]. Also ask for permission to record the interview for the evaluation process, and assure that it will not be published and anonymity is guaranteed.

In this section I aim to get an understanding of the interviewees’ working environments and habits as well as a basic idea of how happy they are to try out new technologies.

- (a) Could you please introduce yourself and tell me about your background in software development?
  - i. For how long have you been doing ...?
  - ii. What kind of projects do you work on?
  - iii. What is your role in these projects?
  - iv. What are your favorite software stacks?
  - v. How come you like / use ... so much?
  - vi. With how many fellow developers do you usually work on such projects?
- (b) Do you code for a living?
- (c) Do you have hobby projects?

#### 2. *Scala on Android*

Since the interviewee is expecting to talk about *Scala on Android* this topic is placed right after the introduction. If I tried to talk about *Scala* and *Android* separately at this point, chances would be that the interviewee would constantly divagate to *Scala on Android*. In this section I aim to gather the respondent's experiences, thoughts and emotions about the technology as detailed as possible.

- (a) What are your experiences with *Scala on Android*?
- (b) Why did you try it?
- (c) How did you get started?
  - i. Which learning resources and tools did you use?
  - ii. What caused the biggest troubles?
  - iii. Was it painful?
  - iv. Did you consider to plunk down and go back to *Java* at some point?
  - v. Was it a team project?
  - vi. What else didn't work out quite as you were expecting?
- (d) Which libraries and tools did you use?
  - i. What are the differences to your *Java on Android* setup?
- (e) Did you use it for commercial or work related projects?
  - i. Are there any apps available which I can have a look at?
- (f) How do you stay up to date with *Scala on Android*?
- (g) What annoys you about *Scala on Android*?
  - i. Compile time?
  - ii. Build configuration?
  - iii. ProGuard?
  - iv. Testing?
  - v. Performance?
  - vi. Community?
  - vii. Documentation?
- (h) Are you still using *Scala on Android*?
- (i) Are you still using *Java on Android*?
- (j) Which build tool plugin do you use?
  - i. Do you like it?
  - ii. What do you think about its documentation?

### 3. *Scala* and *Android*, separately

After excessively talking about *Scala on Android* I want to find out how well the interviewees knew each of the technologies before they combined them on the *Android* platform.

- (a) Did you make experiences with *Android* development before you brought *Scala* in?
  - i. How did you get into *Android* development?
  - ii. When did you start to develop for the *Android* platform?
  - iii. Do you (still) like it?
  - iv. What is your opinion about the *Google-Android* tool-chain (e.g. *Gradle* and *Android Studio*)?
- (b) Did you make experiences with *Scala* development before you started with *Scala on Android*?
  - i. How did you get into *Scala* development?
  - ii. When did you start to develop with *Scala*?
  - iii. Do you (still) like it?
  - iv. What is your opinion about the *Scala* ecosystem (e.g. *Scala Build Tool (SBT)*)?

#### 4. *Scala* vs. *Java* (on *Android*)

By now I should be able to estimate how well the interviewee knows each of the languages, which puts an ad-hoc comparison of the technologies in perspective.

- (a) Which stack would you choose if you started a new project today?
  - i. Under which circumstances would you pick the other tool-chain?
- (b) What are your favorite *Scala* features that *Java* does not offer?
- (c) Do you occasionally think that you are missing out on new tools and features when you are working with *Scala on Android*?
- (d) Which path would you suggest to a freshman that wants to learn *Android*, but has no experience in either *Java* or *Scala* yet?
  - i. What if he knew *Scala* already?
  - ii. What if he knew *Android* already?

#### 5. Finish

Give thanks for the interviews, ask if there is something which the interviewees wanted to add and if they were interested in having an early look at the documentation to provide their thoughts and feedback.

### 3 Conducting the interviews

The five interview partners which I recruited on the community platforms were located in the United States, the United Kingdom, Estonia and China. Due to the significant time differences to some of these locations it was particularly difficult to settle an appointment. I therefore had to conduct the U.S. and China interviews at night time. All of the conversations were carried out via *Skype* calls. I left the interviewees the option to decide whether they preferred to have a video or just a voice call, but all without exception favored the voice variant. As a consequence of this the interviews went off as ordinary telephone interviews, missing the opportunity to catch non-verbal cues [1, p. 82].

Getting to know each other and talking about technology was something the respondents seemed very eager about. In fact, once we started talking about the second schedule point, *Scala on Android*, conversations tended to drift into an unstandardized interview. Most of my questions were already answered by the detailed stories I got to hear. In this situation I fell back on asking follow-up questions in order to deepen topics when appropriate. Despite me thinking that asking the interviewee for build configurations or source code snippets might go too far, my peers were happy to share those with me when I appeared to be interested.

All in all, none of the interviews could be conducted within the intended time frame of one hour. But since the conversations did not feel compulsive, my share of the talking was little and the gained information extremely valuable, I interpreted this as a good sign and did not attempt to shorten the conversations [1, p. 81].

The sample group appeared excited about the intentions of my research and acknowledged that a well-engineered *Scala on Android* documentation would be a valuable resource for their own work and could also have a major impact on the growth of the community.

## 4 Evaluating the interviews

Since content analysis is described as “the most difficult aspect of any qualitative research project” where “it is impossible to establish a complete step-by-step operational procedure that will consistently result in qualitative analysis” [1, p. 102] I just orientated loosely on the concept of *systematic filing systems* [1, p. 103] with *short-answer sheets* [1, p. 105].

The former, *systematic filing systems*, is a method where the interviews are first torn apart into atomic chunks of information that are relevant to the research. These information are then indexed and classified by some schema (e.g. the interview outline) that helps to organize the interview answers for further use [1, p. 103]. *Short-answer sheets* are primarily an addition to *systematic filing systems*, rather than a separate content analysis method. The researcher creates a brief summary for every question of the interview schedule and attaches them to respective indexes of the *systematic filing systems*. This makes it significantly easier to find or look up certain conversation details [1, p. 105].

While post-processing the interviews I relied on the voice recordings to classify the obtained information into the four interview outline categories. Afterwards I summarized the information of each outline in note form to have a compact overview of the gathered information. The next step then was to identify problems, mentioned by the interviewee, which could be resolved if there was a better documentation available that covers the topic. I refrained from applying quantitative measures in this process and honored issues that were only mentioned by one or two interviewees equally important as issues that arose more often.

### 4.1 Identified problems

Following is a list of all problems that I compiled from the interviews. They are annotated with explanations of the exact issues that interviewees had with the topic. The problems are then summarized in the fashion of *key questions* that need to be answered by the documentation in order to resolve the issues that interviewees had with *Scala on Android*.

#### Build tool

Leveraging *Scala* on the *Android* platform can be a surprisingly difficult undertaking. Once the decision for *Scala on Android* was made,

the first obstacle that a novice faces is the choice of the build tool. There are plugins available for multiple build tools, but it is difficult to find out in which way they differ and which one suits the developer's requirements best.

### **Gradle**

Configured via the *Groovy* programming language, officially documented and supported to be used with *Android*

### **SBT**

Configured via the *Scala* programming language, the de facto standard build-tool for *Scala* projects

### **Maven**

A well established build-tool in the JVM environment, configured via *XML*

Furthermore, since these plugins are community developed they tend to neglect certain features and are more likely to contain bugs because they are not as well-tried as the official tool-chain which is backed by *Google*.

#### **Key questions**

- Which build tool should be used?
- What are the differences between the build tool (plugins)?

### **ProGuard and the 65k-limit**

When the decision for a build tool was made and the developers managed to setup a basic project they will now be confronted with *ProGuard*. *ProGuard* is a JVM tool that is able to shrink, optimize and obfuscate byte-code. It is included in the *Android* tool-chain as an opt-in service since the very beginning. *Java* developers may use it to minimize the application size by stripping out unused dependency classes or to make the code harder to reverse engineer by obfuscating identifiers. These services do unfortunately come with the price of two major downsides.

#### **Increased build times**

When the program code has been compiled to *Java* byte-code, *ProGuard* analyzes it, removes unused code and obfuscates identifiers. This process is very time consuming and increases build times dramatically.

## Configuration

*ProGuard* naturally fails to analyze code dependencies that rely on runtime reflection or if a library references a Java Development Kit (JDK) class which is not part of the *Android* Software Development Kit (SDK). The developer therefore has to detect these issues in the library code and adjust the *ProGuard* configuration with a tool-specific syntax. Given the increased build times, reapplying and debugging a configuration change is no job for the impatient.

A *Java* developer has the option to balance the pros and cons, or to run *ProGuard* only in the release build process while disabling it during development. A *Scala* developer, however, depends on *ProGuard* and is forced to use it. This is due to the so called *65k limit*. An *Android* application is only allowed to contain 65,536 methods (including the application's libraries). If the application succeeds this limit, the build will fail. Developing with *Scala* requires to depend on the *scala-library* which already suffices to exceed this limit. *Scala on Android* therefore requires the developer to maintain a proper *ProGuard* configuration and to execute *ProGuard* for every build.

This does not only pose a problem through increased compile times or tedious configuring, but it is also a rather inaccessible topic. Among *Java* developers it is not in wide use because there is usually no reason to do so. Unfortunately the available documentation is accordingly minimalistic. The *Scala on Android* tool-chain takes this even further, building additional tools around *ProGuard* (e.g. caching) that require further configuration. All these issues combined make *ProGuard* the major pain point for *Scala on Android* development.

### Key questions

- What is *ProGuard* good for?
- Are there any heuristics on how to approach configuration problems?
- What is the *ProGuard* cache good for?
- How to configure the *ProGuard* cache?

## Development environment

None of the popular Integrated Development Environments (IDEs) (such as *Android Studio*, *IntelliJ IDEA* or *Eclipse*) provides a seamless integration for *Scala on Android*. This can be frustrating as the developer has to give up some of the convenience that his IDE offered and instead run certain tasks manually via the command line.

### Key questions

- How to use *Scala on Android* with a particular IDE?
- Which IDE features will no longer work with *Scala on Android*?
- Which editor works best with *Scala on Android*?

## Command line

As a consequence of the lacking IDE support, developers have to execute build, deployment and testing related tasks via the command line. This emerges to be a significant obstacle for people that are used to do all of this with the help of their IDE. Furthermore, the *Android* plugins for *Gradle* and SBT introduce a range of new commands that are difficult to grasp, even for developers that are already comfortable with the build tool itself.

### Key questions

- How does the command line workflow look like?
- Which commands are important for *Scala on Android* related tasks?

## Build configuration

Similar to the issues people are having with the command line, configuring a build with additional plugin specific parameters requires a deep understanding of the available settings and their effects. This is a fundamental problem as most of the here listed problems must be resolved via the build configuration.

### Key questions

- Which build plugin configuration keys are important?
- How to manage (*Android*-specific) dependencies?
- How to configure SBT; why are there two distinct ways?

## Parcelable

*Parcelable* is an *Android*-specific Inter-process communication (IPC) framework that serves as an alternative to *Java's Serializable* mechanism. It is significantly faster than the runtime reflection based *Serializable* alternative because the developer has the obligation to spell out the entire serialization logic by himself. The *Android* developer documentation highly recommends to rely on the *Parcelable* framework even though its implementation is somewhat tedious. Besides



implementing the *Parcelable* interface, the developer has to fulfill an additional contract: creating a static field named *CREATOR* which has to hold an instance of *Parcelable.Creator*. This contract, however, can not be fulfilled when coding in *Scala*, because the language does not have a concept of statics.

#### Key questions

- Is it possible to use *Parcelable* with *Scala on Android*?
- How should the *Parcelable* contract be implemented without statics?
- Does *Scala* offer alternative serialization methods?

### Testing

The *Android* SDK comes with support for unit- and instrumentation testing. Especially the latter is a book of seven seals in the *Scala on Android* environment. It remains unclear if it is possible to integrate instrumentation tests with the current generation of build tool plugins.

Furthermore, a *Scala* developer would generally prefer to adopt a testing framework that is tailor-made for his language (e.g. *ScalaTest*) rather than working with the provided *jUnit* Application programming interface (API). Another desirable feature is to make use of *Robolectric*, a famous library that allows to run *Android* tests on the developer's *JVM*, instead of deploying it to a device or emulator (which is a time-consuming process). Ideally, it should be possible to use *ScalaTest* in combination with *Robolectric*.

#### Key questions

- How to do unit tests?
- How to do instrumentation tests?
- Is it possible to use *ScalaTest*?
- Is it possible to use *Robolectric*?
- Can tests be run from an IDE rather than the command line?

### Debugging

In opposite to testing, setting code breakpoints and running the debugger can not be triggered from the command line that easily. An IDE simplifies this process significantly and provides a better visualization. It remains unclear whether it is possible to use IDE debuggers due to the poor *Scala on Android* integration.

### Key questions

- Can the debugger be run from an IDE?

## Library projects

Besides creating an executable Android Application Package (APK), an *Android* codebase may also be distributed in the Android Application Library (AAR) or Android Application Package Library (APKLIB) format which serve the purpose of creating reusable components that may be incorporated into an *Android* project as a dependency. These formats are very similar to the common Java Archive (JAR) (which can of course also be included into *Android* projects) but extend it by adding *Android*-specific resources to the package (such as the application manifest, graphics or internationalization files).

Learning how to configure a *Scala on Android* build in order to create such a library project was a common desire among the interviewees.

### Key questions

- How to create an AAR package?
- How to create an APKLIB package?

## Packaging and signing

An application is only ready for the official *app store* when it is properly packaged (which requires some post-processing of the package, such as *zipalign*) and signed with a valid developer certificate. There are several ways to achieve this goal, but it remains unclear how to automate this process in such a way so that it is not necessary to place cleartext password configuration files within the project (which should generally be avoided, but especially when working with Version Control Systems (VCSs)).

### Key questions

- How to sign an application?
- Is there a way to sign via SBT with password prompt?

## Learning resources

It is difficult to find up-to-date information about *Scala on Android*. On the one hand, the search results are commonly very *noisy* and include *Android*-related topics which have no connection to *Scala*. On the other hand, the resources that show up tend to be rather outdated

and no longer correct. This is due to the fast development of the *Scala* and *Android* platform, and thus also the build tool plugins that try to keep up with this pace.

#### Key questions

- Are there any other articles or blog posts about a particular topic?

### Getting help

In the interviews, people reported that they had a hard time trying to find somebody to take a look at their particular problems. At *StackOverflow*, a famous online *questions and answers* community for developers, their questions went unnoticed and they did not know who else to address. This caused them to get stuck on minor problems for days, causing lots of frustration.

#### Key questions

- Where to ask other developers for help?

## 4.2 Out of scope

During the course of the interviews a couple of valid topics were mentioned which I had to drop from the list of solvable problems. This is primarily due to the lack of personal experience with those and, after further consideration, the resolution that I will not be able to acquire enough expertise to educate a public audience in the limited time of this researching project.

### Gradle

Originally, the recommended build system for *Android* SDK applications used to be *Apache Ant*. Later, when the first projects for *Scala on Android* appeared, SBT emerged to be the more reliable tool for the purpose and a small ecosystem of libraries and projects began to originate around this approach. Then, in 2013, *Google* announced the official migration to *Gradle*, a *Groovy* based build system for the JVM. A plugin for a *Gradle*-based *Scala on Android* integration appeared shortly after and is becoming increasingly popular. It lowers the entry barrier significantly for *Android* developers that already got used to *Gradle*.

Since most *Scala on Android* specific problems are solved by tweaking the build configuration, incorporating *Gradle* into the documentation

would have meant to investigate and solve each problem for both tools, SBT and *Gradle*.

## JNI

Java Native Interface (JNI) is an interface that describes how a *Java* application can interact with native code (*C/C++*). In conjunction with the *Android* Native Development Kit (NDK) the native program parts can be bundled with the ART byte-code into one application. Given the documentation of the *Scala on Android* SBT plugin, this feature is also available to *Scala* users.

Due to my lacking experience with native code and the observation that JNI is only relevant to a minority of *Scala* users, I decided to drop this topic.

## 5 Interview reliability

It is now necessary to clarify whether the chosen researching method, conducting interviews, was a reasonable approach to reveal a developer’s potential obstacles when dealing with *Scala on Android*. Odds are that this strategy left significant information undiscovered. Equally questionable is whether five interviews sufficed to gather the desired information. In order to get a perception of the information that might have been missed out, it is helpful to examine the actually gathered information.

A suitable measure for this purpose is to analyze the interviews (in order of conduct) for the amount of new problems that each of them revealed. Arguing that if the last interviews did only result in little or no additional insights, this implies that the information to be gathered might be exhausted and conducting further interviews would thus not reveal any new insights.

As shown in table 1, interview I helped to discover two additional problems which I did not come up with during my preparations. The second and third interviews revealed one issue each. The fourth interview did not introduce any new insights. If the last interview hadn’t revealed another issue, the trend would have suggest that the amount of undiscovered problems is exhausted. Instead it appears like the fourth interview, yielding no new information, is an anomaly.

When taking a closer look at interview V it strikes that the new problem, JNI, is the only issue that has not been mentioned by anyone else. The topic is furthermore considered as *out of scope* for being only relevant to a minority of *Java*, let alone *Scala*, developers. From this perspective it seems reasonable that JNI may be neglected in this context. It is thus a valid conclusion that conducting more interviews may not have necessarily led to significant new insights because the two latest encounters could not add any valuable new information.

Problem \ Interview	Author	I	II	III	IV	V
<i>Gradle</i>	•		•			•
SBT	•	•		•	•	•
<i>Maven</i>	•					
<i>ProGuard</i>	•	•	•	•	•	•
Development environment	•	•		•	•	•
Command line				•		•
Build configuration	•	•		•	•	•
Parcelable	•		•			
Testing	•	•	•	•		•
Debugging		•	•			
Library projects	•		•			
Packaging and signing	•			•	•	
Learning resources		•	•	•	•	
Getting help			•			•
JNI						•
Amount of new problems (compared to previous interviews)	-	2	1	1	0	1

Table 1: Comparison of the *Scala on Android* related problems that were revealed in each interview. The first column, *Author*, shows the problems which I put together from my own experience before conducting the interviews. In the last row, there is a numeric summary of how many new problems were discovered in every interview.

## 6 Outlining the documentation characteristics

With the intention to create a documentation in the form of a publicly available website, the opportunity occurs to go way beyond static, textual content. A website allows to add features that have the potential to enhance the overall reading and learning experience. The features and characteristics which I considered important in order to create a valuable learning resource are listed below.

### Content

It is easy to give in to the temptation of developing *yet another great feature*. But however the development of the documentation proceeds, the actual, textual content always has to be the number one priority.

### Progressive enhancement

Every feature on top of the basic content has to be implemented according to the strategy of *progressive enhancement*. Meaning that a feature that requires the client's browser to have a certain ability (e.g. executing *JavaScript* code) should never have a negative impact on browsers that are not able to fulfill the requirement (besides missing out on the feature in question).

As an illustrative example of this strategy, imagine a timezone specific *DateTime* value embedded into the Hypertext Markup Language (HTML), such as *Wed Jun 24 17:48:48 2015 +0200*. To improve readability, the *JavaScript* API *Date.toLocaleString()* is used to convert the time according to the user's timezone (e.g. *24.6.2015, 17:48:48*). If the client's *JavaScript* engine does not implement this function, or the client does not run *JavaScript* at all, the raw time information must still be accessible.

### Static

The content has to be delivered as static HTML files and should not require any serverside processing. This simplifies hosting, improves response times and makes it easier to edit the sources.

### Asynchronous JavaScript and XML (AJAX)

To fulfill the *progressive enhancement* and *static* requirements, the document should refrain from loading crucial content dynamically via AJAX. This way the website is also more accessible to search engine bots that analyze the content. They are at risk to miss out dynamic content.

## **Semantics**

Not only the text, but also the underlying HTML structure should semantically make sense. This is necessary to increase accessibility for screen readers and does also improve the understanding of the website for search engine bots. This requirement extends to good practices, such as always providing meaningful *alt* tags for images as well as using *title* tags to provide contextual information that may improve readability.

## **Collaboration**

Since the contained information are always at risk to become outdated rather quickly, the document should be manageable in a highly collaborative way allowing any reader to propose or make changes to the document easily. This does of course also apply for fixing misspellings or substantial errors, improving the wording or even the layout.

## **Readability**

Textual content has to be presented in a way that it is pleasantly readable. This relates especially to fonts, their sizing, line spacing, but also the overall contrast. Furthermore, the user should be able to adjust the font size without breaking the website or making content inaccessible.

## **Gateway**

There are plenty of good blog posts and documents scattered across the web for nearly all the discovered problems. The website has to provide sufficient information to solve the users' issues but also point them the way to more detailed information or to necessary prerequisites that might be taken care of first.

## **Responsive design**

The website should render well, independently of the reader's screen dimension or density.

## **Navigation**

It has to be easy to advance through the pages. Skipping to the next or previous chapter should be a prominent option, but also navigating to every other topic of the documentation.

## **SEO**

Every page has to contain meta tags and meaningful titles embedded into the document. A good search engine ranking does not only increase visibility on the web, but with proper meta information it also



allows the users to jump immediately to the content they care about right from the search results.

### **Syntax highlighting**

Programming code that is embedded into the documentation should be easily identifiable as such. It has to be rendered with a monospace font and should feature syntax highlighting applied on top of that.

### **Examples**

The content should be enriched with practical examples of the described topics wherever possible.

### **Indicate outdated information**

Information that are at risk to become outdated must be highlighted as such. If the users find themselves in situations where they are unable to recreate given instructions they should first consider that the given information might no longer be valid rather than searching for mistakes in their code. In addition to that, users should be able to see when a paragraph was last modified.

### **Fun**

Last but not least, it should never be frustrating to read and use the website. Layout and features may never get in the way of content. It must be fun and beautiful to look at.

## 7 Creating the documentation

### 7.1 Collaboration and hosting

Before scaffolding the actual project or picking tools that might help me putting the documentation together, I thought about where to host and how to enable collaborative content editing. The obvious choice for my purposes was *GitHub*, as it solves numerous problems in an elegant and user-friendly way.

#### Hosting

*GitHub* provides a feature called *GitHub Pages*. With the help of a specific *Git* branch (“gh-pages”), all files on this branch (from the latest revision) can be accessed via Hypertext Transfer Protocol (HTTP) on port 80. This means that static HTML files are served as websites to any browser. The service is free of charge, is based on a reliable server infrastructure and leverages *GitHub*’s Content Delivery Network (CDN) for improved response times. Updating the content is as easy as committing changes to the *gh-pages* branch and pushing them to the remote repository.

By default, the website’s address is `http(s)://user-name.github.io/project-name`. But *GitHub* allows advanced Domain Name System (DNS) settings, enabling the repository maintainer to point custom domains to the project page.

#### Collaboration

Each project has an issue tracker that allows *GitHub* users to notify the repository maintainer about problems they discover. These issues are public and everybody is free to join the discussion to propose solutions or suggestions.

Taking this one step further, users can enrich issues by adding code (or textual content) that attempts to solve the problems they were facing. This procedure is then called a *pull-request*. For the repository maintainer, applying the provided patches to the project is as easy as pushing a single button to confirm.

To simplify the audition process of *pull-requests* it is possible to integrate third-party Continuous Integration (CI) services that automatically compile and test proposed changes and then add their results and logs to the issue discussion.

### Content editor

Every directory and file of a repository that is hosted on *GitHub* can be browsed on their website. In addition to that, files can also be edited with an online text editor that allows to quickly update content without cloning the repository. This is especially useful for text-based projects like documentations because it is not necessary to compile or test the project before submitting a patch.

Another positive side effect of this approach is that a user's edits automatically result in a *pull-request* which can then be easily audited and approved as described earlier.

### Popularity

*GitHub* is incredibly well-known and established among developers. Users tend to know the described collaboration workflows which lowers the barrier to actually bring oneself to participating.

## 7.2 Static page generator

Although the goal of the documentation is to consist of static HTML files only, I intended to integrate a static page generator in order to optimize the development process. This leads to a build step, necessary to compile the source code to HTML files before committing to *gh-pages*. But that's a low price to pay, considering the indispensable advantages of this approach.

### Templating

With a static site generator it is possible to split a document into several files which may then be included as desired. This is especially useful to avoid code duplication considering that certain elements like a page header or footer have to be included into every page. It does also allow to split each section of a long article into separate files which makes it easier to find one's way around when editing content.

### Dialects

It is possible to process languages that could not be interpreted by a browser. E.g. converting *CoffeeScript* to *JavaScript*, Syntactically Awesome Style Sheets (SASS) to Cascading Style Sheet (CSS) or *Markdown* to HTML.

### Linting

To enforce best practices linting tools can be integrated into the build process that perform desired validations.

## Custom tasks

An important requirement for the means of the documentation is the integration of custom build steps. They offer great flexibility and room for improvements. This way it is possible, for instance, to extract file information like the last time of an edit and inject them into the document, rather than maintaining this information manually.

## Post processing

In the last step of the build process it is now possible to apply minification tools to the HTML, CSS and *JavaScript* sources. This file sizes are reduced significantly, improving the time it takes to load a page even further.

I originally started with a static page generator, called *Roots*, that promised to solve most of these problems out of the box. It also integrates *Markdown* processing which I intended to use to format textual content, because it is easier to maintain than HTML code. But I realized rather quickly that such a tool is too specific and limiting for my purposes (especially regarding the custom build steps). Also *Markdown* emerged to not fit in, as it didn't allow me to specify HTML classes which would have been necessary to highlight certain paragraphs. My intention to create central sitemap, resource links and abbreviation indexes were also at risk as they could only be used with *Markdown* by creating custom extensions. This cluttered up the documents and was in no way better to maintain than a more versatile HTML templating system.

As a consequence of these circumstances I migrated to *Gulp*. *Gulp* is, broadly spoken, a *JavaScript* build tool for front-end development. The tool is famous for its simplicity and flexibility, but its greatest strength is the community around it that created thousands of plugins for every imaginable use case. Setting it up and getting the basics working to a similar range of functions like *Roots* provides them was initially very challenging. But in the long run I benefited from its flexibility as I was not forced to make a single compromise in this context.

Instead of *Markdown* I switched to an HTML templating system called *Nunjucks*. It allowed me to split up my files and include them as necessary. To organize textual content separately from the HTML markup I was able to structure the files in such a way that content files only consist of headline, paragraph and image tags, keeping them maintain- and readable.

To create CSS assets I chose to integrate the SASS preprocessor into the build chain as I made the experience in previous projects that it simplifies de-

velopment tremendously. Also, I found myself writing very little *JavaScript* code, so I kept that as simple as possible, not including any preprocessors or other additional tools except for *jQuery*.

## 7.3 Implementation

### Content

The documentation as a whole is subdivided in numerous chapters which are each represented by a respective HTML page. A table of contents is located on the root page offering to jump right into a topic of choice.

1. Introduction
2. About this documentation
  - An explanation of page features, but also a call not to shy away from contributing
3. Prerequisites
4. Build tool
  - Comparison of *Gradle* and SBT, leaving room to add *Gradle* documentation in the future
  - (a) SBT
5. Project setup
6. Editors and IDEs
  - (a) *IntelliJ IDEA*
  - (b) *Android Studio*
7. Working with the command line
8. Dependencies
9. *ProGuard*
  - (a) Cache
10. *Typed Resources (TR)*
11. *Parcelable*
12. Testing
  - (a) *Robolectric*
13. Library projects
14. Publishing

Beside these content chapters, there is a handful of additional pages that aim to help interpreting the documentation or to solve problems of which I took notice through the interviews.

- Fork me on *GitHub*

A typical catch phrase to point to a project's *GitHub* repository. Clicking this link on the documentation website forwards the user directly to the respective *GitHub* page.

- Getting help

A list of development communities and resources that the readers should consider to consult when they get stuck with *Scala on Android*.

- Sources

A large index of noteworthy websites (along with a content description) that contain information which might be relevant to educate further about a certain topic.

- Software

A list of the software that was used to create the examples in the documentation. It mentions the particular version of the software and a download link.

- Abbreviations

An abbreviation index.



Figure 1: Side by side comparison of the documentation website on common screen sizes

## Responsive design

Through CSS *Media Queries* the page automatically adjusts its properties in such a way that pleasant reading on all screen sizes and densities can always be guaranteed. Included graphics and icons are, wherever possible, delivered as Scalable Vector Graphic (SVG) assets. Furthermore, the page refrains from using mouse hover events for crucial interaction since these would be inaccessible for mobile visitors with touchscreens (see figure 1).

## Navigation

At the end of each chapter the user has the possibility to advance to the next topic to provide a natural reading flow (see figure 2).

Attached to the sticky header bar there is also a navigation hidden behind the *hamburger* icon that enables the user to jump to an arbitrary chapter (see figure 3).

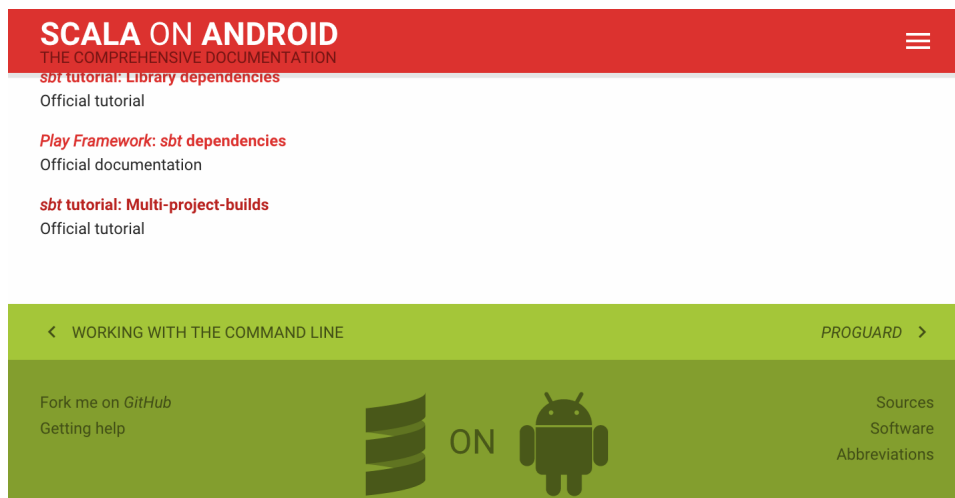


Figure 2: Footer navigation that contains a link to the previous and next chapter

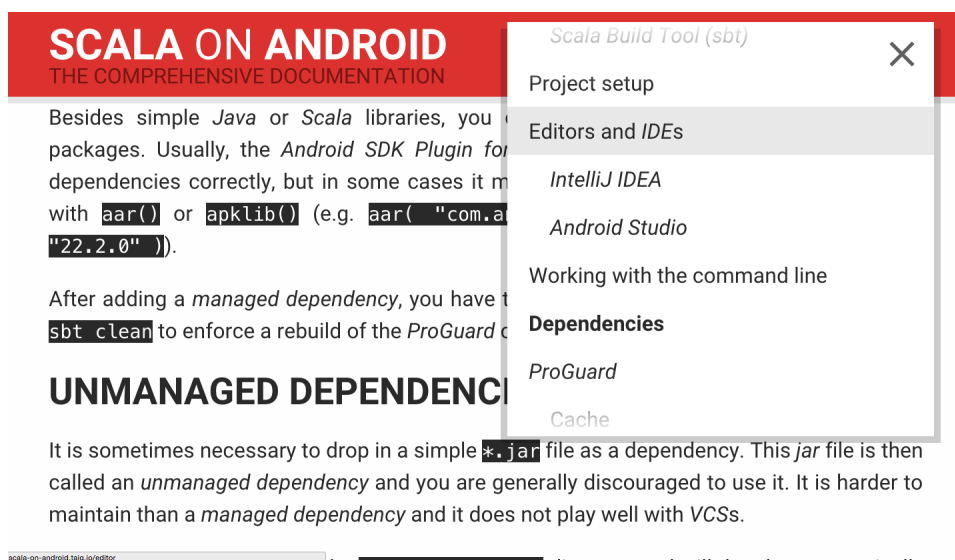


Figure 3: Main navigation, attached to the sticky header

### Metabox

Next to each paragraph is a signal reception icon that intends to indicate information reliability. The ranking ranges from *Fragile*, to *Moderate* up to *Sustainable* and is a representation of my personal impressions which I gathered while using *Scala on Android* or researching the particular topic (see figure 4).





Figure 4: Signal reception icons to indicate information reliability. *Fragile* (left, red) and *Moderate* (middle, orange), *Sustainable* (right, green).

Clicking on one of these signal icons leads to a *Metabox* showing up. It contains an additional textual description of the information reliability, but also the date when the paragraph has been edited lately, enabling the reader to interpret the ranking in this context (see figure 5).

## SCALA ON ANDROID

THE COMPREHENSIVE DOCUMENTATION

development or not, the memory consumption does vary greatly. You should be equipped with at least 4GB of RAM to cover the common use cases that are demonstrated in this documentation. Since compiling *Scala* code is known to be a tedious process, a fast CPU might improve this for you.

### JAVA DEVELOPMENT KIT (JDK)

*Android* supports *Java 7*, so the **JDK 7** should also be your choice. If you don't have a newer version of the *JDK* installed, this will work as well. Make sure to set your compile target properly to version 1.7, which will be explained in the next chapter.

**INFORMATION RELIABILITY** ✕  
**Moderate**  
 Might become outdated

**LAST EDIT**  
 21.6.2015, 01:07:43

[Edit on GitHub](#)

With the upcoming release of *Scala 2.12*, **Java 8 will be required**. Therefore *Scala 2.12* can

Figure 5: A *Metabox* that shows a *Moderate* information reliability ranking.

Another important element of the *Metabox* is the “Edit on *GitHub*” link. When clicked, a new tab with a text editor appears, encouraging the user to make changes. The text editor is part of the *GitHub* website and once an edit has been made, a *pull-request* is created automatically that I, as the the project maintainer, may then review and merge into the project to apply the patch. This approach provides a convenient collaboration workflow with the underlying VCS and also relieves me from the struggle to develop a similar feature.

### Further reading

At the end of most chapters is the “Further reading” section. It contains links to articles and websites I used in order to put the provided information together. Since the overall structure of this section is the

same on every page and numerous links were relevant to multiple chapters, I constructed a central sources index which contains entries that are shaped as shown below.

```
'android':
{
  description: 'Official website',
  url: 'http://www.android.com/',
  title: 'Android'
},
'android-api-guides':
{
  description: 'Official documentation',
  url: 'https://developer.android.com/guide/index.html',
  title: 'Android API Guides'
},
...
```

With the help of the HTML templating engine I could then create a function `section_further_reading()` that generates the section with minimal effort (see figure 6).

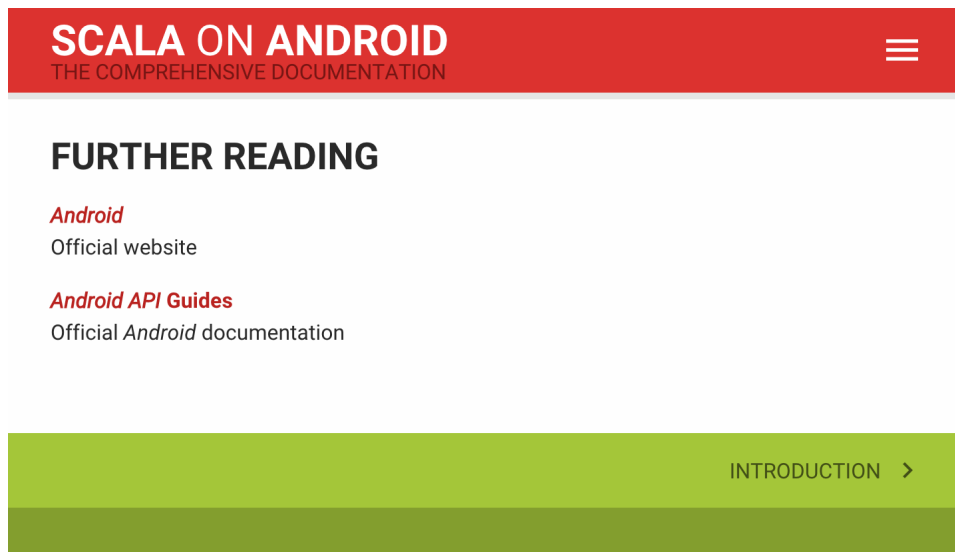


Figure 6: The further reading section at the end of a chapter.

```
section_further_reading( [ 'android', 'android-api-guides' ] );
```

Based on the index, the entire sources page is also generated automatically as part of the build process.

## Abbreviations

Similar to the sources index, there is also a name index. I found myself repeating names of companies and technologies all over the place and intended to display them with an italic font. Additionally, in many cases, there are also common abbreviations for these names. To stop repeating myself and to be less prone to errors and inconsistencies, I introduced the names index.

```
...
'robolectric': { name: 'Robolectric' },
'robotest': { name: 'RoboTest' },
'sbt':
{
  abbreviation: 'sbt',
  name: 'Scala Build Tool'
},
'sbt-plugin': { name: 'Android SDK Plugin for SBT' },
...
```

After defining a couple rendering functions, I was able to access the index from templates in this fashion:

```
names.get( 'robolectric' ) // <em>Robolectric</em>
names.get( 'sbt' ) // <em>Scala Build Tool (sbt)</em>
names.abbr( 'sbt' ) // <abbr title="Scala Build Tool">sbt</abbr>
names.name( 'sbt' ) // <em>Scala Build Tool</em>
```

The major advantage of this approach is the semantically valuable rendering of abbreviations with the `<abbr />` tag. If a reader is unfamiliar with an abbreviation, the title tooltip reveals the full meaning. Furthermore, I used all entries that contain a name and an abbreviation value to render the abbreviations page as part of the build process.

## Highlighted paragraphs

A major reason that brought me to replace *Markdown* with an HTML templating system was the desire to highlight paragraphs that should stand out from the rest of the content. In *Markdown* documents this could be achieved by embedding HTML (which defeats the purpose of *Markdown*) or, a surprisingly common practice, by abusing the block-quote syntax for this purpose. The templating engine allowed me to create a simple function that takes care of generating an appropriate markup, including an icon that illustrates why a paragraph was highlighted (see figures 7 and 8).

```
{% call alert( 'warning' ) %}  
  <p>Lorem Ipsum</p>  
{% end call %}
```

This works for common text paragraphs, blockquotes and also code blocks without abusing HTML elements in a context where they do not belong.


## SCALA ON ANDROID

THE COMPREHENSIVE DOCUMENTATION

most features of Java and adds a variety of highly useful capabilities, such as: type inference, functional programming paradigms, implicits and traits (as an improvement to Java interfaces) to just mention a few of them. Scala's steadily growing community has recently created a couple of ambitious projects aiming to make *Scala on Android* a compelling experience.

### TARGET AUDIENCE

This documentation teaches you how to properly use and configure *Scala on Android*. It neither teaches you *Scala* nor *Android* development.



When you have no experience with either *Scala* or *Android* development yet, please consider that both technologies come with quite a steep learning curve. It is highly recommended to first get a basic understanding for both technologies separately.

### YOU ARE NOT SAFE HERE

Leaving the safe haven of Google's Java environment does not come without a price. To enjoy the benefits of *Scala on Android* you have to not only sacrifice official tooling support and deal with longer compile times but you also have a much tougher build configuration ahead of you to get basic things working. If you still think that *Scala* is worth the hassle, this documentation is there to guide you past the rough edges.

### FURTHER READING

**Scala**  
Official website

Figure 7: An example of a paragraph that is highlighted as an alert box to draw attention.

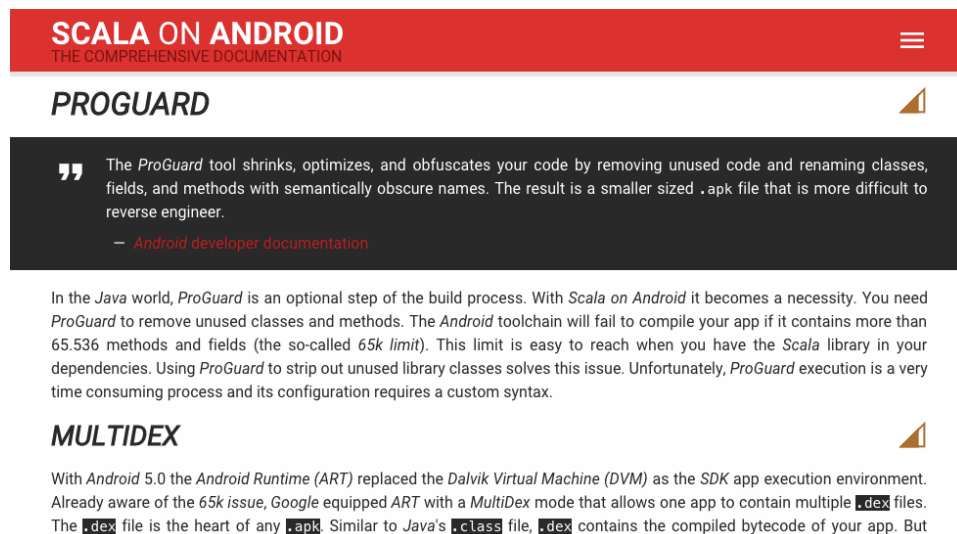


Figure 8: An example of a blockquote that is styled in the fashion of a highlighted paragraph.

## Syntax highlighting

In terms of styling, code blocks are basically just highlighted paragraphs (see figure 9). But they were designed in such a way that the code may be placed in a separate file rather than inserting right into the `alert()` function body.

```
// Render code file and apply Scala syntax highlighting
code( 'page/proguard/configuration.scala', 'scala' )
```

The actual highlighting of the program code is done by *highlight.js* which is usually executed on the client side. Since it was also possible to integrate this process into the *Gulp* build process easily I preferred to encode the syntax highlighting into the delivered HTML documents. As a consequence the client does not need to load the additional *JavaScript* dependency and can also benefit from the feature when *JavaScript* is not available.

As an extension of *Android*'s `R` class, the *Android SDK Plugin for SBT* introduces a the `TR` class. It generates type safe mappings from ids in *XML* views to their respective class. This is explained best by a simple example.

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">

    <TextView
        android:id="@+id/my_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

## JAVA

Based on the *XML* view definition above, you will typically search for the `TextView` by its id to specify the title at runtime. This is a very common use case, and in *Java* it works as shown below.

```
public class MyActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
    }
}
```

Figure 9: An example of code blocks that have syntax highlighting applied.

## *Hello Scala*

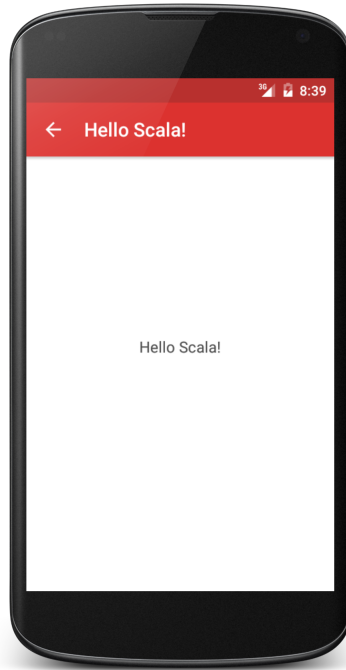


Figure 10: A screenshot of the *Hello Scala* sample application

Besides documenting how to set up and configure a *Scala on Android* project I saw the need to create a comprehensive and executable project that implements the described practices. It focuses on a lightweight configuration that is preconfigured with common dependencies (such as the *Android* support libraries) and test cases. Just like the documentation, the *Hello Scala* project is available on *GitHub* and is intended to be cloned in order to have an up-to-date project template. The *Scala on Android* documentation contains numerous references to its sister project, pointing out that there is a working example waiting to be explored.

*HelloScala* aims to meet three important needs:

### **Curiosity**

Newcomers should be able to clone and get it running on their device with minimal effort. As a demonstration of how easy a *Scala on Android* set up can be, it aims to encourage the visitor giving *Scala on Android* a serious try.

**Documentation**

As it implements the best practices advertised in the documentation, it serves as a functional showcase and learning resource.

**Scaffolding**

Rather than starting a new project from scratch, it should be a considerable alternative to clone the *HelloScala* project as the foundation. It features a sophisticated SBT configuration that can easily be trimmed down to specific needs rather than gathering up necessary configuration values from the web.



## 8 Conclusion and Feedback

Creating a comprehensive documentation for *Scala on Android* was a challenging exercise. I managed to address every issue which was discovered by the help of the interviews, but in some rare cases I saw no other solution than to surrender and accept that there was currently no way to fix this particular problem. These documentation gaps were a disappointing experience and made it incredibly difficult for me to still bring myself to publishing the document, yet alone asking the developers for feedback explicitly.

Luckily both, the former interviewees and the community as a whole, appeared to appreciate the website. After setting the documentation up and publishing it on the web, I reached out to the interview partners asking them for their thoughts about my work result.

Hi \_\_\_\_\_,

I just published the current state of my efforts for the *Scala on Android* documentation here on Reddit. But since you shaped the form of the project through our interview I am especially interested in your thoughts about it. It's far from perfect and turned out to be way more work than I expected, but I'd still like you to have a look at it. And, if time permits, provide me with brief feedback (3-4 sentences suffice) of what you like and don't like about it.

Thank you,  
Nik

---

You really did a great job there. Especially the hello-scala project is a huge win for me. I can't believe why we didn't have anything like this before. Maybe you should put more emphasize on collaboration to keep the project active? It'd be a shame if it was a wasteland in a couple of months. I'll try to stick around and make a contribution every now and then so that we can keep it up to date.

---

I'm pleasantly surprised by the overall appearance, it looks very clean and elegant. Your articles seem well thought out and I've enjoyed reading so far. The chapter about parcelables was especially useful for me. I always kinda avoided that with Scala (and hated it with Java). Incredibly good job. Will definitely recommend!

---

I think you did good work in that short time. I did not read all of it now but looks very good. Too bad you couldn't add gradle. Maybe I should try out sbt ;)

Unfortunately, not all interviewees responded to my inquiry but other community members that did not participate in the interviews also left some valuable and encouraging feedback when I promoted the website on their platform.

This [is] one of those things I've been looking for, for a while.

Please, can you keep updating?

I on the site would emphasize that others could contribute on Github more prominently; maybe it will help growth.

What can I, Scala and Android noob, do to help?

---

This is great work. Thanks so much.

---

I haven't tried to follow the steps but I glanced through the whole thing and it seems like a great start!

---

I am very pleased to bookmark this for perusal as it is something I want to learn very soon – in fact, I'm only delaying so I can brush up on my Scala in the first place. I hope to look at it in the near future!

It is still to be clarified whether I spent too much time on making the documentation fun, rather than improving its content.

I spent more time playing with your `a: hover` css effect than actually reading the documentation. I am easily amused :/

Since I received this overall positive feedback I am particularly excited about the project and looking forward to refining it. Having the opportunity to create such a valuable resource for the development community as part of my thesis turned out to be a privilege.

## 9 Scala on Android on the web



Figure 11: The *Android* mascot dressed in a robe that illustrates the red helix of the *Scala* branding, serving as the *Scala on Android* logo.

Within the scope of this thesis, several projects emerged on the web which you may find under the following addresses:

- *Scala on Android* documentation  
<http://scala-on-android.taig.io/>
- *Scala on Android* repository  
<https://github.com/taig/scala-on-android/>
- *Hello Scala* repository  
<https://github.com/taig/hello-scala/>

## Index of abbreviations

<b>AAR</b>	Android Application Library
<b>AJAX</b>	Asynchronous JavaScript and XML
<b>API</b>	Application programming interface
<b>APK</b>	Android Application Package
<b>APKLIB</b>	Android Application Package Library
<b>ART</b>	Android Runtime
<b>CDN</b>	Content Delivery Network
<b>CI</b>	Continuous Integration
<b>CSS</b>	Cascading Style Sheet
<b>DNS</b>	Domain Name System
<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IDE</b>	Integrated Development Environment
<b>IPC</b>	Inter-process communication
<b>JAR</b>	Java Archive
<b>JDK</b>	Java Development Kit
<b>JNI</b>	Java Native Interface
<b>JVM</b>	Java Virtual Machine
<b>NDK</b>	Native Development Kit
<b>SBT</b>	Scala Build Tool
<b>SASS</b>	Syntactically Awesome Style Sheets
<b>SDK</b>	Software Development Kit
<b>SVG</b>	Scalable Vector Graphic
<b>VCS</b>	Version Control System

## References

- [1] Bruce L. Berg. *Qualitative research methods for the social sciences*. Allyn & Bacon, 2001.
- [2] Martin Odersky. *An overview of the Scala programming language*. École Polytechnique Fédérale de Lausanne, 2004.