

Coding Conventions

Android

Coding Conventions Android

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| No | Date | Content | Author |
|  |  |  |  |
|  |  |  |  |

# Approvals

|  |  |  |  |
| --- | --- | --- | --- |
| No | Date | Role | Approved By |
|  |  |  |  |

# **Table of Contents**

Android

[Revision History 1](#_Toc528683023)

[Approvals 1](#_Toc528683024)

[**Table of Contents** 2](#_Toc528683025)

[**Table of Figures** 2](#_Toc528683026)

[Introduction 3](#_Toc528683027)

[Purpose 3](#_Toc528683028)

[Related Documents 3](#_Toc528683029)

[Android 4](#_Toc528683030)

[**Naming Conventions** 4](#_Toc528683031)

[**Indentation** 5](#_Toc528683032)

[**Layout** 5](#_Toc528683033)

[**Exception Handling / Logging** 5](#_Toc528683034)

[**Comment** 7](#_Toc528683035)

# **Table of Figures**

No table of figures entries found.

# Introduction

## Purpose

The purpose of this document is describing about our code conventions

## Related Documents

-

# **Android**

### Naming Conventions

Non-public, non-static field names start with m.

Static field names start with s.

Other fields start with a lower case letter.

Public static final fields (constants) are ALL\_CAPS\_WITH\_UNDERSCORES.

For example:

public class MyClass {

public static final int SOME\_CONSTANT = 42;

public int publicField;

private static MyClass sSingleton;

int mPackagePrivate;

private int mPrivate;

protected int mProtected;

}

Layout files should match the name of the Android components that they are intended for but moving the top level component name to the beginning. For example, if we are creating a layout for the SignInActivity, the name of the layout file should be activity\_sign\_in.xml.

| **Component** | **Class Name** | **Layout Name** |
| --- | --- | --- |
| Activity | UserProfileActivity | activity\_user\_profile.xml |
| Fragment | SignUpFragment | fragment\_sign\_up.xml |
| Dialog | ChangePasswordDialog | dialog\_change\_password.xml |

Resources files

Resources file names are written in lowercase\_underscore.

### Indentation

Use four (4) space indents for blocks

### Layout

Braces do not go on their own line; they go on the same line as the code before them:

class MyClass

{

int func()

{

if (something) {

// ...

} else if (somethingElse) {

// ...

} else {

// ...

}

}

}

We require braces around the statements for a conditional.

if (condition) {

body();

}

### Exception Handling / Logging

Don't ignore exceptions, such as:

void setServerPort(String value) {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e)

{ }

}

Acceptable alternatives (in order of preference) are:

* Throw the exception up to the caller of your method.

void setServerPort(String value) throws NumberFormatException {

serverPort = Integer.parseInt(value);

}

* Throw a new exception that's appropriate to your level of abstraction.

void setServerPort(String value) throws ConfigurationException {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e) {

throw new ConfigurationException("Port " + value + " is not valid.");

}

}

* Handle the error gracefully and substitute an appropriate value in the catch {} block.

/\*\* Set port. If value is not a valid number, 80 is substituted. \*/

void setServerPort(String value) {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e) {

serverPort = 80; // default port for server

}

}

* Catch the Exception and throw a new RuntimeException. This is dangerous, so do it only if you are positive that if this error occurs the appropriate thing to do is crash.

/\*\* Set port. If value is not a valid number, die. \*/

void setServerPort(String value) {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e) {

throw new RuntimeException("port " + value " is invalid, ", e);

}

}

* As a last resort, if you are confident that ignoring the exception is appropriate then you may ignore it, but you must also comment why with a good reason:

/\*\* If value is not a valid number, original port number is used. \*/

void setServerPort(String value) {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e) {

// Method is documented to just ignore invalid user input.

// serverPort will just be unchanged.

}

}

### Comment

At the first time create new function, create the standard comment on the top of the function explain its purpose and meaning/usage of parameters if there is any.

When we change the return function or list of parameters, we also need to update the top comment to reflect the new function

/\*\* Returns the correctly rounded positive square root of a double value. \*/

static double sqrt(double a) {

...

}

When add more code or change existing code of the function, add one more comment line to the top comment as below:

/\*\* Returns the correctly rounded positive square root of a double value. \*/

//Add new rule to fix bug 11111, ld

static double sqrt(double a) {

...

}

Therefor whenever we change the code, we will have things to verify to make sure that it will not cause any troubles for previous bugs/features