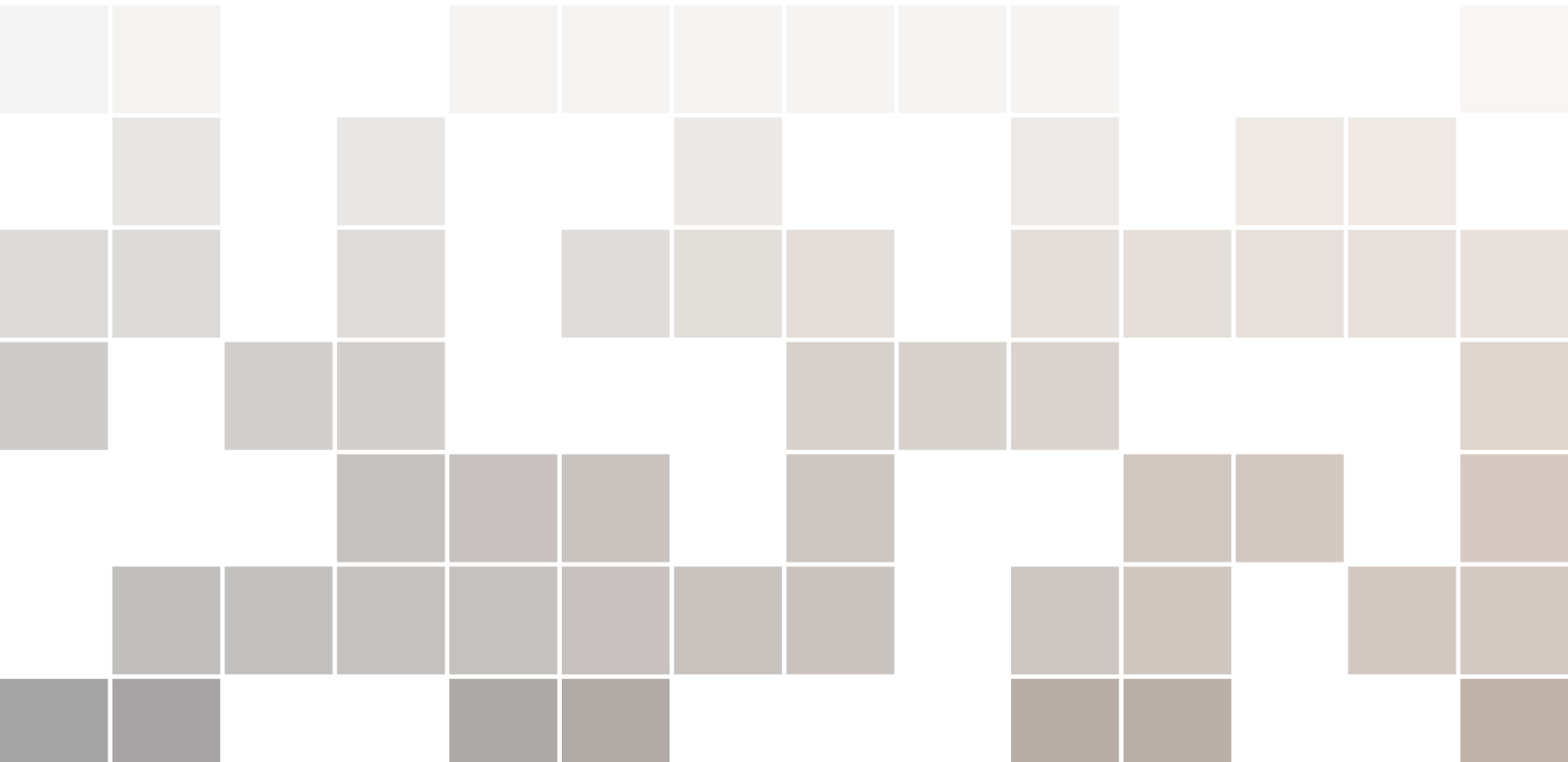


GIRAF Design Manual

*

*



Contents

I	Preface	
1	Terms	9
1.1	<i>GIRAF</i> Software Suite	9
1.2	Contextual	9
1.3	Screen Real Estate	9
2	Box Model	11
II	Core Elements	
3	Icons	15
3.1	<i>GIRAF</i> Software Suite Icon	15
3.2	Application-specific Icons	15
3.3	Icons used to represent functionality	15
3.4	Icons and when to use them	16
3.4.1	User Management	16
3.4.2	Camera	16
3.4.3	Microphone	16
3.4.4	Media	17
3.4.5	Arrows	17
3.4.6	Version control	17
3.4.7	Data Management	18
3.4.8	Utilities	18
3.4.9	Miscellaneous	19
4	Image Representation	21
4.1	Appearance	21
4.2	Marking	22
4.3	Indicator overlay	22

5	Typography	23
5.1	Fonts, Sizes and Colors	23
6	Tone of Voice	25
6.1	Short and Precise	25
6.2	Addressing Users	25
7	Application structure	27
7.1	Top Bar	27
7.1.1	Back Button	27
7.1.2	Help Button	27
7.2	Side Bar	27
7.3	Bottom bar	28
7.4	Content	28
7.5	Combinations of Bars	28
7.5.1	Combination #1	29
7.5.2	Combination #2	29
7.5.3	Combination #3	29
7.5.4	Combination #4	30
7.6	Clickable Elements	30
7.6.1	Element Margin	30
7.6.2	Container Padding	31
8	Colors	33
8.1	Text and Background	33
8.2	Buttons	33
8.3	Action Bar	34
8.4	Week indicators	34
8.5	Page Indicator	34

III

Components

9	Progress Bar	37
10	Dialog	39
10.1	Confirm dialog	39
10.2	Notify dialog	39
10.3	Profile selector dialog	39
10.4	Waiting dialog	40
10.4.1	Long tasks	40
10.5	Custom buttons dialog	41
10.6	Inflatable dialog	41

IV

Appendix

A	Threading AsyncTask	45
A.1	Android GUI Thread	45
A.1.1	GUI Thread	45
A.1.2	AsyncTask	45
A.1.3	Long Tasks	45
B	How to use dialogs	47
B.1	Confirm dialog	47
B.2	Notify dialog	48
B.3	Profileselector dialog	49
B.4	Waiting dialog	51
B.5	Custom buttons dialog	52
B.6	Inflatable dialog	53

V

Index

Index	57
--------------------	-----------



Preface

1	Terms	9
1.1	<i>GIRAF</i> Software Suite	
1.2	Contextual	
1.3	Screen Real Estate	
2	Box Model	11

1. Terms

1.1 *GIRAF* Software Suite

When referring to the “*GIRAF*-software suite” throughout the manual, we mean the entire *GIRAF* project being developed by the current Software 6 semester at Aalborg University. This extends to all the individual applications and their dependency applications being developed, which at the current time of writing includes:

- Giraf (Main Launcher)
- Administration (Profile Manager)
- Sekvens (Sequence)
- Ugeplan (Week Schedule)
- Piktosearch (Picto Search)
- Pictoplæser (Picto Reader)
- Kategoriværktøjet (Category Manager)
- Kategorispillet (Category Game)
- Tidstager (Timer)
- Livshistorier (Life Story)
- Stemmespillet (Voice Game)

1.2 Contextual

Describe what w
mean by context

1.3 Screen Real Estate

Describe what w
mean by screen

2. Box Model

This design manual utilizes the standard box model that consists of content, padding, border and margins as seen in Figure 2.1. The main difference to notice is that *margin* is the outer spacing and that *padding* is the inner spacing on elements.

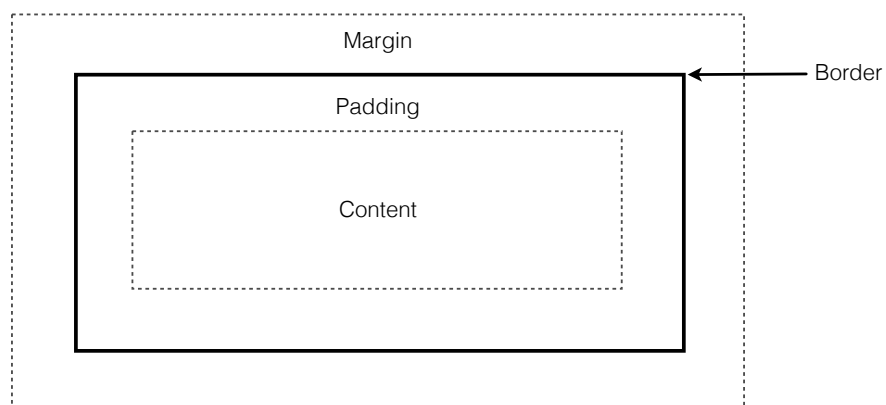


Figure 2.1: The Box Model

One should use *margin* when spacing between elements is desired, and if one would like spacing from the content of the element to the border of the element, *padding* should be used.



Core Elements

3	Icons	15
3.1	GIRAF Software Suite Icon	
3.2	Application-specific Icons	
3.3	Icons used to represent functionality	
3.4	Icons and when to use them	
4	Image Representation	21
4.1	Appearance	
4.2	Marking	
4.3	Indicator overlay	
5	Typography	23
5.1	Fonts, Sizes and Colors	
6	Tone of Voice	25
6.1	Short and Precise	
6.2	Addressing Users	
7	Application structure	27
7.1	Top Bar	
7.2	Side Bar	
7.3	Bottom bar	
7.4	Content	
7.5	Combinations of Bars	
7.6	Clickable Elements	
8	Colors	33
8.1	Text and Background	
8.2	Buttons	
8.3	Action Bar	
8.4	Week indicators	
8.5	Page Indicator	

3. Icons

Throughout the *GIRAF*-software suite different icons will be used to reference different applications and functionalities. Refer to the sections below to see how and when to use these icons.

3.1 *GIRAF* Software Suite Icon

The different applications in the *GIRAF* software suite may want to refer to the software suite itself. To do this, the overall application icon seen in Figure 3.1 can be used. This icon will also be used in some situations for indicating activity.



Refer to a section describing activity icons

Figure 3.1: Overall application icon for the *GIRAF* software suite

3.2 Application-specific Icons

Each application in the *GIRAF* software suite must have its own icon. This icon should reflect the content and functionality of the application and must not be ambiguous. All application icons should furthermore use the icon-base seen in Figure 3.2.

Rendered icons must be available in sizes defined in the Android Iconography article. The foreground of the icon must be clear in any of these sizes. Furthermore, the icon must not appear smudged or otherwise distorted on any scale.



Figure 3.2: Base for all application specific icon

3.3 Icons used to represent functionality

Specific icons may be used in buttons (See `GirafButton` in *GIRAF Components*) to represent functionality. The foreground of these icons must be gray-scaled and be quadratic (square). The

functionality that the icon represents should be reflected in the icons itself. The icons should use the icon-base seen in Figure 3.3.

Note 3.3.1 Please be aware that the actual icons should not be designed/rendered using the base as described above. Instead use the `GirafButton` from the *GIRAF Components* library. This will allow you to only design the actual foreground and not worry about the background (base).



Figure 3.3: Base for all functionality-specific icons

3.4 Icons and when to use them

This section will describe the different icons that must be used throughout the applications in the *GIRAF* software suite. Please notice that icons in this section are displayed on a button-like background. Icons may be used in different contexts if necessary.

3.4.1 User Management

Switch User: Used to indicate that the current profile can be switched



Back: Used to indicate that the user can go back



Log out: Used to indicate that the user can log out of the application



3.4.2 Camera

Camera: Used to indicate that a camera can be used. For instance to take a picture



Switch Camera: Used to indicate that the camera used can be switched. For instance from rear camera to front camera



3.4.3 Microphone

Microphone: Used to indicate that the microphone in the tablet may be utilized. For instance for recording speech



Microphone (off): Used to indicate that the microphone is unavailable



Microphone (on): Used to indicate that the microphone is currently in use



3.4.4 Media

Play: Used to indicate that something is playing



Record: Used to indicate that something is recording



Stop: Used to indicate that either something playing or recording can be stopped



3.4.5 Arrows

Consider if these should be available

Arrow (down): X



Arrow (left): X



Arrow (right): X

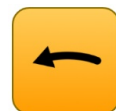






Arrow (up): X











3.4.6 Version control

Undo: Used to indicate that an action can be undone







Redo: Used to indicate that an undone action can be redone	
Synchronize: Used to indicate that data can be synchronize to/from the server	
Save: Used to indicate that actions performed can be saved	
Cancel: Used to indicate that some action can be canceled	



3.4.7 Data Management

Add: Used to indicate that a new instance of something can be created	
Rotate: Used to indicate that something can be rotated	
Resize: Used to indicate that something can be resized	
Copy: Used to indicate that something can be copied	
Edit: Used to indicate that something can be edited	
Delete: Used to indicate that something can be deleted	
Change Pictogram: Used to indicate that a pictogram can be changed or set	
Search: Used to indicate that a search-action can be performed	

3.4.8 Utilities

Help: Used to indicate that the user might seek help	
Settings: Used to indicate that the user might adjust settings	
Landscape to Portrait: Used to indicate that the orientation of the tablet can be changed from landscape mode to portrait mode	
Portrait to Landscape: Used to indicate that the orientation of the tablet can be changed from portrait mode to landscape mode	

3.4.9 Miscellaneous

Accept: Used to indicate that something is accepted. For instance when user is presented with a confirmation	
Choose: Used to indicate that the user should make a choice	

4. Image Representation

4.1 Appearance

In the *GIRAF* software-suite there exist various types of images, such as pictograms and profile pictures. Images should be quadratic, have a white background regardless of content and have a black border, as seen in Figure 4.1.

Refer to black

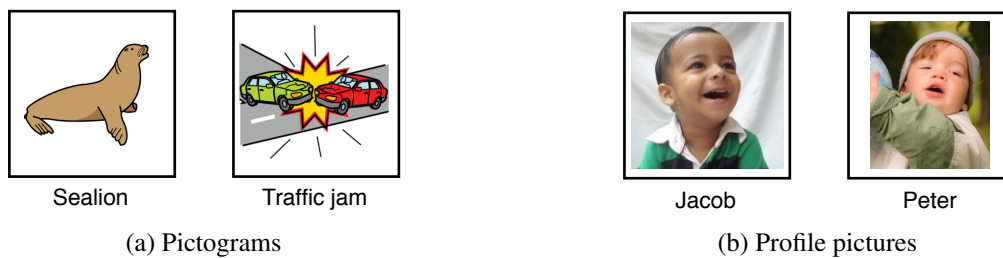


Figure 4.1: Image representation

Note 4.1.1 Profile pictures, as seen in Figure 4.1b, should, as the first profile (Jacob), fit to the canvas. However, many profile pictures are captured in a different way as seen in the second profile (Peter). It is wanted that in the future, when profile pictures are captured, that they only allow for pictures that have the layout of the first profile. For the sake of convenience and backwards compatibility we allow old profile pictures to be as the second.

The image view should have a 10 dp padding so that the content gets spacing to the border as seen in Figure 4.2.

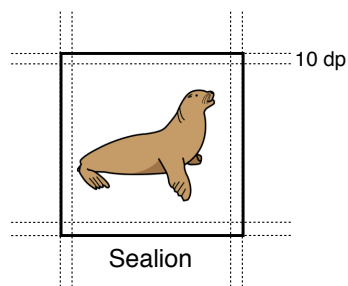


Figure 4.2: Image representation

4.2 Marking

It is often possible to mark images throughout the suite, and whenever this happens one should mark the background of the selected item with an orange color, as seen in Figure 4.4.

orange color

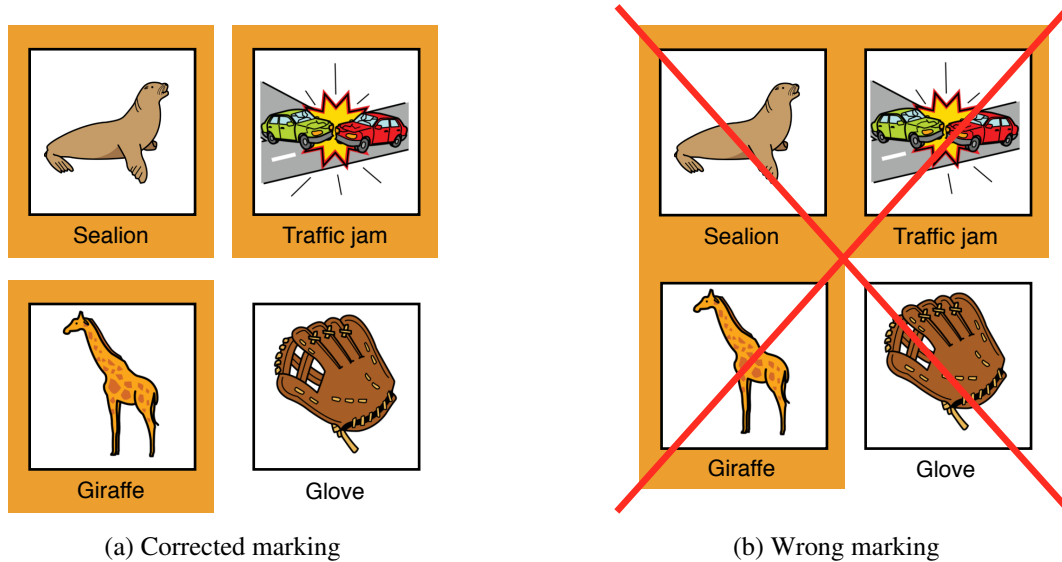


Figure 4.3: Marking of images

Note 4.2.1 It is important that when having multiple marked images that there are some margin between these elements. This should be implemented as seen in Figure 4.3a, and not as seen in Figure 4.3b.

4.3 Indicator overlay

If one wants to indicate that an image is editable, for instance when picking an icon for a category, the image should have an overlay indicating this as seen in Figure 4.4a. Other types of indicators should look similar, e.g. when indicating that an image is a placeholder for a category it is indicated as seen in Figure 4.4b.

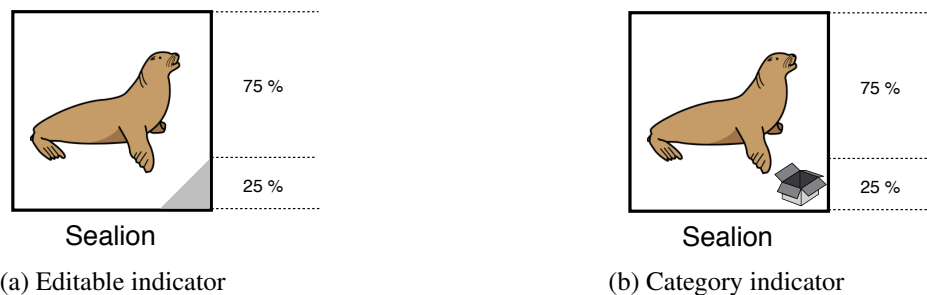


Figure 4.4: Indicator overlays

5. Typography

5.1 Fonts, Sizes and Colors

When creating text fields in the *GIRAF* software suite, is it imperative that they look alike across the entire suite. For this reason one should at all times use the default font-type in the applications.

The size of any text across the applications should always be given in “SP”, and should always be readable on both 7 and 10 inch tablets. It is up to the developers of the individual applications to decide upon a specific text-size, but it should fit the general layout of the suite.

All text should by default be black (#000000) across the suite when placed in buttons, views, etc. When creating hint text, e.g. when there are no applications present in the launcher, one should use the android implementation of Darker Gray¹. When using the showcase library to highlight features, the text color of the help text should be white (#FFFFFF), so it is readable on the dark background.

Update this section so that it references the color part

Insert a preview of the font

¹http://developer.android.com/reference/android/R.color.html#darker_gray

6. Tone of Voice

When displaying messages to the user the following rules must be respected.

6.1 Short and Precise

All messages must be short and precise. If a part of a sentence is redundant, remove it. However, you must provide enough information to cover the whole situation.

■ **Example 6.1 — Correct use.** The application can only be started from a guardian profile

■ **Example 6.2 — Incorrect use.** The application could not be started

■ **Example 6.3 — Incorrect use.** The application could not be started unless you start it from a guardian profile. Please log onto a guardian profile to gain access to this application.

6.2 Addressing Users

When referring to people with Autism Spectrum Disorder (ASD), use the word *citizens*. When referring to either institutional- or legal guardians, use the word *guardian*.

■ **Example 6.4 — Correct use.** The application cannot be started from a citizen-profile

■ **Example 6.5 — Incorrect use.** The application cannot be started from a child-profile

7. Application structure

7.1 Top Bar

All activities in every application, except the home-activity in the Launcher-application, must have a top bar. This top bar should look like the example in Figure 7.1. This top bar should provide a short and simple title with enough information to allow the user to know where in the application he or she is. The height of the top bar must be *56dp*. Furthermore, this top bar must include at the two buttons described in Section 7.1.1 and Section 7.1.2.



Figure 7.1: Top bar example

Add reference to colors used and describe the gradient

Note 7.1.1 Such a top bar is easily achieved by letting all of your Activity extend the *GirafActivity* from the *GIRAF Components* library. Doing this will also implement the back button described in Section 7.1.1.

7.1.1 Back Button

The left-most button in the top bar must be a back button. This button must have exactly the same functionality as the back button on all Android devices. The icon of this button must be...

Insert reference to section where icon is described

7.1.2 Help Button

The right-most button in the top bar must be a help button. This button must provide the user with some useful help information regarding the current screen which the user is presented with. For instance if the user is assigning applications to users, the help might be a guide on how to do this correctly.

7.2 Side Bar

Side bars need not, but may be contextual. Side bars may be used to switch between content of applications - for instance if the application is split into two parts (side bar and content). A side bar should look like Figure 7.2. Side bars may not use more than 20% of the total screen estate. When using a side bar it must appear on the left side of the screen. The height of the side bar must be exactly the same as the container that it is inside.

Add reference to colors used and describe the gradient

Note 7.2.1

Write a note describing how this can be made using *GIRAF Components*



Figure 7.2: Side bar example

7.3 Bottom bar

Bottom bars must be entirely contextual and depend on the current content displayed in the current activity. A bottom bar should look like Figure 7.3.

Reference to the
used and de-
the gradient

Note 7.3.1

Write a note describing how this can be made using *GIRAF Components*



Figure 7.3: Bottom bar example

7.4 Content

The main content of applications should be focused in the center of the layout and any menu bars should be above, under, and to the sides of the main content.

Note 7.4.1 We recommend using Android `Fragment` instances to manage content of an Activity if the main content of an Android Activity needs to change between different content that needs to be controlled differently.

7.5 Combinations of Bars

You might want to combine usages of the bars mentioned in the previous sections. This section will describe the different possible combinations of bars.

7.5.1 Combination #1

This is the most commonly used “combination” of bars (just the top bar actually). Since the top bar is required for all layouts this will be the base for the next combinations. Figure 7.4 shows an illustration of the layout using only the top bar. Please notice that the dark gray area is the top bar while the lighter gray is the actual content of the activity.

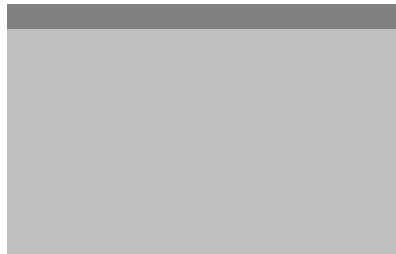


Figure 7.4: Layout example with only the top bar

7.5.2 Combination #2

If the content of the application needs to be switched out or replaced at some point, one may want to use a side bar. Figure 7.5 shows an illustration of such a layout using a top (darkest gray) bar and a side bar (dark gray). The content of the application is the light gray color.



Figure 7.5: Layout example with both a side bar and a top bar

7.5.3 Combination #3

If the content of the application needs to be switched out or replaced at some point, one may want to use a side bar. If the content that is replaced needs to be managed somehow a bottom bar can be used. Figure 7.6 shows an illustration of such a layout using a top (darker gray) bar, a side bar (dark gray), and a bottom bar (darkest gray). The content of the application is the light gray color.



Figure 7.6: Layout example with both a side bar, a top bar, and a bottom bar

7.5.4 Combination #4

If the content of the application needs to be switched out or replaced at some point, one may want to use a side bar. If the content that is replaced (and side menu) needs to be managed somehow a bottom bar can be used. Figure 7.7 shows an illustration of such a layout using a top (darker gray) bar, a side bar (dark gray), and a bottom bar (darkest gray). The content of the application is the light gray color.

Consider if this should be allowed



Figure 7.7: Layout example with both a side bar, a top bar, and a bottom bar

7.6 Clickable Elements

All elements that are clickable must have a safety-distance to other elements. This will ensure that the user does not accidentally press the wrong thing and ultimately does something wrong. This safety distance may be achieved using several different methods. Please refer to the following sections. Figure 7.8a shows an example of correct item spacing while Figure 7.8b shows an example of incorrect item spacing.

Elements must have a safety distance to ...

- Other clickable elements
- Borders of it's container
- Borders of the tablet



(a) Correct item spacing



(b) Incorrect item spacing

Figure 7.8: Examples of correct and incorrect element spacing

7.6.1 Element Margin

Elements may be spaced apart from each other using margin on the individual elements. The distance between the elements should be consistent throughout all activities of any given application. Figure 7.9 shows an example of the margin for a given element..

Note 7.6.1 If margin is used inside a container each element with margin will also be a certain distance from the borders of that specific container. If, for instance, an element has a margin of 10, then this element would be a distance of 10 from the borders of the container. This means that there *might* not be need for any padding on the given container.

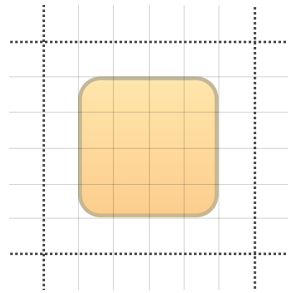


Figure 7.9: Example of element margin

Consistent Margin

Elements of the same type appearing in the same context must have the same distance to other elements. However, if the elements appear in an order, for example a horizontal list, the first and last element may differ. For instance, the first element may have a smaller left-margin and the last element may have a smaller right-margin. Figure 7.10 shows an illustration of this example.

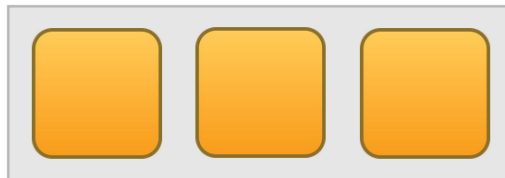


Figure 7.10: Example of element margin

7.6.2 Container Padding

Each container should provide some padding for its content. This padding should be somewhat identical to the spacing between elements inside the container. Figure 7.11 shows an example of a container with padding.



Figure 7.11: Example of a container with padding

Note 7.6.2 Whenever the content of the container can be scrolled through (for example a `GridView`) a property called `clipToPadding` must be set to `false`. Please refer to the Android documentation for additional information.


The effect of this can be explained more clearly using a figure.


8. Colors


Throughout this chapter a single color in each entry will denote a solid color, while two colors in an entry will denote a gradient between the two colors.

8.1 Text and Background

These colors should be used throughout any application.

Regular text-color	#000000	
--------------------	---------	---

Text-color used to indicate placeholder or hint-texts	#AAAAAA	
---	---------	---

Background-color for any applications window background	#000000	
---	---------	---

Background-color for any activity	#E9E9E9	
-----------------------------------	---------	---

8.2 Buttons

All buttons in the *GIRAF* software suite should use these colors for buttons. Please note that all gradients defined below are from top to bottom. Also note that the colors for the disabled button must be slightly transparent (65%).




Regular button background	#FFCD59		#FF9D00	
---------------------------	---------	---	---------	---

Regular button stroke/border	#8A6E00	
------------------------------	---------	---

Pressed button background	#D4AD2F		#FF9D00	
---------------------------	---------	---	---------	---

Pressed button stroke/border	#493700	
------------------------------	---------	---

Focused button background	#FF9D00		#FF5900	
---------------------------	---------	---	---------	---

Focused button stroke/border	#8A6E00	
Focused button background	#FAD355	
Focused button stroke/border	#E4AE4E	

8.3 Action Bar

All applications that use action bars must use the following colors. Please notice that the gradient for the topbar is from top to bottom.

Background of any action bar	#FDBB55	
	#FED76C	
Stroke/border of the action bar	#E5BE53	



8.4 Week indicators

These colors must be used whenever a certain weekday is referenced. Note that colors are primarily used to increase the usability for citizens.

Monday	#007700	
Tuesday	#800080	
Wednesday	#FF8500	
Thursday	#0000FF	
Friday	#FFDD00	
Saturday	#FF0000	
Sunday	#FFFFFF	

8.5 Page Indicator

These colors must be used for indicating which page the user is currently on.

Active page	#FF9D00	
Inactive page	#FFCD59	



Components

9	Progress Bar	37
10	Dialog	39
10.1	Confirm dialog	
10.2	Notify dialog	
10.3	Profile selector dialog	
10.4	Waiting dialog	
10.5	Custom buttons dialog	
10.6	Inflatable dialog	

9. Progress Bar

The Android framework includes a widget called `ProgressBar` which can be used both as an activity indicator (see Figure 10.4), and as an actual progress bar. Both usages of the word will henceforth be referred to as just `ProgressBar`, unless otherwise made explicit. Both are great at indicating that the application is not frozen and that something is going on. The `ProgressBar` (as a progress bar) should generally be used when there is a reliable way of calculating the actual progress on the running task. The Activity indicator should generally be used when there is no way of telling how long there will be until the task is complete.

Show examples of the different progress bars

10. Dialog

When the user has to respond to a specific event, dialogs should be used. A dialog consists of a title, a description, some buttons and possibly some additional views. In *GIRAF Components* there exists some classes for this purpose. The general layout of a dialog can be seen in Figure 10.6. For a guide of how these dialogs can be implemented see Appendix B.

10.1 Confirm dialog

When a user needs to confirm that some action is going to happen, the confirm dialog should be used as it looks in Figure 10.1.

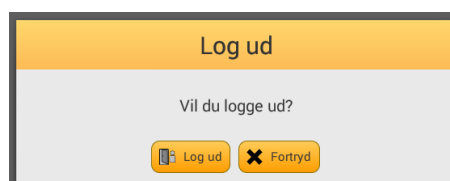


Figure 10.1: Confirm dialog

10.2 Notify dialog

When a user needs to be notified that some action has happened, the notify dialog should be used as it looks in Figure 10.2.

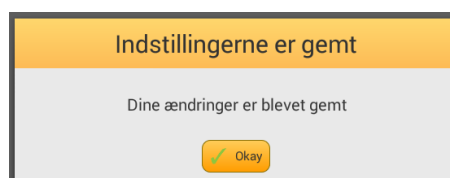


Figure 10.2: Notify dialog

10.3 Profile selector dialog

When a user needs to select a profile in some context, eg. change the current citizen on the tablet, one should use the dialog as it looks in Figure 10.3a. In other use cases where a user might need to select multiple profiles, one should use the dialog as it looks in Figure 10.3b.

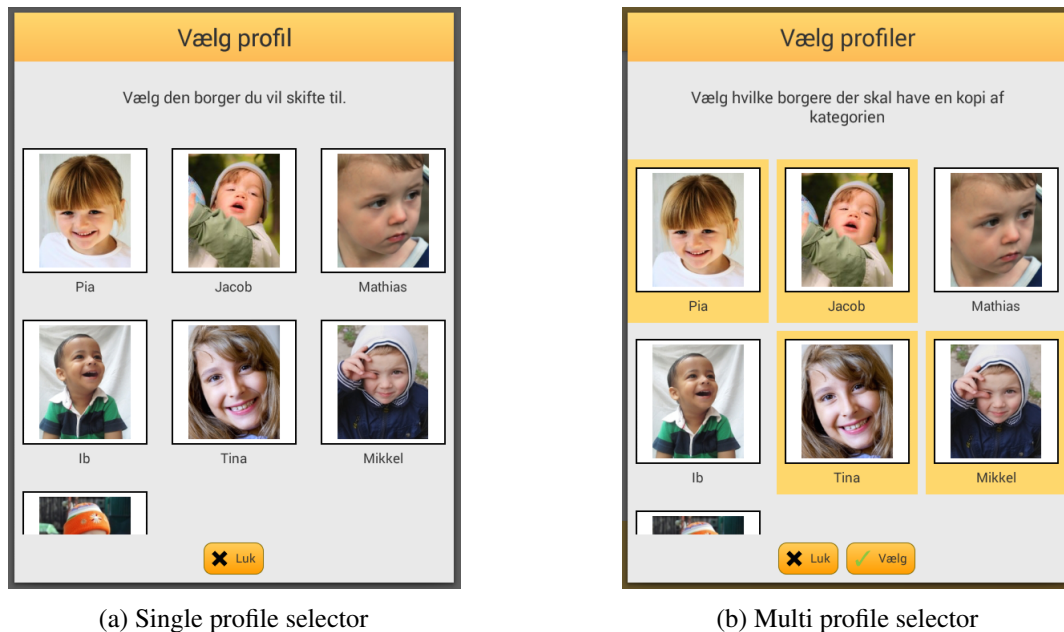


Figure 10.3: Profile selectors

10.4 Waiting dialog

In cases where the system needs to perform some action that takes a long time to execute (See Section 10.4.1), to indicate that the system is not frozen, the waiting dialog, as it looks in Figure 10.4, can be used.

Note 10.4.1 If unused screen real estate is temporarily available at the location onto which elements are currently being loaded, one should place the `ProgressBar` in this unused screen real estate. If there is not enough screen real estate available, or if it does not make sense to place the `ProgressBar` at the available location, one should use a `Dialog` with a `ProgressBar` instead. An activity indicator as the `ProgressBar` should generally be used when there is no way of telling how long there will be until the task is complete.

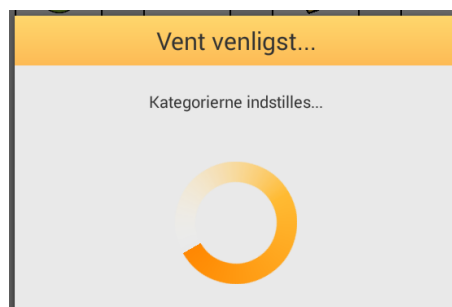


Figure 10.4: Waiting dialog

10.4.1 Long tasks

Long running tasks should generally not block the GUI. Any task that can potentially take a long time should be done on a background thread and NOT on the main GUI thread. See Appendix A for a detailed description of how to enforce this.

10.5 Custom buttons dialog

If one want to have more than two buttons in a dialog the Custom buttons dialog is the best solution as seen on Figure 10.5

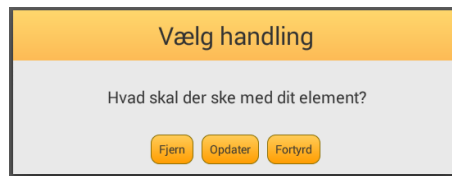


Figure 10.5: Custom buttons dialog

10.6 Inflatable dialog

Some uses of dialogs might be more specific than the ones already existing in *GIRAF Components*, for this reason the inflatable dialog exists. If one wants to add input fields or a custom view one should use this dialog. In Figure 10.6, an example of dialog is shown, this example shows the usecase when a user needs to edit a category.



Figure 10.6: Inflatable dialog

Note 10.6.1 It is important that if buttons should be added to this type of dialog it must be placed in the very bottom of the dialog and should be divided as shown in Figure 10.6. Also note that buttons should be 40dp in height.

IV

Appendix

A	Threading AsyncTask	45
A.1	Android GUI Thread	
B	How to use dialogs	47
B.1	Confirm dialog	
B.2	Notify dialog	
B.3	Profileselector dialog	
B.4	Waiting dialog	
B.5	Custom buttons dialog	
B.6	Inflatable dialog	

A. Threading AsyncTask

This chapter introduces good practice to managing the GUI thread of Android applications.

A.1 Android GUI Thread

The Android system imposes real time constraints on applications to keep them smooth and lag free. This section covers how one should make sure not to violate these constraints across all application code.

A.1.1 GUI Thread

The real time constraints are implemented on a special GUI thread which runs the user interface event loop, which is basically a queue of tasks (Java Runnable). Applications, e.g. activities, in the Android eco system run, i.e updates their views, on this GUI thread. Any updates to views not coming from the GUI thread will result in an exception. In Android, these real time constraints on the GUI thread have been implemented by simply setting a hard limit on how long, in real time, tasks on the GUI thread are allowed to run. Any violation of the constraints will result in an Application Not Responding (ANR) message to the user ¹. Users will not be able to distinguish an unresponsive application, e.g. the result of a deadlock, from a task taking too long on the GUI thread, which may cause the user to terminate an otherwise functioning application that was doing some complicated database query or downloading some file from the Internet.

A.1.2 AsyncTask

The Android framework provides a very useful abstraction which makes it easy to run long tasks in the background with a class called AsyncTask. The AsyncTask provides four non final methods that make it easy to synchronize between a background thread and the GUI thread. The three methods `onPreExecute`, `onProgressUpdate`, `onPostExecute` are always all guaranteed to run on the GUI thread. The last method, `doInBackground`, always runs on an unspecified background thread from a pool of threads maintained by the system. The class then ensures an order on these method calls being: `onPreExecute`, `doInBackground`, `onPostExecute`. The `onProgressUpdate` method can be run multiple times while the `doInBackground` method is running and is started by a call to a method called `publishProgress` from the `doInBackground` method.

A.1.3 Long Tasks

Not causing the application to show an ANR message is one thing, making the user explicitly aware that a long task is running, through some kind of feedback, is considered good practice as well ². We have tried to provide this feedback through Android ProgressBar widgets whenever

¹ Keeping Your App Responsive - <http://developer.android.com/training/articles/perf-anr.html>

² David Benyon - Designing Interactive Systems 2nd Edition

long operations take place. The name “ProgressBar” is a bit misleading since the style of the ProgressBar we use, which is the default style, is more like a spinning activity indicator as seen in Figure 10.4 on Page 40.

B. How to use dialogs

When using the various dialogs implemented in the *GIRAF Components*, it is important that one uses the support library, this is done by including the support library in the gradle build file as seen in Code Snippet B.1.

Code Snippet B.1: build.gradle

```
1 dependencies {  
2     compile ('com.android.support:support-v4:+')  
3     // Other dependencies  
4 }
```

The dialogs is an extension of the android class `DialogFragment`, meaning that all dialogs is handled as fragments. This means that callbacks from the dialogs is done using interfaces, which the activity starting them should implement. Through out the description of these dialogs there will be talked about a method called `onActionButtonClick` which is an event that is called whenever some gui element is clicked, eg. when a button is clicked. This method will in all of the following examples create an instance of the dialog and show it using the `show` method.

Also note that in all of the following code snippets there is declared a string tag (`DIALOG_TAG`) this is a tag that the Android system needs to handle fragments. However in many of the code snippets there also exist an integer (`DIALOG_ID`) which is used to call a method in the activity from the fragment.

Note B.0.1 It is important that whenever one creates a dialog one uses the `newInstance` method on the specific dialog.

Note B.0.2 Remember to call the `show` method on the dialog when the dialog should be displayed. Simply instantiating it is not enough.

Note B.0.3 It is important to check that it is the correct `dialogIdentifier` that is used in the methods implemented from interfaces in order to determine which dialog was actually responded to.

B.1 Confirm dialog

The confirm dialog is used to make the user confirm an action before it is executed. This section describes how one could implement an confirm dialog as described in Section 10.1. An example of such an implementation can be seen in Code Snippet B.2.

Using the `GirafConfrimDialog.Confirmation` interface (Line 1 in Code Snippet B.2) allows the activity (`ExampleActivity`) to implement the method `confirmDialog`, as in Line 24 in Code Snippet B.2, that will be called from the dialog whenever a response has been made for it (when the user press the acceptance button).

Code Snippet B.2: Implementaion of confirm dialog

```

1 public class ExampleActivity extends GirafActivity implements GirafConfrimDialog.Confirmation {
2
3     // Identifier for callback
4     private static final int CONFIRM_DIALOG_ID = 1;
5
6     // Fragment tag (android specific)
7     private static final String CONFIRM_DIALOG_TAG = "DIALOG_TAG";
8
9     ...
10
11    /**
12     * When some button is clicked in the gui
13     * @param view the view that was clicked
14     */
15    public void onActionButtonClick(View view) {
16        // Creates an instance of the dialog
17        GirafConfrimDialog confirmDialog = GirafConfrimDialog.newInstance("A good title", "This
            describes the dialog", CONFIRM_DIALOG_ID);
18
19        // Shows the dialog
20        confirmDialog.show(getSupportFragmentManager(), CONFIRM_DIALOG_TAG);
21    }
22
23    @Override
24    public void confirmDialog(int dialogIdentifier) {
25        if (dialogIdentifier == CONFIRM_DIALOG_ID) {
26            // Perform some action after the confirmation
27        }
28    }
29 }

```

The default behavior of buttons in the dialog is to dismiss it. The acceptance button will furthermore call the `confirmDialog` with the identifier `CONFIRM_DIALOG_ID`, as in Lines 24-28 in Code Snippet B.2.

B.2 Notify dialog

The notify dialog is used to make the user aware of some event. This section described how one could implement an notify dialog as described in Section 10.2. An example of such an implementation can be seen in Code Snippet B.3.

Using the `GirafNotify.Notification` interface (Line 1 in Code Snippet B.2) allows the activity (`ExampleActivity`) to implement the method `noticeDialog`, as in Line 24 in Code Snippet B.3, that will be called from the dialog whenever a response has been made for it (when the user press the button).

Code Snippet B.3: Implementaion of notify dialog

```

1 public class ExampleActivity extends GirafActivity implements GirafNotifyDialog.Notification {
2
3     // Identifier for callback
4     private static final int NOTIFY_DIALOG_ID = 1;
5
6     // Fragment tag (android specific)
7     private static final String NOTIFY_DIALOG_TAG = "DIALOG_TAG";
8

```



```

9      ...
10
11     /**
12      * When some button is clicked in the gui
13      * @param view the view that was clicked
14      */
15     public void onActionButtonClick(View view) {
16         // Creates an instance of the dialog
17         GirafNotifyDialog notifyDialog = GirafNotifyDialog.newInstance("A good title", "This
18             describes the dialog", NOTIFY_DIALOG_ID);
19
20         // Shows the dialog
21         notifyDialog.show(getSupportFragmentManager(), NOTIFY_DIALOG_TAG);
22     }
23
24     @Override
25     public void noticeDialog(int dialogIdentifier) {
26         if(dialogIdentifier == NOTIFY_DIALOG_ID) {
27             // Perform some action after the notification
28         }
29     }

```

The default behavior the button in the dialog is to dismiss it. The button will furthermore call the `confirmDialog` with the identifier `NOTIFY_DIALOG_ID`, as in Lines 24-28 in Code Snippet B.2.

B.3 Profileselector dialog

The profile selector dialog is used to allow for user to select one ore more profiles. This section describes how one could implement an confirm dialog as described in Section 10.3. Examples of such an implementation can be seen in Code Snippet B.4 and Code Snippet B.5.

This dialog is used in two use cases. One where the user wants to select exactly one profiles, and another use case where one wants to select arbitrary many profiles. Using the `GirafProfileSelectorDialog.OnSingleProfileSelectedListener` interface allows for the activity (ExampleActivity) to respond on a single selected profile as in Line 1 in Code Snippet B.4. Whereas the `GirafProfileSelectorDialog.OnMultipleProfilesSelectedListener` allows for the activity to respond on multi selected profiles an in Line 1 in Code Snippet B.5.

Note B.3.1 Observe that the main difference between the instantiating of the single and multi version of the profile selector dialog is the fourth argument called `selectMultipleProfiles`. That determines which interface will be called. If this boolean is set to `true` the activity should implement `GirafProfileSelectorDialog.OnMultiProfileSelectedListener` otherwise the activity should implement `GirafProfileSelectorDialog.OnSingleProfileSelectedListener`. Obviously if an activity have one dialog of each type both interfaces should be implemented.

Code Snippet B.4: Implementaion of single profile selector dialog

```

1 public class ExampleActivity extends GirafActivity implements
2     GirafProfileSelectorDialog.OnSingleProfileSelectedListener {
3
4     // Identifier for callback
5     private static final int PROFILE_SELECT_DIALOG_ID = 1;
6
7     // Fragment tag (android specific)
8     private static final String PROFILE_SELECT_DIALOG_TAG = "DIALOG_TAG";
9
10    ...
11
12    /**
13     * When some button is clicked in the gui

```

```

13  * @param view the view that was clicked
14  */
15  public void onActionButtonClick(View view) {
16      // Creates an instance of the dialog
17      GirafProfileSelectorDialog singleSelectDialog =
18          GirafProfileSelectorDialog.newInstance(this, getGuardianIdentifier(), false, false, "Select
19          a profile for some purpose", PROFILE_SELECT_DIALOG_ID);
20
21      // Shows the dialog
22      singleSelectDialog.show(getSupportFragmentManager(), PROFILE_SELECT_DIALOG_TAG);
23  }
24
25  @Override
26  public void onProfileSelected(int dialogIdentifier, Profile profile) {
27      if(dialogIdentifier == PROFILE_SELECT_DIALOG_ID) {
28          // Perform some action based on the profile selected
29      }
30  }
31
32  private long getGuardianIdentifier() {
33      // Some code that returns the wanted guardian identifier
34  }

```

When a profile is clicked in a single profile selector the `onProfileSelected` method is called, with the profile that was selected and the identifier `PROFILE_SELECT_DIALOG_ID` as in Lines 24-28 in Code Snippet B.4.

Code Snippet B.5: Implementaion of multi profile selector dialog

```

1  public class ExampleActivity extends GirafActivity implements
2      GirafProfileSelectorDialog.OnMultipleProfilesSelectedListener {
3
4      // Identifier for callback
5      private static final int PROFILE_SELECT_DIALOG_ID = 1;
6
7      // Fragment tag (android specific)
8      private static final String PROFILE_SELECT_DIALOG_TAG = "DIALOG_TAG";
9
10     ...
11
12     /**
13      * When some button is clicked in the gui
14      * @param view the view that was clicked
15      */
16     public void onActionButtonClick(View view) {
17         // Creates an instance of the dialog
18         GirafProfileSelectorDialog multiSelectDialog =
19             GirafProfileSelectorDialog.newInstance(this, getGuardianIdentifier(), false, true, "Select
20             some profiles for some purpose", PROFILE_SELECT_DIALOG_ID);
21
22         // Shows the dialog
23         multiSelectDialog.show(getSupportFragmentManager(), PROFILE_SELECT_DIALOG_TAG);
24     }
25
26     @Override
27     public void onProfilesSelected(int dialogIdentifier, List<Pair<Profile, Boolean>>
28         checkedProfileList) {
29         if(dialogIdentifier == PROFILE_SELECT_DIALOG_ID) {
30             // Perform some action based on the list of profiles and check status
31         }
32     }
33
34     private long getGuardianIdentifier() {
35         // Some code that returns the wanted guardian identifier
36     }
37 }

```

When a profile is clicked in a multi profile selector the `onProfilesSelected` method is called, with a list of profiles and a boolean telling of they were marked or not alongside the identifier `PROFILE_SELECT_DIALOG_ID` as in Lines 24-28 in Code Snippet B.5.

B.4 Waiting dialog

The waiting dialog is used to indicate to the user that some task is being executed and that he/she should wait for that task to end. This section describes how one could implement an waiting dialog as described in Section 10.4.

Note B.4.1 Note that in this dialog the dialog is instantiated in the `onCreate` method (see Line 40 in Code Snippet B.6), and is called showed implicitly by the `onActionButtonClick` method which starts the async task (see Line 50 in Code Snippet B.6).

This dialog is often used when some thread syncing tasks is running (see Appendix A). This dialog requires not interface to work properly. Good practice with this dialog is to show it before the long task is executed. This can be done using the `onPreExecute` method (see Line 16 in Code Snippet B.6). One should dismiss the dialog after the task is done which can be handled in `onPostExecute` method (see Line 29 in Code Snippet B.6).

Code Snippet B.6: Implementaion of waiting dialog

```

1 public class ExampleActivity extends GirafActivity {
2
3     private static final String WAITING_DIALOG_TAG = "DIALOG_TAG"; // Fragment tag (android
        specific)
4     private GirafWaitingDialog waitingDialog;
5
6     /**
7      * An async task that performs a task that takes time
8      */
9     private class ExampleTask extends AsyncTask<Void, Void, Void> {
10
11         @Override
12         protected void onPreExecute() {
13             super.onPreExecute();
14
15             // Shows the dialog
16             waitingDialog.show(getSupportFragmentManager(), WAITING_DIALOG_TAG);
17         }
18
19         @Override
20         protected Void doInBackground(Void... params) {
21             // Perform the task the user needs to wait for
22         }
23
24         @Override
25         protected void onPostExecute(Void aVoid) {
26             super.onPostExecute(aVoid);
27
28             // Hides the dialog
29             waitingDialog.dismiss();
30         }
31     }
32
33     ...
34
35     @Override
36     protected void onCreate(Bundle savedInstanceState) {
37         super.onCreate(savedInstanceState);
38
39         // Initialize the waiting dialog
40         waitingDialog = GirafWaitingDialog.newInstance("Please wait", "We are currently performing
        a task that takes time");

```

```

41     }
42
43     /**
44      * When some button is clicked in the gui
45      * @param view the view that was clicked
46      */
47     public void onActionButtonClick(View view) {
48
49         // Starts a tasks
50         new ExampleTask().execute();
51     }
52
53 }

```

B.5 Custom buttons dialog

The custom buttons dialog is used when the developer wants to provide the user with more than two buttons in the dialog. This section described how one could implement a custom buttons dialog as described in Section 10.5.

Using the `GirafCustomButtonsDialog.CustomButton` (Line 1 in Code Snippet B.7) interface allows for the developer to add arbitrary many buttons to the dialog.

Note B.5.1 There is no limit on buttons that can be added to the dialog, but one should be careful that the design is consistent. The width should not exceed the width of other dialogs.

Code Snippet B.7: Implementation of custom buttons dialog

```

1 public class ExampleActivity extends GirafActivity implements
   GirafCustomButtonsDialog.CustomButton {
2
3     // Identifier for callback
4     private static final Integer CUSTOM_BUTTONS_DIALOG_ID = 1;
5
6     // Fragment tag (android specific)
7     private static final String CUSTOM_BUTTONS_DIALOG_TAG = "DIALOG_TAG";
8
9     ...
10
11     /**
12      * When some button is clicked in the gui
13      * @param view the view that was clicked
14      */
15     public void onActionButtonClick(View view) {
16         // Creates an instance of the dialog
17         GirafCustomButtonsDialog customButtonsDialog = GirafCustomButtonsDialog.newInstance("Pick
           an action","Choose what should happen to the element", CUSTOM_BUTTONS_DIALOG_ID);
18
19         // Show the dialog
20         customButtonsDialog.show(getSupportFragmentManager(),CUSTOM_BUTTONS_DIALOG_TAG);
21     }
22
23     @Override
24     public void fillButtonContainer(int dialogIdentifier, GirafCustomButtonsDialog.ButtonContainer
       buttonContainer) {
25         if(dialogIdentifier == CUSTOM_BUTTONS_DIALOG_ID) {
26             GirafButton renameButton = new GirafButton(this,"Rename");
27             GirafButton copyButton = new GirafButton(this,"Copy");
28             GirafButton deleteButton = new GirafButton(this,"Delete");
29
30             // Set onClickListeners to the buttons
31
32             buttonContainer.addGirafButton(renameButton);
33             buttonContainer.addGirafButton(copyButton);

```

```

34         buttonContainer.addGirafButton(deleteButton);
35     }
36 }
37 }

```

Before the dialog is created the `fillButtonContainer` method is called, see Lines 24-36 in Code Snippet B.7. The identifier `CUSTOM_BUTTONS_DIALOG_ID` determines which dialog is being filled with buttons.

B.6 Inflatable dialog

The inflatable dialog is used to create more custom dialogs and want some custom gui elements to be shown in the dialog. In this example we have created a custom dialog that contains an `AnalogClock` element and a button to dismiss the dialog.

When creating an instance of this type of dialog the `newInstance` method takes an layout resource as third argument. The argument `R.layout.example` as seen in Line 17 in Code Snippet B.8 comes from the layout seen in Code Snippet B.9.

Code Snippet B.8: The implementation of the inflatable dialog

```

1  public class ExampleActivity extends GirafActivity implements
    GirafInflatableDialog.OnCustomViewCreatedListener {
2
3      // Identifier for callback
4      private static final Integer CLOCK_DIALOG_ID = 1;
5
6      // Fragment tag (android specific)
7      private static final String CLOCK_DIALOG_TAG = "DIALOG_TAG";
8
9      ...
10
11     /**
12      * When some button is clicked in the gui
13      * @param view the view that was clicked
14      */
15     public void onActionButtonClick(View view) {
16         // Creates an instance of the dialog
17         GirafInflatableDialog clockDialog = GirafInflatableDialog.newInstance("Time", "Here is the
            time on an analog clock", R.layout.example_layout, CLOCK_DIALOG_ID);
18
19         // Show the dialog
20         clockDialog.show(getSupportFragmentManager(),CLOCK_DIALOG_TAG);
21     }
22
23     @Override
24     public void editCustomView(ViewGroup customView, int dialogIdentifier) {
25         if(dialogIdentifier == CLOCK_DIALOG_ID) {
26
27             // Find the close button defined in xml
28             GirafButton closeButton = (GirafButton) customView.findViewById(R.id.close_button);
29
30             // Do something with the close button eg. set an onClickListener
31
32         }
33     }
34 }

```

Using the `GirafInflatableDialog.OnCustomViewCreatedListener` interface (Line 1 in Code Snippet B.8) allows for the developer to access the custom created layout by the `editCustomView` method. See Lines 24-33 in Code Snippet B.8 for an example of how one access the custom view.

Note B.6.1 One cannot access the custom inflated view before the `editCustomView` is called from the dialog it self.

Code Snippet B.9: The custom layout for an inflateable dialog

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:orientation="vertical">
7
8     <!-- An analog clock to be shown in the dialog -->
9     <AnalogClock
10         android:id="@+id/analogClock"
11         android:layout_width="match_parent"
12         android:layout_height="100dp"
13         android:layout_gravity="center" />
14
15     <!-- The container for buttons -->
16     <LinearLayout
17         android:layout_width="match_parent"
18         android:layout_height="match_parent"
19         android:orientation="horizontal">
20
21         <!-- A button to close the dialog -->
22         <dk.aau.cs.giraf.gui.GirafButton
23             android:id="@+id/close_button"
24             android:layout_width="wrap_content"
25             android:layout_height="40dp"
26             app:icon="@drawable/icon_cancel"
27             app:text="Close clock dialog" />
28
29     </LinearLayout>
30
31 </LinearLayout>
```



Index

Index

GIRAF Software Suite, 9

Action bar, 27, 34

Activity indicator, 40

Arrow icons, 17

ASyncTask, 45

Back button, 27

Background task, 40

Bars, 28

- Bottom bar, 29, 30

- Progress Bar, 37

- Side bar, 29, 30

- Top bar, 29, 30

Bottom bar, 28–30

Box model, 11

Button, 27

- Back button, 27

- Help buttons, 27

Camera icons, 16

Colors, 33

- Action bar, 34

- Background color, 33

- Button colors, 33

- Page indicator, 34

- Text color, 33

- Week indicators, 34

Confirm dialog, 39

Context, 9

Contextual, 9

Custom buttons dialog, 41

Data management icons, 18

Dialog, 39–41

- Confirm dialog, 39

- Custom buttons dialog, 41

- Inflatable dialog, 41

- Notify dialog, 39

- Profile selector dialog, 39

Waiting dialog, 40

Editable, 22

Estate, 9

Font color, 23

Font size, 23

GUI thread, 45

Help button, 27

Icons, 15

- Arrows, 17

- Camera, 16

- Data management, 18

- Media, 17

- Microphone, 16

- Miscellaneous, 19

- User management, 16

- Utility, 18

- Version control, 17

Image, 21

Indicator, 22

- Custom indicator, 22

- Editable indicator, 22

Inflatable dialog, 41

Layout, 29, 30

Layout content, 28

Long tasks, 40

Margin, 11, 30, 31

Marking, 22

Media icons, 17

Microphone icons, 16

Miscellaneous icons, 19

Notify dialog, 39

Overlay, 22

- Custom indicator, 22
- Editable indicator, 22
- Padding, 11, 31
- Page indicator, 34
- Pictogram, 21
- Profile selector dialog, 39
- Progress Bar, 37
- Screen Real Estate, 9
- Selection, 22
- Side bar, 27, 29, 30
- Terms, 9
- Text, 23
- Thread, 45
 - GUI thread, 45
- Tone of voice, 25
- Top bar, 27, 29, 30
- Typography, 23
- User management icons, 16
- Utility icons, 18
- Version control icons, 17
- Voice, 25
- Waiting dialog, 40
- Week indicators, 34