F O R U M   N O K I A

Flash Lite UI Design Guide for Keypad and Touch Devices

Version 1.0; December 9, 2008

Contents

Change history

December 9, 2008	Version 1.0	Initial document release

1 Introduction

Until recently, developers and designers working with Flash Lite from Adobe have been designing applications for different resolutions and layouts without considering touch interaction. With the introduction of touch devices, this situation has changed, and touch interaction has become an important factor in designing good applications that will work with a variety of screens and interactions.

Because this is a new development, there are currently no available resources addressing Flash Lite. This document provides a basic resource for Flash Lite developers and designers, describing how to design a Flash Lite application UI with dynamic layout control for multiple screen resolutions and inputs (key, touch, and, in the future, key/touch devices).

The document approaches the topic from developer and designer perspectives, offering an overview of present-day knowledge related to Flash Lite UI design, as well as discussing new, arising issues. Instead of addressing a specific platform, it looks at Flash Lite-enabled Nokia devices. The main goal will be to maintain a good user experience across a variety of devices, focusing on key- and touch-based input. The ideal use of this document will be as an accompaniment to *Sudokumaster: Designing a Flash Lite Game for Keypad and Touch Devices* [1], which offers a practical application of the information provided here.

Many general conventions in this document can be used as guidelines. They neither prevent a developer from creating innovative designs nor offer the only correct approach to a particular problem.

2 Before beginning

2.1 UI design and functionality

Usability is a basic user need and should be a priority in the design process. Applications should function in such a way that users can learn to use the product quickly, perform basic tasks smoothly, and have a pleasant experience with all input methods available.

Coherence and consistency in design should be maintained throughout the entire UI, not just for the sake of usability; it also brings clarity to the development process. Specifically, when an application is run in the S60 platform, it is important that the application reacts well to changes caused by the user and system (such as orientation changes).

Unless there are other requirements, platform selection and thus the architecture will depend highly on the UI design and functionality of the application. Therefore, it is good practice to start your work by designing the application UI and defining its functionalities. These need not be polished diagrams or visuals, but sufficiently representative that they can help you consider the UI and application functionalities.

Briefly, the following general issues should be considered when designing your application:

- Main use cases
  - What are the common use cases for your application?
  - How could use cases be prioritised to suit your goals?
  - What kinds of functionalities do these use cases involve/require?
    - Can these functionalities be segmented?
  - How do these use cases and functionalities affect the UI?
    - Is implementation possible with Flash Lite?
  - Is it possible to separate the UI into components?
- Wireframes and other UI diagrams for states, views, navigation, and inputs methods for the application (in draft level).

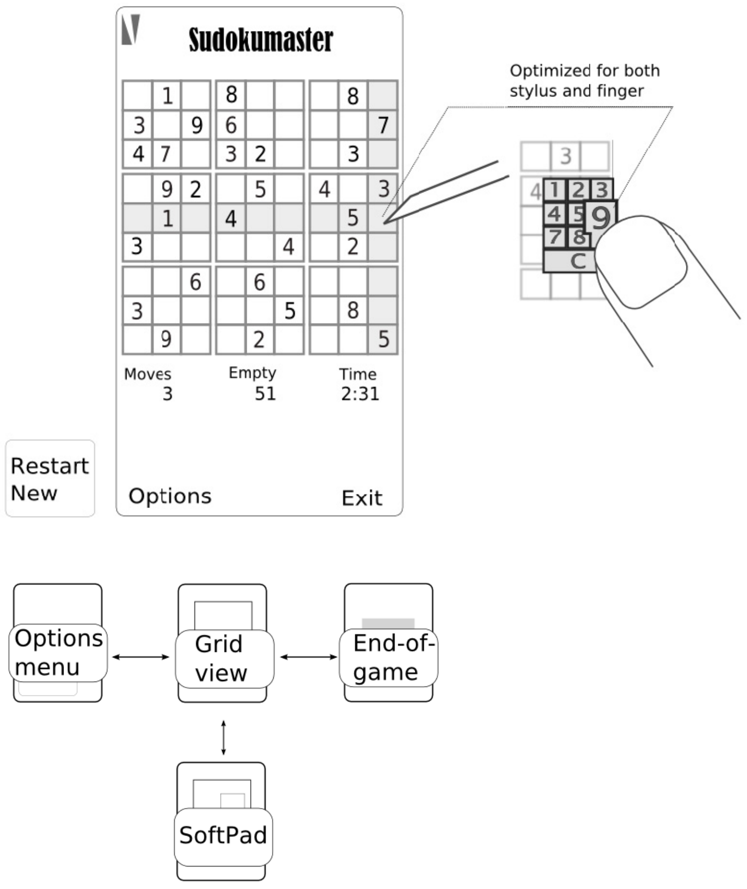


Figure 1: Application UI diagrams

2.2 Platform selection

Many application developers wish to support as many devices as possible, which would seem to call for the use of Flash Lite 1.1. However, consider the drawbacks of Flash Lite 1.1 with regard to touch screen devices.

On the one hand, Flash Lite 1.1 allows you to create optimised applications. However, since it requires heavy use of buttons (not encouraged) and extra implementation, it does not offer the best option for our purposes. Therefore, Flash Lite 2.x or newer is recommended.

Below are several reasons supporting this conclusion.

- The exact touch position outside a button cannot be located with Flash Lite 1.x.
- Drag and drop cannot be implemented with Flash Lite 1.1.
- Buttons cannot react to a touch event, due to the missing `trackAsMenu` property for buttons with Flash Lite 1.x.
- The Flash Lite 1.x player does not support inline text input or dynamic loading of external multimedia files.

The `System.capabilities.hasStylus` function is not supported, which is vital for detecting if a device has touch screen capability and offering a customised experience for that device profile.

After defining the target platform, go to the target device set and check the platform (Series 40 or S60 platform), resolution, colour depth, Platform Services capabilities, and support for different `fscommand2` calls you may want to use. This will help you see the overall picture better and enable you to group and define your target groups more easily. Adobe Device Central (CS3 or CS4) is a great resource, providing a searchable library of device profiles that enables developers to quickly determine the features supported on various Series 40 or S60 devices.

To have separate functionalities or UI elements for different platforms, use

```
statusplatform = fscommand2("GetPlatform", "platform");
```

which sets a parameter according to the current platform. For more information, go to the Adobe Web site.

3 Layout

When structuring content and presenting information, keep an eye on the balance; make sure there is not too much information on the screen and that different elements are grouped properly. For instance, navigation can become cumbersome if there are too many interactive objects on the screen, such as links or buttons. Reduce and group them using existing conventions, like collapsing/expanding links or menus.

S60 touch has compatibility mode enabled by default. It can be disabled by using `fscommand2()` with `DisableKeypadCompatibilityMode` command. See the *Flash Lite Developer's Library* [2] for more information on compatibility mode.

3.1 Dimensions

On devices with a low pixel-per-inch ratio, very small objects can be difficult to distinguish. In particular, if the device is a pure touch device, distinguishing the elements as well as interaction would be a serious issue.

For touch screens, the size of the hit areas should be large enough for finger and thumb input. For instance, S60 UI style defines the target minimum size for a UI element as the following:

- 7x7 mm with 1x1 mm gap for index finger;
- 8x8 mm with 2x2 mm gap for thumb;
- List type of components with a minimum of 5 mm line spacing.

Unless you are designing separate user interfaces for each device profile (key, touch, and key and touch), these measures should apply as the least common denominator.

Font size is also important for presenting readable content to users. Recommended font sizes for different screens are listed below. Use these numbers as a reference guide and decide how big or small the font sizes should be for unlisted resolutions.

128 x 160 pixel display

Softkeys: 12 (LSK and RSK), 16 (MSK)

Small text: 9 - 12

Lists and menus: 16 - 23

240 x 320 pixel display

Softkeys: 20 (LSK and RSK), 24 (MSK)

Small text: 20

Medium to large text: 24 - 48

Lists and menus: 24

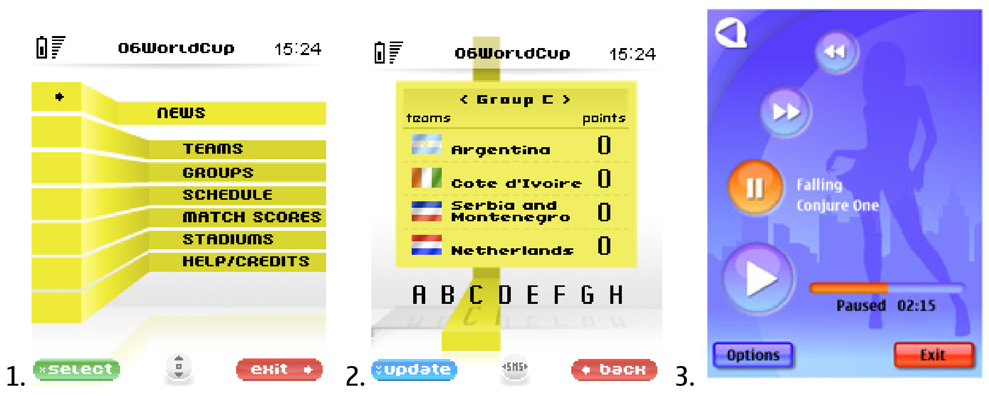


Figure 2: Examples of good information structure and appealing interface

Examples 1 and 2 above (courtesy of Gideon Multimedia) have good information structure and an appealing interface. However, since the application was not designed with touch screen use in mind, the input elements are too small for easy use. Example 3 shows an interface with clearly separated UI elements that are suitable for both key and touch input.

3.2 Orientation and rotation

Layout changes are vital for different orientations. When the orientation mode of the device is detected or changes, the layout should be restructured and resized.

The exact orientation of the device can be discovered by querying `Stage.width` and softkey locations with the following command:

```
status = fscommand2("GetSoftKeyLocation");
```

where the return values are:

```
-1: Not supported.
0: Soft keys on top.
1: Soft keys on left.
2: Soft keys on bottom.
3: Soft keys on right.
```

For orientation changes that might happen during application runtime, a listener should be created that listens, determines, and takes the application to the appropriate layout.

```
var stageListener:Object = new Object();
stageListener.onResize = function() {
   // Detect stage width
   // Detect softkey locations
   // Decide the right layout
};
```

3.3 Reorganisation

Because display dimensions between devices have different aspect ratios and resolutions, the same layout may not necessarily work for all, and the layout must be considered separately for certain displays by grouping, reorganising, and limiting the number of objects.

While there may be numerous methodologies for layout reorganisation, creating layouts for different screen sizes and orientations can be a practical and fast solution. In order to apply this approach, the software should be designed in a component structure, where each component can be resized, relocated, and have different layouts for different orientations.

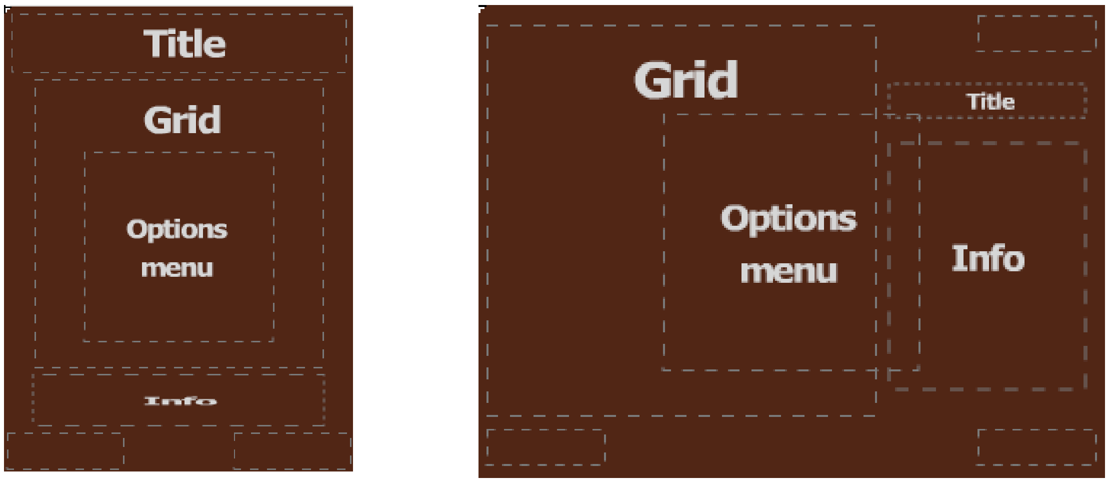


Figure 3: Sample layouts representing different component positions and sizes on different screen orientations

The sample layouts shown in Figure 3 represent different component positions and sizes on different screen orientations. Depending on the situation, components can either be resized and relocated, or have different modes (portrait/landscape) and be placed in the right location.

In any case, key and touch interaction should be considered, and reorganisation should be optimised, to enable both key and touch interaction, by hiding, zooming, or using other convenient techniques.

4 Scalability

4.1 Resolution

The resolution of a device is a critical factor in presenting the right UI to the user. Depending on the resolution, the content (and its components) should be resized and presented in the proper layout.

Concerning the resolution, first prioritise the resolutions according to the application's nature, the target user, and the device profiles. Prioritisation will help you see the most common use cases and decide the original size of your components. Currently, since most devices have QVGA screens, 240 x 320 pixels might dominate the top of the lists, whereas low, 1:1 or rare screens will appear at the bottom.

Table 1 shows standard display resolutions and aspect ratios (in portrait mode):

	Display dimensions (in pixels)	Aspect ratio (width : height)	Example devices
QVGA	240 x 320	3:4	Nokia N95, N71, N73, N93, E50, 6300
Double resolution	352 x 416	11:13	Nokia N80, E60, E70
Square	208 x 208 128 x 128 (Series 40)	1:1	Nokia 5500
Low resolution	176 x 208 128 x 160 (Series 40)	11:13	Nokia 3250, N91
High resolution	360 x 640	9:16	Nokia 5800
Custom screen	352 x 800	11:25	Nokia E90*

Table 1: Standard display resolutions and aspect ratios (in portrait mode)

In general, the long-term development trend is towards higher resolutions. For more detailed information about standard and custom screen sizes and aspect ratios, see the *Design and User Experience Library* [3].

Knowing the dominant resolution also helps to define the initial size of components. Although Flash Lite offers quite good scalability, scaling images and fonts up or down might cause problems, especially on players that do not support anti-aliasing or interfaces using pixel fonts. Therefore, if the dominant resolution for your application is 240 x 320 pixels, it could be a good practice to create your original UI for this size, unless you choose to create separate components for separate resolutions and not use scaling at all.

5 Navigation and input

For usability, navigation should be efficient and effortless by providing easy access to options, minimising the number of required inputs, using shortcuts to frequently used information and input data, or fast ways to return to a previous state.

Navigation of the application should ideally support both key and touch interaction, where they could be limited or enabled depending on the device capabilities. Although using button events may be convenient in some cases, it should be avoided, since they might not work as expected on mobile devices.

Various input methods can be applied. In general, avoid default input fields because the user experience cannot be guaranteed on different devices. If possible, custom input methods are preferable, leaving the default way as the last option.

5.1 Input detection

Currently devices have either key or touch input methods. In the future, however, devices might support both key and touch inputs. Therefore, a good application should be designed with components that support both. However, there may be cases where the UI should be limited or customised for one or the other. In this case, detecting device input is vital, in order to offer the right UI for the right device.

Currently there are various ways to detect whether a device has only key or only touch input, but not a dedicated method (yet). The following code represents one way to determine the input method of the device.

```
if(System.capabilities.hasStylus) // Has stylus capability?
{
    // Touch or Touch & Key device
}
else
{
    // Key device
}
```

5.2 Finger, stylus, and keys

Implementing touch events in Flash Lite is similar to mouse or button events in Flash. A touch action can be received by a button or a treated as a mouse event. Various touch gestures (tap, tap and hold, slide, drag) can be detected by using existing mouse handlers (`onMouseDown`, `onMouseUp`, `onMouseMove`) or button handlers (`onRelease`, `onPress`, `onRollOver`, `onRollOut`).

Although using buttons is not encouraged, in some cases it might be more practical, since buttons work both with keys and touch. If mouse events are used to detect touch events, key navigation should be implemented separately.

Finger and stylus differ in size of interaction. While a stylus can interact with small objects, a finger cannot. Therefore, it is preferable to consider using the finger as the default way of interacting and to arrange the navigation and input elements accordingly.

Focusing has different implementations with key and touch. For key interaction, focusing is usually the first thing accomplished, and then the user selects an element after that he or she desires. With touch-based interaction, it is possible to select an element with direct selection (without focusing), focus on tap and select on release, or focus on first touch and select on second. Although the choice of behaviour strongly depends on the case, direct selection and separate focus/selection should be avoided because the former is highly prone to mistakes, whereas latter requires an extra step.

5.3 Two-hand control

There are various cases where users have to use one hand to hold the device and the other for input. A good design should take both one- and two-handed navigation and input into consideration. Since single-hand control means using a thumb, make sure your application performs basic tasks with one hand only.

The following are examples of cases where a user needs both hands:

- Applications that require use of stylus;
- Applications intended for landscape mode;
- Applications that require data input.

Don't confuse two-hand control with simultaneous touch. Currently Nokia devices do not support multi-touch capability, therefore it is beyond the scope of this document.

5.4 Text input

Since Flash Lite 1.1, text input has been an important concern for UI designers. Although user experience has improved from Flash Lite 1.1 to 3.x, it is still not the best. While some developers may prefer to use default text input methods, others may decide to create custom ways to improve the user experience.

Text input optimised for both key and touch devices is an even bigger concern. While users use a keypad to input text on key-only devices, the default input text field triggers a virtual keyboard on touch devices. Visually, this results in a weaker user experience, which may even become acute (for instance, when you would like to display only a number pad or limit some character entry). Furthermore, since Flash Lite native text input fields have to be focused first on key-based devices, application design should consider focus handling as well.

Depending on the nature of your application, try to choose the best method for text input by balancing user experience and implementation efforts. For instance, if the only text input in your application is numbers, it is relatively easy to create a custom number entry pad compared to a full QWERTY virtual keyboard.

5.5 General strategies

5.5.1 Focus and select

In general, the main ways to interact with touch devices are direct selection (without focusing), focus on tap and select on release, or focus on first touch and select on second. As mentioned previously, navigation and input methods should be decided based on the nature of your application and the ease of compatibility between key- and touch-based devices.

Focus is also an important issue to consider if using buttons.

5.5.2 Drag and drop

On touch devices, drag and drop can be a very handy method of interaction. On key-based devices, one can focus, select, move, and drop an object, trying to emulate drag and drop functionality. However, this does not result in a good user experience, and it requires extra implementation. Therefore, drag and drop should be avoided on optimised applications.

5.5.3 Buttons

If you decide to use buttons for your application, keep in mind that, for both key and touch devices, RollOver handler is notified when a button gets focus and RollOut handler is notified when a button loses focus. On touch devices, a button is focused on Press or Release; it loses the focus when a tap occurs out of the button hit area. Press and Release events can be used to detect long presses; however, since that wouldn't be possible to implement on key devices, it should be avoided.

In general, button use is discouraged, since there are many performance-based concerns, focus difficulties (most devices do not support button wraparound), and unresolved issues, as mentioned on the Adobe Web site.

6 Feedback

With touch devices in particular, it should be obvious to users when their input is acceptable and the operation successful. Mostly it is sufficient to address error situations only, and not bother the user with too many confirmation messages. If, for example, an operation takes a significantly long time to complete, this should be indicated with a waiting note.

6.1 Audio feedback

It can be a good idea to offer audio feedback to the user, especially when it comes to touch devices. However, using audio is a sensitive issue, not just for user experience concerns, but also with regard to the different performance values of various devices. Depending on the device your application runs on, audio may cause a delay, or may not even work. Therefore, make sure you know the target platform and devices extremely well, to ensure that the experience is the same for most, if not all users.

6.2 Physical feedback

Physical feedback using vibration is possible with Flash Lite. Tactile feedback in the touch screen can be used to indicate a successful action, for example, when the user presses a button on the touch screen.

Note that by default some touch screen devices vibrate to offer feedback to a particular user interaction. Developers can ensure a recognisable user experience by implementing such behaviour for all devices. However, make sure that physical feedback is not exaggerated and is used only if convenient. Consider the following issues:

- Tactile feedback increases power consumption; excessive use will drain the battery.
- If tactile feedback is used for every possible UI event, the device will be vibrating all the time and the vibration will be meaningless. Additionally, continuous tactile feedback may become irritating.
- Tactile feedback should also be supported by a change in the visual style of the elements, especially in case of buttons.

6.3 Visual feedback

It should be obvious to users when their input is acceptable and the operation successful. Mostly it is sufficient to address error situations only and not bother the user with too many confirmation messages. If, for example, an operation takes a significantly long time to complete, this should be indicated with a waiting note.

6.4 Feedback delay

Natural latency for tactile feedback is 10 to 20 ms. If latency is more than 20 ms, users will start to notice the delay. When using vibration, sequences should be short in order to keep the feedback pleasant. Avoid sequences longer than 50 ms.

If audio feedback is used, vibration sequences and audio should be synchronised. If there is clear latency in tactile feedback, consider delaying possible visual or audio feedback as well. The different modalities should complement each other.

7 Visual guidance

Focus is an important matter for any user interface. Without clear focus indications, users may very well be distracted, misled, or unable to use the application. When focus is clear, the user understands instantly how to act.

For instance, a small arrow can indicate movement between screen/section in a horizontal or vertical direction and the current active screen/section. Try emphasising the scenario visually by colouring, positioning, and sizing the elements to indicate the most important element and the related action on the current screen. For instance, make sure that the buttons can be understood as buttons, that is, make them look clickable.

For Flash Lite, the default behaviour of a button's (if used) focus should be always overridden for a better user experience by adding

```
_focusrect = false;
```

to a frame at the beginning of the movie and customising the button's Up, Over, and Down states and Hit area.

If there are input selection components, the default item should be preselected on the screen using

```
 setFocus(<instanceName>);
```

This will avoid extra interaction requirements and user distraction. Use Adobe Device Central to test how the UI will look in different circumstances (changing backlight, gamma, contrast, and reflection) and to see if your focus indications are sufficient for good usability.

8 Standards

If you are designing an application with Flash Lite, you may decide to be free from any standards and create your own unique experience. However, even if that is the case, it is good practice to consider platform conventions to avoid confusion and a steep learning curve for users.

For instance, an argument can be made whether using softkeys is a necessity for a pure touch device. If the aim is to create optimised content across devices with different input, a good design should implement general guidelines in functionality, if not visually.

Here are some general rules for softkey layouts:

- Softkey labels should be visible and named distinctly from each other;
- Options for exiting and moving backwards should be behind the right softkey;
- Menus and other kinds of options should behind the left softkey;
- The maximum length for a label is about 35 percent from the left/right margin;
- When orientation changes, the softkey labels should be relocated (see the following figures);
- Series 40 devices might have a middle softkey (see the leftmost example in Figure 4);
- With touch screens, softkeys should be implemented so that they function on touch.

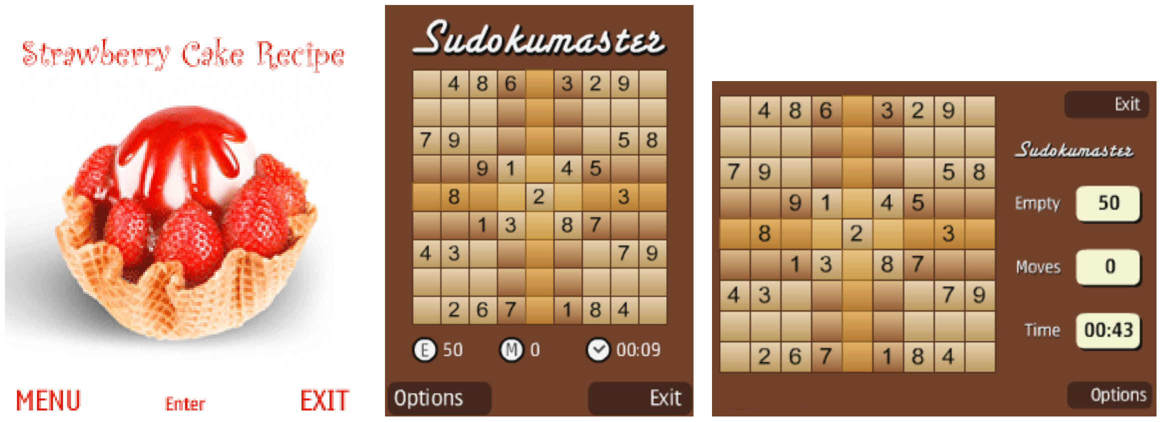


Figure 4: Different softkey layouts

9 Other issues

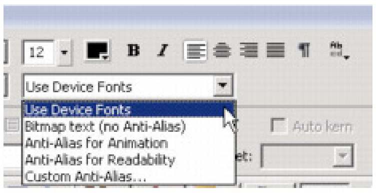
Although an ideal application design avoids scaling, in real life it is not possible to achieve that goal easily. Flash Lite handles scaling well, but there are issues to consider, especially on fonts and bitmaps to be used in your application.

9.1 Fonts

9.1.1 Pixel fonts

Using normal fonts might offer a good user experience for large screens, but when fonts are scaled down for smaller resolutions, they may not appear as expected. Therefore, consider using pixel fonts instead of anti-aliased fonts to avoid problems and provide a clean look. If using pixel fonts, remember that they also have some drawbacks. Make the decision depending on your application.

If you decide to use device fonts, remember that Flash Lite will try to match the selected generic font to a font on the device at runtime. Thus, if the device font you select is not available on a device or has differences, the unity of the user experience may be lost.



9.1.2 Embedded fonts

Embedding font outlines offers a known technique for ensuring that a dynamic text field's font appears the same on all target platforms. Embedding fonts on dynamic fields is not only a necessary practice, but is also the best way to ensure that fonts display correctly on all devices. Note that embedding all characters to a dynamic text field may increase the file size; you can reduce it by embedding only the characters required.

9.1.3 Placement of fonts

Use the **Snap to Pixel** feature under **View** to keep fonts clear. If the fonts are not of whole _x and _y values, they will appear blurry, especially if you are using pixel fonts.

9.2 Bitmaps

Use of bitmaps in a cross resolution and layout application should be minimised and, if possible, avoided. The reasons are obvious: Scaling an image may not offer a good presentation to the user, especially on Flash Lite 1.1, which doesn't have anti-aliasing.

If bitmaps are unavoidable, use different size variations to display the right size on the right layout. However, note that more bitmaps will result in larger file sizes, therefore a good optimisation might be required before importing these to the application.

10 Terms and abbreviations

Term or abbreviation	Meaning
CS3	Adobe Creative Suite 3, containing the Adobe Flash CS3 Professional tool, which is covered in this document.
CS4	Adobe Creative Suite 4.
Cursor	Vertical element indicating the insertion point or visual pointer.
Feedback	The feedback that an application provides after a user action.
Flash Lite	Lightweight version of Adobe Flash Player. Flash Lite players on Nokia mobile devices are covered in this document.
Focus	Interface element currently selected by the user.
Input methods	In the context of this document, mainly covering key and touch input methods in Flash Lite application development.
Key repeat	The key's function is performed consecutively while the key is kept pressed down.
Middle softkey	In the middle (centre) of navigation keys.
Keypad	All the main key buttons.
Navigation keys	Main directional pad, also named D-pad or scroll key. Can be other mechanical solution than keys.
Numeric keys	0-9, '*', and '#' keys.
Orientation	Refers to portrait and landscape modes, which reflect the current display aspect ratio.
Pixels per inch	Corresponds to pixel size per screen size ratio. If the device has a larger screen but fewer pixels, the PPI value is smaller; likewise, the value is larger with small screens with more pixels.
Scalability	Issue concerning layout and visual appearance when application is run on different screens.
Softpad	Small numeric input pop-up menu used in Sudokumaster.
Layout design	Arrangement of the elements in the graphical user interface.
Softkeys	Main left and right softkeys and optionally middle softkey.
Tap-and-hold	User input method in which a finger touches the surface and is held down, while moving.
UI, User interface	In this document, the graphical user interface designed with Flash Lite.
Visual guidance	Tips, hints, and clues provided to the user for improved usability.
QVGA	Quarter VGA resolution 320 x 240 pixels.
QWERTY	Standard keyboard layout set or alphabet text input mode.

11 References

[1] Sudokumaster: Designing a Flash Lite Game for Keypad and Touch Devices available at http://www.forum.nokia.com

[2] Flash Lite Developer's Library at http://www.forum.nokia.com

[3] Design and User Experience Library at http://www.forum.nokia.com

Further reading:

Getting Started with Flash Lite at http://www.adobe.com

Flash Lite Visual Guide at http://www.forum.nokia.com

Creating Mobile Content with Flash Lite at http://www.adobe.com

Series 40 UI Style Guide at http://www.forum.nokia.com

S60 UI Style Guide at http://www.forum.nokia.com

Introduction to fscommands in the Forum Nokia wiki at http://wiki.forum.nokia.com

12 Evaluate this resource

Please spare a moment to help us improve documentation quality and recognise the resources you find most valuable, by rating this resource.