
Introduction To The S60 Scalable UI

Version 1.4
February 24, 2006

S60 platform

Legal Notice

Copyright © 2006 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1.	Introduction	5
2.	Why Is Scalable UI Needed.....	5
3.	New Screen Resolutions	5
4.	New C++ APIs	6
5.	Bitmaps and Icons	6
6.	Localization.....	6
7.	Scalable Themes	7
8.	Compatibility and Design Basics.....	7
9.	Scalable UI in Java™ Applications.....	8
10.	Terms and Abbreviations	10
11.	References	11
12.	Further Reading.....	11
13.	Evaluate This Resource.....	12

Change History

September 16, 2004	Version 1.0	Initial document release
October 11, 2004	Version 1.1	References to screen resolution 208x208 removed.
January 18, 2005	Version 1.2	Correct SDK API names added
November 2, 2005	Version 1.3	Minor terminology update
February 24, 2006	Version 1.4	Chapters 2 and 12 added. Chapters 1, 8, and 9 updated.

1. Introduction

S60 2nd Edition, Feature Pack 3 introduces new screen resolutions and APIs to support the scalable UI. In addition, a scalable graphics format is introduced for icons and themes. In order to implement applications for the scalable UI, developers must take the different screen resolutions into account; however, when using standard S60 (Avkon) UI components (recommended), the UI scales automatically.

This document briefly describes what has changed with the introduction of the scalable UI, the new APIs and features provided by the S60 platform to support the change, and how and where the change needs to be taken into account by the developer.

More documents and code examples on developing scalable applications are available in Forum Nokia; see Chapter 12, "Further Reading."

2. Why Is Scalable UI Needed

Mobile devices are evolving to support higher resolutions and more accurate displays, which consumers demand and appreciate. Device segmentation, besides just price point, is getting broader with different requirements: for example, a music-optimized device can survive with a lower-resolution display compared to a device optimized for media consumption, such as Internet browsing. Bigger and more accurate displays also enable new use cases and richer content, such as maps, fancy games, photos, videos, and thereby also new business opportunities. But it is not just about resolution; it is also about orientation. Mobile TV and browsing, for example, utilize and fit better into a widescreen landscape display compared to the traditional portrait.

As explained above, display variance and evolution is needed and unavoidable. With that in mind, developers need to consider how to deal with that variance effectively, which is where the scalable UI concept comes into picture. Instead of providing multiple software platforms with almost the same APIs, one platform can support multiple UIs and screen resolutions by providing a scalable UI framework. Applications built on top of the S60 platform either work automatically on the supported resolutions or can be made to support all those resolutions with the help of the APIs provided by the platform. The level to which extent the platform handles the scalability automatically, as opposed to being handled by the developer, varies significantly depending on the application UI implementation.

3. New Screen Resolutions

Applications implemented for S60 2nd Edition, Feature Pack 3 and later should scale to all of the supported resolutions. In addition to the 176 x 208 resolution currently in use, two new resolutions are now available. From this point on, S60 devices may use one of the following screen resolutions:

- 176 x 208 – standard
- 240 x 320 – Quarter VGA (QVGA)
- 352 x 416 – double resolution

In addition to the traditional portrait layout, a landscape layout is supported in the double resolution (352 x 416) and QVGA (240 x 320) modes.

4. New C++ APIs

The following new APIs have been introduced to allow the development of applications for scalable UIs:

- UI Metrics API

The API provides layout information about the UI component (size, position, etc.). The UI Metrics API is part of the Utilities API under the UI Framework in the SDK help. The term UI Metrics API does therefore not appear in the SDK help.

- Orientation Mode API

The API provides `set` and `get` methods for the orientation mode (portrait or landscape). The Orientation Mode API is part of the Application framework API under the UI Framework in the SDK help. The term Orientation Mode API does therefore not appear in the SDK help.

- Scalable Icons API

The API is used to create scalable icons. The Scalable Icons API is part of the Skins API under the UI Framework in the SDK help. The term Scalable Icons API does therefore not appear in the SDK help.

- Logical Font API

New logical fonts are defined to get suitable fonts regardless of the current screen resolution. The Logical Font API is part of the Utilities API under the UI Framework in the SDK help. The term Logical Font API does therefore not appear in the SDK help.

5. Bitmaps and Icons

Bitmap graphics do not scale very well for different screen resolutions. Therefore a new C++ API has been introduced for loading old bitmap icons and scalable icons based on Scalable Vector Graphics – Tiny (SVG-T) [SVGT].

A new tool (MifConv.exe) is also available to produce multi-image files (MIFs) — similar to Multi-Bitmap (MBM) — from SVG-T files. A new application information file (AIF) framework has been introduced to support scalable icons.

6. Localization

Due to new resolutions and the change in the orientation mode the space available for text strings in the screen can change during runtime. Therefore a Symbian C++ application can use shorter and longer versions of the same text. However, it is wise to study the usability of each case. Texts in a localization file (.loc) can have alternative versions for different text spaces. When reading this kind of text from resources and passing it to Avkon components, the available space is checked for the text based on the current layout and orientation, and the correct alternative is selected. In addition, a utility function in `AknTextUtils` can be used directly, especially if the text is drawn directly to the screen.

7. Scalable Themes

In the previous S60 releases, themes were based on bitmaps. In S60 2nd Edition, Feature Pack 3, themes are extended to support SVG-T graphics. Themes can contain bitmaps or SVG graphics or a mixture of both.

Series 60 Theme Studio [THEM] will be updated to support new SVG-T graphics.

8. Compatibility and Design Basics

The key to an easily portable and scalable UI is good application design/architecture, such as separating the engine and UI code, not hard-coding layout information, and designing scalability in mind from the very beginning. This is of ever-increasing importance after the introduction of the S60 scalable UI.



Do not make any assumptions about the screen resolution or orientation, that is, do not hard code the screen resolution and layout.

If an application is using standard Avkon UI components, no modifications to the code, or only minor ones, may be needed, since the Avkon components scale automatically. However, recompilation is still needed.



As long as you do not make any assumptions about the screen resolution or orientation, your UI will be scalable when using the standard Avkon UI components.

If an application implements its own custom controls or draws directly to the screen, modifications are needed to handle different screen resolutions and orientations.



When using other than standard Avkon UI components, the UI does not scale automatically.

In addition, S60 2nd Edition, Feature Pack 3 has a compatibility mode that enables legacy applications designed for a 176 x 208 resolution to run in devices using other resolutions. In the compatibility mode there are bars around the screen (240 x 320) or all the pixels are quadrupled (352 x 416). Relying on that is highly unrecommended, since it typically does not result in an optimal and very good-looking solution. The compatibility mode is no longer supported in S60 3rd Edition.



In addition to screen resolution and orientation, there are also other dimensions in the UI that may need to be taken into account, such as the keyboard, number of softkeys, and screen color depth.

There are several different ways to use graphics and draw to the screen. The implementation of the application determines how and to which extent scalability needs to be taken into account by the developer, as opposed to being automatically handled by the platform. Table 1 is a simplified summary of the graphics/draw types and their scalability.

Type	Screen resolution	Screen color depth	Compatibility mode (2nd Edition FP3)	Format	Benefits / where to use
Standard Avkon UI components	Automatic	Automatic	Not relevant	Vector	Least work for the developer, best compatibility and scaling, tested and proved usability.
Custom UI components	Developer (utilizing the APIs)	Automatic	Yes	Vector (preferred), bitmap	Where a custom UI is needed; often with games and personalization, for example.
Traditional screen drawing - vector graphics	Developer (utilizing the APIs)	Automatic	Yes	Vector	Vector graphics provide better scalability and should be preferred where they fit, i.e., drawing shapes and where the images can be formed by drawing shapes (2nd Edition FP3 onwards).
Traditional screen drawing - bitmap graphics	Developer (utilizing the APIs)	Automatic	Yes	Bitmap	Bitmaps need to be used with "real life pictures." Also better performance may be achieved with bitmaps.
Direct Screen Access - low-level graphics	Developer (utilizing the APIs)	Developer	No	Bitmap	Best performance, but most work and least compatibility. To be used only if maximum performance is really needed.
3D Graphics and OpenGL ES	Scale by nature	Automatic	Yes	3D	3D

Table 1: Graphics/draw types and their scalability

9. Scalable UI in Java™ Applications

In Java™ applications, high-level UI components scale automatically. However, developers must take care of scaling `Canvas` and `CustomItem`. If MIDlets are setting the preferred sizes for `Form` items, they may also need to be scaled with resolution changes. Scalable icons are not available in the Java platform; therefore a set of bitmaps are needed for each of the resolutions. In the future JSR 226: Scalable 2D Graphics API for the Java™ 2 Platform, Micro Edition (J2ME™) [JAVA] provides support for SVG. Device implementation gives notification (calls the `Displayable.sizeChanged` method) when the drawable area changes.

S60 devices have several display resolutions. The trend towards larger resolutions and also different resolutions within one device (for example portrait and landscape modes in a device) may pose compatibility and performance problems with the Java MIDlets designed for the earlier fixed and, often, smaller resolutions. Therefore, it is necessary to enable scaling of the Java execution resolution.

The scaling is enabled by two attributes that may be placed in the JAD file or the manifest:

- `Nokia-MIDlet-Original-Display-Size`
- `Nokia-MIDlet-Target-Display-Size`

The scaling means essentially that, from the MIDlet's point-of-view, all pixel coordinates and sizes in all classes function as if the device's display resolution was the resolution defined in the scaling attribute `Nokia-MIDlet-Original-Display-Size`. However, the MIDlet is scaled on the physical display by the Java platform implementation towards the device's full display resolution. The MIDlet may indicate its desired target resolution with the other scaling attribute `Nokia-MIDlet-Target-Display-Size`.

The attributes are applied to MIDlets with `Canvas` and `GameCanvas` in full-screen mode and `FullCanvas` in the Nokia UI API. `Canvas` and `GameCanvas` in normal mode and `Screens` are not affected by the defined scaling behavior. The attributes affect all appropriate MIDlets in a MIDlet Suite.

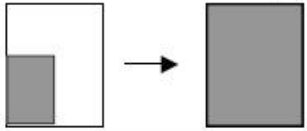
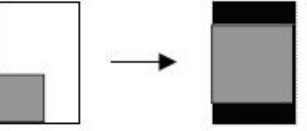
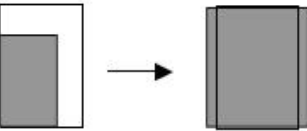
Use case	Attribute values	Referential result on the display
A 176x208 game is scaled to full-screen 352x416. <i>(Note that the developer gets the same result even without defining the target display size.)</i>	<code>Nokia-MIDlet-Original-Display-Size: 176,208</code> <code>(Nokia-MIDlet-Target-Display-Size: 352,416)</code>	
A 128x128 game is scaled to portrait QVGA 240x320, leaving black borders on top and bottom sides (40 pixels each). <i>(Note that the developer gets the same result even without defining the target display size.)</i>	<code>Nokia-MIDlet-Original-Display-Size: 128,128</code> <code>(Nokia-MIDlet-Target-Display-Size: 240,320)</code>	
A 176x208 game is scaled to full-screen portrait QVGA so that the aspect ratio is changed AND a few pixels are dropped from the left and right. The target resolution could be for example 256x320, so that eight pixels are lost from both sides.	<code>Nokia-MIDlet-Original-Display-Size: 176,208</code> <code>Nokia-MIDlet-Target-Display-Size: 256,320</code>	

Table 2: Examples of the scaling attributes' intended use and behavior

10. Terms and Abbreviations

Term or abbreviation	Meaning
AIF	Application information file. Specifies an application icon and caption.
Avkon	S60 UI library.
Bitmap	Bitmap graphics are pixel-by-pixel representations of an image. For each pixel, there are a number of bits that define the color of the pixel. Common bitmap file types include BMP, JPEG, and PNG.
QVGA	Quarter VGA. QVGA is a common size for high-resolution phone displays. VGA is 640 x 480 pixels and QVGA is one-quarter of the area.
MBM	Multi-Bitmap file. A file format for storing bitmap images.
MIF	Multi-image file. Similar to an MBM file, but used for scalable icons instead of bitmaps.
SDK	Software development kit.
SVG	Scalable Vector Graphics. A language for describing two-dimensional graphics and graphical applications in XML.
SVG-T	SVG Tiny. A Mobile SVG Profile containing a subset of SVG modules.

11. References

- [JAVA] Java Specifications, Sun Microsystems Inc.
<http://java.sun.com/>
- [SVGT] Mobile SVG Profiles: SVG Tiny and SVG Basic, W3C SVG Working Group
<http://www.w3.org/TR/SVGMobile/>
- [THEM] Series 60 Theme Studio (for Symbian OS), Forum Nokia
<http://www.forum.nokia.com/tools>

12. Further Reading

See the following resources available on the Forum Nokia Web site:

- [Scalable UI Learning Path](#)
- [S60 Platform: Scalable UI Support](#)
- [S60 2nd Edition Feature Pack 3: Scalable UI Example](#)
- [S60 Platform: Scalable Screen-Drawing How-to](#)
- [S60 Platform: Scalable Screen-Drawing Example](#)
- [S60 Platform: Vector Graphics Optimization](#)
- [Symbian OS: 2D Game Engine Example](#)

13. Evaluate This Resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by [rating this resource](#).