

# Series 40 Web Apps Best Practices Guide

**NOKIA** Developer

Document created on 25 October 2011

Version 1.2



# Table of contents

1.	Introduction	3
1.1	Introduction to Nokia Browser for Series 40	3
1.2	Other documents	4
2.	Unsupported web techniques	5
2.1	Page layout	5
2.2	Graphics	5
2.3	General browser functions	6
2.4	User input	6
2.5	Advanced features	7
3.	Accommodating screen orientation	8
4.	Getting the best performance	10
4.1	Hardware factors affecting performance	10
4.2	Client/Server round trips	10
4.3	Image handling	10
4.3.1	Images in a web app	10
4.3.2	Images from the web	11
4.4	Suggestions for optimal performance	11
4.4.1	Split large pages	11
4.4.2	Download content incrementally	11
4.4.3	Use simple page design	12
4.4.4	Avoid unnecessary graphics	12
4.4.5	Find ways to reduce processing	12
4.5	Recommended techniques for common page structures	13
5.	Other tips and tricks	14
5.1	Multi-page web apps	14
5.2	Page layout suggestions	14
5.2.1	A margin and negative margin example	15
5.3	A non-MWL reference problem	17
6.	Font support	19
7.	Third-party JavaScript libraries	20
7.1	Round trip considerations	20
7.2	Metered APIs	20

## Change history

7 April 2011	1.0	Initial document release
6 June 2011	1.1	Updated to add advice on handling screen orientation
25 October 2011	1.2	Updated to cover features of Series 40 web apps 1.5

# 1. Introduction

This document provides information on the coding practices that are the best to adopted in creating Series 40 web apps, practices that will assist in ensuring your web apps provide the best overall UX.

## 1.1 Introduction to Nokia Browser for Series 40

Nokia Browser for Series 40 is a distributed or proxy-based web browser that supports full web page rendering on phones with limited processing power and memory (such as some Series 40 phones). As shown in Figure 1, a Nokia Browser for Series 40 Proxy server communicates with websites on behalf of the Nokia Browser for Series 40 Client on a phone. The server then performs the process intensive tasks associated with rendering web pages, such as executing scripts or resizing images. The server then passes optimised web content, reduced in payload and streamlined for efficient rendering, to the Nokia Browser for Series 40 Client on a phone.

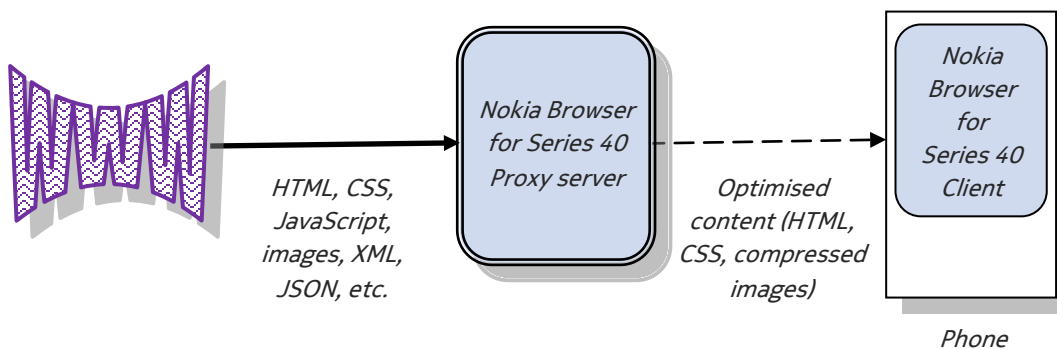


Figure 1: An overview of the Nokia Browser for Series 40 architecture is shown.

However, a unique feature of Nokia Browser for Series 40 among proxy-based browsers is that some JavaScript™ code can be executed on the phone. This feature enables Series 40 web apps to offer an enhanced user experience through dynamic UIs with features such as transitions.

This novel design means that when coding a Series 40 web app there are some new use cases to consider to ensure the best experience is provided to users. Many functions work as they do in a conventional browser, but others must take into account the processing power and memory on Series 40 phones and the need for client and server to communicate frequently.

A web app's HTML and JavaScript code is processed on the proxy server only. The server prepares all content for the handset, converting it to optimised HTML and delivering JavaScript codes that use the client's MWL (Mobile Web Library) JavaScript library. MWL contains code to support application-like interaction on the phone. MWL

processing should normally be the only JavaScript that executes on a phone. (For more information on MWL, please see the [Series 40 Web Apps Developer's Guide and API Reference](#).)

The content of this document applies to version 1 and version 1.5 of Series 40 web apps.

## 1.2 Other documents

See these documents for more information on Nokia Browser for Series 40 and Series 40 web apps:

- [Series 40 Web Apps Platform Overview](#).
- [Series 40 Web Apps Developer's Guide and API Reference](#).
- [Series 40 Web Apps Publishing Guide](#).
- [Series 40 Web Apps UX Guidelines](#).

## 2. Unsupported web techniques

This chapter provides information on the web design techniques that aren't supported in Series 40 web apps or that could result in performance issues in the Nokia Browser for Series 40 environment. Recommendation on how to allow for unsupported or performance related issues are provided.

Tables in the [Series 40 Web Apps Developer's Guide and API Reference](#) list the HTML tags and CSS properties supported in Series 40 web apps.

### 2.1 Page layout

Table 1 lists the page layout features not supported in Series 40 web apps and, where possible, information on how to work around these limitations.

Unsupported technique	Comments and workarounds
Some types of nested tables	It's recommended that you experiment with nested tables to determine what works in your web app.
<code>z-index</code> , <code>float</code> , and <code>position</code>	Workaround for float and general lateral positioning: Use tables.
Most overflow modes (all but <code>visible</code> and <code>hidden</code> )	Workaround: use <code>overflow:hidden</code> .

Table 1: Unsupported page layout techniques are listed.

### 2.2 Graphics

Table 2 lists the graphics handling features not supported in Series 40 web apps and, where possible, information on how to work around these limitations.

Unsupported technique	Comments and workarounds
Changing image dimensions on the phone	No workaround available.
Opacity and RGBA graphics	Workaround: Use transparent PNGs.

Unsupported technique	Comments and workarounds
Canvas/SVG support	No workaround available.
Fine gradients and high colour depth	Be aware that the proxy server automatically reduces colour depth to 256 colours (8-bit) or less depending on optimisation requirements. Users might set the depth even lower to reduce their bandwidth charges.

Table 2: Unsupported graphics techniques are listed.

## 2.3 General browser functions

Table 3 lists the general browser functions not supported in Series 40 web apps and, where possible, information on how to work around these limitations.

Unsupported technique	Comments and workarounds
Directly modifying client-side style properties with JavaScript	Workaround: Change predefined classes to modify properties.
Using JavaScript native timers	Timers don't perform well when run on the proxy server rather than a phone. Workaround: Use the <code>MWL.timer()</code> and <code>stopTimer()</code> methods. For more information, see <a href="#">Series 40 Web Apps Developer's Guide and API Reference</a> .

Table 3: Unsupported general browser function techniques are listed.

## 2.4 User input

Table 4 lists the user input features not supported in Series 40 web apps and, where possible, information on how to work around these limitations.

Unsupported technique	Comments and workarounds
Direct gesture support using gestures other than <code>swipe</code> and <code>longPress</code>	Workaround: Use the built-in <code>swipe</code> and <code>longPress</code> gestures. For more information, see <a href="#">Series 40 Web Apps Developer's Guide and API Reference</a> .

Table 4: Unsupported user input techniques are listed.

## 2.5 Advanced features

Table 5 lists the advanced features not supported in Series 40 web apps and, where possible, information on how to work around these limitations.

Unsupported technique	Comments and workarounds
Timing-dependent games requiring real-time feedback processing	Since most of the system processing occurs on the Nokia Browser for Series 40 Proxy server rather than the phone, the client-server communication lag makes this type of game impractical. This environment supports games that run entirely on the phone with no proxy server support (other than the initial download).

Table 5: Unsupported advanced feature techniques are listed.

## 3. Accommodating screen orientation

Series 40 web apps run on Series 40 phones with 240 x 320-pixel screens that can be in landscape or portrait orientations. To offer the best UX it's desirable for web apps to be formatted optimally for both orientations.

To ensure content fits optimally to each screen orientation, the following formatting approaches should be used:

- Where content can be formatted by use of relative size controls it should be. For example, by the use of percentage width settings in tables.
- Where content needs to be explicitly sized, it should be done so by use of separate CSS definitions for landscape and portrait screen orientation.

The use of relative size controls is straightforward, standard HTML coding practices can be used. However, setting the correct CSS for landscape or portrait orientation may not be obvious. The first step is to add JavaScript code to your web app that queries the screen object and reads the width property. Next test the screen width (anything greater than 240 indicates landscape orientation). Use the result to conditionally write a `<link>` to the web app's document for the CSS file defining the layout for the detected screen orientation, as shown in Example 1.

```
/**
 * Selecting a CSS file based on screen width
 */
if (screen.width > 240) {
    document.write('<link rel="stylesheet" href="basicLandscape.css"
type="text/css" />');
}
else {
    document.write('<link rel="stylesheet" href="basicPortrait.css" type="text/css"
/>');
}
```

Example 1: JavaScript code to determine screen orientation and define the corresponding CSS is listed.

Now it's simply a case of executing this JavaScript from the <head> of the web app's main HTML, as shown in Example 2.

```
<head>
  <title>Screen size example</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <!-- Common css between the two orientations -->
  <link rel="stylesheet" href="common.css" type="text/css" />

  <!-- The following script sets and gets the appropriate CSS file for the
screen
  orientation -->
  <script type="text/javascript" src="screensize.js"></script>
</head>
```

Example 2: The code to execute the script to determine screen orientation and create the appropriate CSS file link is shown.



Note: This change is static as Series 40 phones don't support dynamic orientation changes.

## 4. Getting the best performance

The performance of a Series 40 web app will depend on the features of the phone upon which it is running and the characteristics of the network over which the client and server communicate. In addition, the design of the web app itself can affect performance, particularly the frequency of round trips between the client and server. This chapter examines these issues and provides information on how to optimise the performance of your web app.

### 4.1 Hardware factors affecting performance

Several factors affect the performance of Series 40 web apps, these include the:

- speed of the phone's processor.
- availability of runtime and permanent storage memory.
- communication technology used between the server and client.

### 4.2 Client/Server round trips

A *round trip* is a common communication method between the Nokia Browser for Series 40 Client and the Nokia Browser for Series 40 Proxy server. In a round trip, the client browser sends a request to the server. The client then waits while the server builds a response and sends it back to the client. Nokia has taken steps to speed up this communication (by using optimised HTML, not re-sending unchanged content for page updates, compressing graphics, and such like), but a round-trip sequence will still take several seconds to complete. Some round-trip communication is unavoidable, but your web app will probably work better if you keep it to a minimum.

### 4.3 Image handling

Web apps source images from two locations: the internet and from in the web app package.

#### 4.3.1 Images in a web app

From Series 40 web apps 1.5 images included in the web app are cached on the phone. This helps improve page display performance and reduces overall network traffic, as these images don't need to be sent to the client each time the web app or a web app page is opened. Therefore, any static images should be included in the web app and not downloaded from the internet.

In addition, all static images should be delivered at the size used in the web app and using 256 colours (8-bit colour depth).

## 4.3.2 Images from the web

The Nokia Browser for Series 40 Proxy server processes all internet sourced by scaling them to the size required and reducing the colour depth (if necessary). Where images can be made available exclusively for use in the web app it's advantageous to adjust the size and colour depth, thereby eliminating the need for the Proxy server to process the images (unless the user has chosen a lower colour depth for images). This reduces the server side processing and thereby reduces network traffic and improves response time.

## 4.4 Suggestions for optimal performance

Good mobile page design principles are as important in Series 40 web apps as they are in web design for other mobile browsers. This section provides some guidance on how you might improve your web app's performance. This isn't an exhaustive list of tips and tricks, and you might not find that all of these ideas will help your web app. It may be necessary to experiment with the use of these techniques to determine which ones offer the optimal results in any particular web app.

When designing your web app, remember to consider that memory and processing power are limited resources on Series 40 phones.

### 4.4.1 Split large pages

If you wish to present long articles to the user, consider the following techniques to reduce memory requirements:

- Split the articles into several pages. This reduces memory use as its the split page's content only that needs to be stored in memory. This approach may also improve the perceived performance, as each page will have a short download time and — although the overall download time for the entire article will similar to loading a single page — the user perception is likely to be of a better overall performance.
- Provide a 'view entire story' link and open the article in the Nokia Browser for Series 40.

### 4.4.2 Download content incrementally

Downloading large blocks of content in one go can affect the web app, due to the download time as well as impact on the phone's memory.

Consider these suggestions:

- if you have a single page that contains a long list of items, such as a list of products, consider delivering the content in an accordion list. The initial load would display the product name and a small graphic. Then, when the user wants more details on an item, those details can be downloaded when they expand the accordion. In

this case the web app downloads only the changed section of the page. As most user will not open all items in the list, the network and phone memory use reductions from using this technique could be beneficial.

- if you have an application with several tabs, each displaying different content, it might be appropriate to download the content for the active tab only, rather than for all tabs, to reduce memory usage. When the user selects another tab, the app would fetch the content for the second tab. However, as downloading each tab’s content will involve a round trip to the server testing to find the best balance between approaches is advised.

### 4.4.3 Use simple page design

The more you can reduce the use of graphics, the better it is for page loading times and memory use.

### 4.4.4 Avoid unnecessary graphics

In addition to reducing the use of graphics, there are techniques that can be used to optimise graphics handling. Take for example the use of a single-pixel tiled background image instead of setting the background colour with CSS properties.

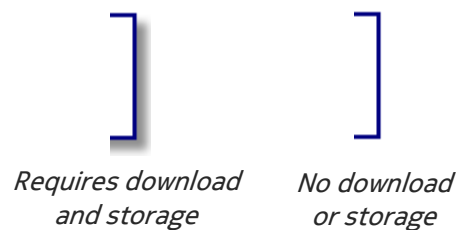


Figure 2: An example of avoiding unnecessary graphics is shown.

Another example is that you might consider to improve the visual appeal of your web: using boxes with drop shadows or rounded corners rather than square corners.

The Nokia Browser for Series 40 Client has no built-in support for these styles, so you would have to construct the elements with downloaded graphics. Using a simple square cornered graphics, as shown in **Error! Reference source not found.**, would by comparison save memory and reduce a page’s load time.

### 4.4.5 Find ways to reduce processing

In addition to optimising for memory use, in Series 40 web apps it’s important to be aware of the processing overheads created by certain content.

Take the requirement to display a button 50 pixels wide, with a gradient from top to bottom. You could do this by downloading a vertical slice of the image, then replicating the slice a number of times to construct the button. To save memory and improve download speeds, you might consider downloading a slice that is 1 pixel wide, but then the processor must do a lot of work to

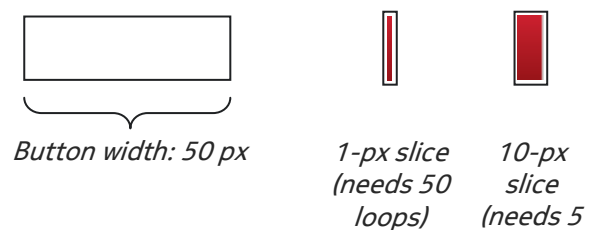


Figure 3: Options for reducing the processing time for building a graphic are shown.

replicate the slice into the final graphic. Another approach would be to send down a 10-pixel slice, as shown in Figure 3, which the processor could use to build the button significantly faster (with one-tenth the number of processing iterations). Again, you may have to experiment to find what works best in your application.

## 4.5 Recommended techniques for common page structures

For ideas on the best way to implement accordions, carousels, tabbed pages, lists, and other page structures, please see [the Series 40 web app code examples](#) on the Nokia Developer website.

## 5. Other tips and tricks

This chapter contains miscellaneous tips and techniques that don't fit into the previous chapters.

### 5.1 Multi-page web apps

If your web app is to contain multiple scenes, views, or pages you shouldn't create separate HTML pages and loading them using `mw1.loadURL()`. If you do so, you may encounter rendering issues with any pages other than `index.html`.

The correct approach is to place the HTML content for each view in its own container (such as a `div`) in the `index.html` file. Then, to display a specific view the currently displaying view should be hidden and the required view shown using JavaScript code.

One technique that can help with this is to add a couple of classes, as shown in Example 3, and apply them to the `divs` you want to show and hide.

```
.hidden {
display: none;
}

.visible {
display: block;
}
```

**Example 3:** Classes to assist with the display of views in a web app are listed.

Also note that hiding a container node (such as a `div`), instead of removing the node, provides for faster access to the `div`'s content within the same web app session; as there will be no need to request the content and rendering methods from the server when the user switches back to the view.

### 5.2 Page layout suggestions

There are no definitive rules for the best way to lay out pages. What works well for one web app might not work for another. This section suggests a few different approaches to try:

- Tables are good tools for setting up the spacing in a web app. You can use column widths to specify the horizontal location of different page elements, and row heights to control vertical location. Nested tables (tables within tables) are useful also, though not all nested tables work correctly in Series 40 web apps. It's important to test your application on all target phones to make sure the layout is as you expected.

Be aware that tables can sometimes have an effect on web app performance. If you notice slow responses for operations such as animated transitions — processes that run purely on the phone without proxy server intervention — see if removing one or more tables from the page layout improves the response.

- CSS margins are another way to handle horizontal layout. While you may not be familiar with this use of margins, a number of common HTML tags have margin properties you can use to set spacing. (For a complete list of tags supporting margins, please see Appendix A of [Series 40 Web Apps Developer's Guide and API Reference](#).)

As you create the layout for your web app pages, experiment with different techniques and test as much as possible on all target phones.



Note: Series 40 web apps don't support the `float` and `clear` properties.

## 5.2.1 A margin and negative margin example

Example 4 shows the body of an HTML file that displays two coloured squares. The squares are defined in the `<style>` section, which occurs in the code that follows the sample code shown.

```
<body>
  <div>
    <div id="btn1">
    </div>
    <div id="btn2">
    </div>
  </div>
</body>
```

Example 4: The body of an example page displaying two coloured squares is shown.

The default alignment of `div` statements such as these is vertical. You can see this in the first row of Table 6, which has `<style>` settings, but no margins. You can add margin settings to the box definitions to align them horizontally instead.



<style> settings	Result
<pre>#btn1 {   height: 40px; width: 40px;   background-color: yellow; } #btn2 {   height: 40px; width: 40px;   background-color: aqua; }</pre>	
<pre>#btn1 {   height: 40px; width: 40px;   background-color: yellow;   margin-left:0;   margin-top:0; } #btn2 {   height: 40px; width: 40px;   background-color: aqua;   margin-left:40px;   margin-top:-40px; }</pre>	

Table 6: The code for aligning divs horizontally is listed.

In the style settings for #btn2, the `margin-left` setting adds 40 pixels to the left margin, in effect moving it to the right of the first square. The negative `margin-top` value moves the top margin *up* by 40 pixels, putting it at the same height as the first square, as shown in Figure 4. (A positive top margin adjustment would move the square down rather than up.)

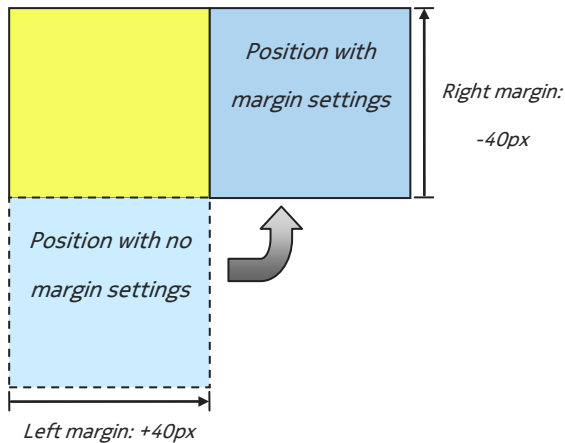


Figure 4: The effects of margin settings are shown.

## 5.3 A non-MWL reference problem

You may run into unexpected problems due to the way in which the client processes MWL and non-MWL JavaScript function calls. Take the code in Example 5, which has been simplified to illustrate the issue.

```
<script>
function attachHorizView()
{
    // Function processing includes defining #btns (not shown).
}
</script>

<!-- #btns doesn't exist until attachHorizView runs on the server: -->

<div onclick="attachHorizView();
    mw1.setGroupTarget('#btns', '#btn0', 'show', 'hide');
    mw1.switchClass('#views', 'view1', 'view2');">
```

Example 5: The code causing an unexpected run-time error is listed.

One action of the `attachHorizView` function is to define an object called `#btns`. The intention of the code is to define `#btns` first and then reference it in the `setGroupTarget` call that follows. The problem is that the distributed browser doesn't execute the JavaScript calls in the expected order.

When the Nokia Browser for Series 40 Client processes a line that calls both MWL and non-MWL functions, it performs the MWL calls first, regardless of order. This is because MWL is a local library, running on the phone. The client browser first processes all MWL calls, then asks the Nokia Browser for Series 40 Proxy server component to handle the non-MWL calls. Thus the actual order these calls occur in is this: `setGroupTarget`, then `switchClass`, and finally `attachHorizView`, rather than the expected `attachHorizView`, `setGroupTarget`, and `switchClass`. In the code in Example 5, when the client browser performs the `setGroupTarget` call, `#btns` isn't defined, causing a run-time error.

This issue can be fixed by moving the `setGroupTarget` call into the `attachHorizView` function itself, at the end of the function, as shown in Example 6.

```
<script>
function attachHorizView()
{
// Function processing includes defining #btns (not shown).

// setGroupTarget call moves to the end of the attachHorizView function.
  mwl.setGroupTarget('#btns','#btn0', 'show', 'hide');
}
</script>

<!-- Now the setGroupTarget call happens after all other processing in
      attachHorizView. -->

<div onclick="attachHorizView();
              mwl.switchClass('#views', 'view1', 'view2');">
```

**Example 6:** The code corrected to achieve the desired processing is listed.

Now, when the client processes the `setGroupTarget` call, the variable `#btns` is defined and everything works as expected.

## 6. Font support

This chapter examines issues that may be caused by the limited font support in Series 40 web apps.

Series 40 web apps support one font only, although the supported font changed between Series 40 6th Edition and Series 40 6th Edition, Feature Pack 1. For more information see [Series 40 Web Apps UX Guidelines](#). The server component maps all fonts used in a web app to this Nokia font, and it doesn't support downloading other font definitions. The system supports simple text effects like bold text and italics with CSS styles or standard HTML tags such as `<b>` and `<i>`.

## 7. Third-party JavaScript libraries

Your web app can use third-party JavaScript libraries for interactive UI elements. This chapter makes recommendations on the use of such libraries.

### 7.1 Round trip considerations

Non-MWL JavaScript code is executed on the server rather than a phone. As a result, each library call requires round-trip communication between the Nokia Browser for Series 40 client and server. Because of the time delays involved with round-trip messaging, it's recommended that you use MWL for UI manipulation rather than third-party JavaScript libraries if possible.

### 7.2 Metered APIs

Many third-party APIs for web services, such as Nokia Maps or Google APIs, apply use limits. In such cases, the use of APIs that associate all requests to a developer API key, rather than per IP address, are recommended.

APIs that restrict the number of requests on a per IP address basis will probably consume any use limit quickly, as the requests from all instances of a Series 40 web app are routed through the Proxy server and therefore are associated with a single IP address.

Copyright © 2011 Nokia Corporation. All rights reserved.

Nokia and Nokia Developer are trademarks or registered trademarks of Nokia Corporation. Oracle and Java are registered trademarks of Oracle and/or its affiliates. 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. 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 document at any time, without notice.

Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release.

#### Licence

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