

# Nokia Web Browser Design Guide

Version 1.0; May 29, 2007

# Browsing

**NOKIA**

Copyright © 2007 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are trademarks or 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. 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

<b>1</b>	<b>Introduction</b> .....	<b>5</b>
<b>2</b>	<b>Overview of the Nokia Web Browser</b> .....	<b>6</b>
2.1	The Nokia Web Browser — An open-source browser delivering the full Web experience .....	6
2.2	Browser features and standards support .....	6
2.3	Page rendering with True Web layout.....	7
2.4	Latency and caching.....	8
<b>3</b>	<b>Content delivery, usability, and development and testing environments</b> .....	<b>9</b>
3.1	A different approach .....	9
3.2	W3C Mobile Web Best Practices — Developing usable Web sites for mobile browsing .....	10
3.3	Development and testing environments .....	11
<b>4</b>	<b>Basic Web development issues</b> .....	<b>13</b>
4.1	Getting started: Building simple HTML structures.....	13
4.2	Utilizing the “hot spot”: Designing effective navigation .....	13
4.3	Supporting the mobile user with navigational aids.....	14
4.4	Using CSS for presentation .....	15
4.4.1	Embedded versus external CSS.....	15
4.4.2	Tailoring presentation format.....	15
4.5	Font size recommendations.....	15
<b>5</b>	<b>Site layout design</b> .....	<b>17</b>
5.1	Detecting the mobile browser .....	17
5.2	Two-column layouts.....	18
5.3	Tips for making mobile browsing efficient.....	18
<b>6</b>	<b>Media formats</b> .....	<b>19</b>
6.1	Video content in the mobile browser.....	19
6.2	Designing Flash for the mobile browser .....	19
6.2.1	Substituting Flash content.....	20
6.3	Embedding audio .....	20
6.4	Embedding SVG .....	21
<b>7</b>	<b>Progressive enhancement — the future Web design philosophy</b> .....	<b>22</b>
<b>8</b>	<b>Evaluate this resource</b> .....	<b>23</b>

## Change history

May 29, 2007	Version 1.0	Initial document release

# 1 Introduction

This document contains information about the Nokia Web Browser, mobile browsing in general, and Web development issues that take users using mobile browsers into consideration. The document includes guidelines and recommendations, as well as best practices, and is supplemented with simple, practical design and code examples that can be used as templates.

The goal of the document is to help developers see what kinds of issues to consider as their audience begins to use more and more mobile clients to access Web content. Some modifications may be small and easy to implement on top of the existing site design, while others may require more extensive work. At the end of the day, the “one-site-for-all” approach, where there is no need for completely separate mobile and desktop sites, will prove to be very effective for developers and intuitive for users.

The document covers the following subjects:

- Discovering the Nokia Web Browser
- Basic layout example
- CSS techniques for mobile design
- Useful code snippets
- Embedded media

## 2 Overview of the Nokia Web Browser

### 2.1 The Nokia Web Browser — An open-source browser delivering the full Web experience

The new Nokia Web Browser has a fundamental design philosophy — to deliver a full, desktop-like, Web experience with a mobile browser, without the need for creating and maintaining separate Web sites for desktop units and mobile devices. First introduced with the S60 3rd Edition release, updated versions of the Nokia Web Browser are arriving with each platform release. The new browser will be delivered with all future S60 3rd Edition devices, ensuring that the user base will grow rapidly in the near future.

With the new browser readily available on their devices, users can access the same Web content they would on their desktops, checking the latest news, accessing Web mail, reading their favorite blogs, and checking updates by using RSS feeds. However, the mobile use context must address a number of technical obstacles affecting the browsing experience. The most important considerations are the limited screen size on the mobile device and the latency in the network connection.

While focusing on delivering a desktop-like experience, the Nokia Web Browser includes features specifically designed to help the mobile browsing context. For example, the Page Overview (or MiniMap) feature allows the user to quickly see a complete overview of the entire page. Latency issues are addressed with caching algorithms specifically tailored for mobile use.

The Nokia Web Browser is based on the WebCore and JavaScriptCore open source components, which are also used by Apple's Safari Internet browser. The open source components, such as the KHTML Rendering Engine, are covered by the GNU Lesser General Public License (LGPL). The KHTML engine is developed by the KDE project and is considered faster than the Gecko engine used in the Mozilla and Mozilla Firefox browsers, for example. The browser also includes Symbian components and Nokia proprietary components, such as the Browser Control API, Browser UI, and HTTP filters.

### 2.2 Browser features and standards support

The Nokia Web Browser offers numerous innovations that take the user's experience of browsing the Web with a mobile device to the next level.

For developers designing Web sites, the most important features to consider are:

- **True Web Layout:** The browser displays Web pages in their original layout, as they would look on a desktop screen.
- **Pointer navigation:** An arrow pointer enables the user to navigate around a page intuitively, exactly as he or she would when browsing on a desktop computer using a mouse.
- **Page Overview:** The user is provided with a miniature thumbnail view of the entire Web page to quickly check his or her position on the page.
- **Visual History:** Navigation history is displayed as graphical thumbnails of all pages visited during the current browsing session. When the user selects a thumbnail image, the selected page is displayed immediately without reloading.
- **Landscape mode:** On supported devices, the landscape mode can be used to display Web pages, allowing the user to see more of the page horizontally without scrolling.
- **Web Feeds:** The user can subscribe to RSS feeds from Web sites, blogs, and news services. The feeds are accessible from the Bookmarks view.

- **Multiple Windows with Pop-up Blocking:** Links can be opened to a new window while browsing, and the navigation history of each window is separate and unique. The user can choose to allow or block pop-up windows while browsing. The total number of simultaneous windows is limited to 5.
- **AJAX support:** The browser supports Asynchronous JavaScript™ and XML (AJAX) applications, such as Google Maps.
- **Customization:** The browser can be customized by mobile operators or S60 licensees. Individual users can customize the browser with their own settings.

The Nokia Web Browser is highly compliant with industry standards. The support includes the following specifications:

- HTML 4.01, XHTML 1.0, including image maps, frames, background images, `<meta>` and `<object>` tags;
- CSS 1, 2, 3 (partially), including external style sheets;
- DOM 1, 2;
- SVG-Tiny;
- JavaScript 1.5.

The browser supports plug-ins for SVG-T, audio, and video. For more detailed information, refer to the comprehensive feature tables published on the Forum Nokia Web site.

Web developers can get architectural details about the browser, download the SDK, and get access to the S60 WebKit source code at [opensource.nokia.com/projects/s60browser](http://opensource.nokia.com/projects/s60browser).

### 2.3 Page rendering with True Web layout

Unlike earlier mobile browsers, the Nokia Web Browser can render any Web page (HTML, XHTML, WML) and display it on the device, making all Web content accessible to mobile users. The browser displays Web pages in their original form, retaining the page structure and layout as intended by the page designer. The user can intuitively scroll the view and move within the content using the pointer with the navigation key. Users can use the Page Overview to get a high-level view of the entire page and quickly move across the page to the area of interest.

The browser eliminates the need to vertically scroll back and forth through long lines of text by modifying elements containing text. With any column or frame that contains a text flow, the browser wraps the text to fit the maximum screen width on the device. This feature, called Layout Scaling, facilitates intuitive vertical scrolling and makes reading text from the screen easier. Users can also enlarge or shrink image and font sizes according to their preferences.

Browser performance is enhanced by the ability to display page content incrementally as content is downloaded, allowing the user to quickly view page content even before the entire page has been loaded. Upon initial loading of the page content, the browser first renders the page layout immediately without external CSS or script files. The page is quickly scanned for all external objects.

After the initial rendering, the browser performs a second pass of rendering that includes the external CSS and script files. Remaining external objects are included to the displayed page. The browser stores the files to the cache.

The current industry approach is to produce tailored solutions for mobile users (such as Google Mobile or Opera Mini) but the True Web layout approach aims at delivering the familiar desktop-like browsing experience to the mobile context — just as the Web designer intended. The different approaches will be discussed in Chapter 3.

## 2.4 Latency and caching

In mobile Internet browsing, latency and data transfer speeds are more significant considerations than in the desktop world, except when using WLAN to connect the mobile device. Using device cache in an efficient manner is very important to enhance the browsing experience by speeding up the page-rendering process and keeping latency and the need to transfer data between the network and the client device to a minimum.

The Nokia Web Browser uses two separate caches: the memory (or RAM) cache and a file system cache. Both support all the cache directives specified in the HTTP header. The file system cache size and location on the file system can be configured.

The caching algorithm is optimized to Web content. Items are removed from the cache based on the frequency of use as well as file size. For example, if items A (large file size) and B (small file size) are used equally often, item A will be removed before item B. The caching algorithm gives special weight and priority to CSS and JavaScript files, which in Web use are often reused and heavily impact the time needed to display the complete Web page.

Some guidelines for effective utilization of the browser cache:

- Specify the cache headers. Using E-Tag or Last-Modified will often result in calls to the network server for content updates. Unnecessary server connections can be avoided with a simple expiration header.
- Reuse CSS, images, and script files. Reference the same files in subsequent files to reuse the same downloaded file and keep the file in the cache.
- Remember that the device cache is limited and how the caching algorithms work. Do not solely rely on certain files from your site, such as large images, to be in the device cache for the entire duration of the browsing session — it is possible that the mobile browser will have to reload them after they have expired from the cache.

## 3 Content delivery, usability, and development and testing environments

This chapter takes a look at how content is provided for mobile devices, gives an overview on usability issues, and explores development and testing environments. The rest of the document will research practical techniques, such as designing navigation and using CSS, and provides examples to illustrate the topics.

### 3.1 A different approach

When a browser requests a page and accesses a Web site, it announces its identity by providing User Agent information. Based on this, the content of the site can be tailored for that particular browser. For Web designers, customizing the Web site for a particular browser is often an unwelcome but necessary task. It is good practice to try to design a site that works well in all major browsers without having to fall back on “fixes.” The situation has improved in recent years, but skillful coding is still needed to achieve the correct results across standard desktop browsers and platforms.

With mobile platforms, the situation has perhaps been even more difficult. Mobile browsers have technical limitations concerning display size, processor speed, and network latency. In addition, users may be billed for the amount of data that is transferred during a session.

There are different methods for tailoring content for browsers. Content can be modified on the server side (before sending the data to the mobile client device) or on the client side (in the actual mobile browser). The solution might also be a combination of the two.

A server-side solution is achieved with techniques such as the use of Java™, PHP, and ASP programming languages. Based on the information supplied by the browser, the server can modify the content and structure of the requested page. For example, the server can automatically strip large images, restructure the content to better fit the mobile display to ease scrolling, and remove parts that are not supported by the client browser. As a result, the user does not have to download extra content over a limited bandwidth, the browsing becomes faster, and the amount of transferred data is kept to a minimum. The rendered page may fit the mobile device screen, but it can look very different from the “Full” site layout.

There have been problems where the server pushes “Mobile” content to the Nokia Web Browser, which leaves the user with a much reduced version of that Web content. For any developer of content targeting mobile devices it is important to include a clear link to the “full” HTML page.

A practical example of the server-side approach is used by the **Opera Mini** browser. Opera Mini uses a remote server to pre-process Web pages prior to sending them to the mobile device. An engine on the proxy server reformats and compresses the original Web page before sending and may adjust the images on the page before passing the data forward.

Another example of a server-side solution is **Google Mobile**. The Google [mobile search service](#) is formatted to fit a small display. After performing a search and selecting a link from the results, the Google server reformats the Web page to fit a mobile display. For a demonstration, type “nokia” on the mobile search service and select the first link. Then use the normal search and compare the results. Both of these approaches are successful in delivering mobile content, but are a solution to a problem that does not exist with the Nokia Web Browser. It can render the full Web content without resorting to server portals or filtered pages.

Server-side solutions require active upkeep and development. Because there are several different mobile browsers (and different versions of the same browser), all with different capabilities and restrictions, the server administrator has to constantly test and update the modification scripts. There

are useful open source solutions that are available to help query and return the correct information (for example, see the [Wireless Universal Resource File](#)).

In general, implementing server-side solutions requires more initial work and assumes a level of server access and expertise that goes beyond the simple Web design skill set. For many smaller site owners, server-side scripting is something to be avoided, and many hosting companies charge additional fees for providing this service.

Adopting a client-side solution involves a different approach. The requested page is sent “as is” to the mobile device, and any possible tailoring of the content is done in the browser rendering engine. The Web developer can include JavaScript™ and alternative CSS styles to reorganize the content that is actually processed on the browser.

The Nokia Web Browser aims to deliver a full, desktop-like browsing experience. Web developer can be sure that the user will see the page as it is designed, without external reorganizing of the content. From the user’s point of view, this approach is quite simple and natural: a Web site has the same look and feel on all browsers. Branding and “sense of location” — both important elements to the success of a Web space — are properly maintained.

However, it is still advisable to take the mobile browser limitations into consideration while developing Web sites. The key is making the sites easy to adjust to the mobile device, without a major overhaul of the entire site or maintaining a separate site for mobile use.

### 3.2 W3C Mobile Web Best Practices — Developing usable Web sites for mobile browsing

The World Wide Web Consortium (W3C) has created basic guidelines for Best Practices for delivering content to mobile devices. Although the W3C practical guidelines are a useful starting point, they do not cover all the issues relevant to the Nokia Web Browser, and many of the restrictions do not apply to the Nokia Web Browser as such. The following link gives a brief summary of the W3C mobile guidelines: [Summary — Mobile Web Best Practices 1.0](#).

The principal objective of the W3C guidelines is to improve the user experience of Web browsing when accessed from mobile devices. The context of mobile use, device capability variations, and bandwidth issues may affect the presentation of Web pages. Although browser and device usability are important (for reading, navigation, and interaction with content), the focus of the guidelines is primarily on Best Practices for improving the usability of a Web page with regard to the following issues:

- **Presentation:** Many Web pages are laid out for presentation on desktop-size displays. Because of the limited screen size of mobile devices, the subject matter of the page may require considerable scrolling to be visible, and users may not get immediate feedback as to whether they are accessing the right content. Creating a mental image of the site quickly is important; this should be assisted by adopting a consistent style.
- **Input:** Since mobile devices often have a very limited keypad, lengthy URIs are difficult to type correctly and forms are hard to fill in. The “back button” functionality built into the Nokia Web Browser eases navigation considerably. However, it still may be difficult to recover from errors, broken links, and so on.
- **Bandwidth and Cost:** Mobile networks can be slow compared to fixed data connections and often have a measurably higher latency, which can lead to long retrieval times. Also, the user may be paying separately for mobile data transfer. The browsing experience may not be satisfactory if Web pages contain content that the user has not specifically requested. In the mobile world, this extra material contributes to poor usability and may add considerably to the cost of retrieval.
- **User Goals:** Mobile users typically have more immediate and goal-directed objectives than desktop Web users. Their intentions are often to locate specific pieces of information that are

relevant to their context, and they typically are less interested in lengthy documents or leisurely browsing.

- **Advertising:** Some mechanisms that are commonly used for presentation of advertising material (such as pop-ups, pop-unders, and large banners) may not work well on small devices. If advertising exists, the user experience of the site as a whole, including advertising, should be as compact and effective as possible.
- **Device Limitations:** Rendering Web pages— for example reflowing pages, laying out tables, processing long and complex style sheets, and handling invalid markup — can stress the CPU. Because of this, page rendering may take a noticeable time to complete. Some devices have limited memory available for pages and images, and exceeding their memory limitations can result in incomplete display as well as other problems. For Nokia devices, refer to the [Device Feature Tables](#) for more information on precise memory and cache sizes on individual devices.

One clear benefit of mobile browsers is that the full Web goes wherever the user goes. Users can access a Web site immediately, whenever they want, within the immediate context that prompted the need. With the integration of new technologies such as location, imaging, voice recognition, and touch screens, the range of possibilities for services and functions is growing rapidly. Also, particularly in developing countries, mobile devices are more common than desktop computers. Web-capable mobile devices will play an important role in deploying widespread Web access.

### 3.3 Development and testing environments

During the process of Web site development, the Nokia Web Browser can be considered as just another browser upon which to test and verify the design and functionality. However, testing should be incorporated in the early stages of development. Developers should run a quick test on the mobile browser every time they make large layout or design changes or add advanced functionality to the site. If developers have a defined set of test cases for Web site testing, they should check their applicability with the mobile browser and modify them or add new ones if necessary.

During development and testing, developers may discover that there are some features that simply cannot be displayed with the current version of the Nokia Web Browser. If these features and functionalities are essential to their Web site, they should not dismiss them simply because of this restriction — browser features are being developed continuously (for example, version 3.1 of the Nokia Web Browser adds native support for Macromedia Flash Lite from Adobe), and the problem may be solved this way. However, they should also consider the impact and possible solutions for the problem. Does the lack of a particular feature cripple the site completely? How can this situation be avoided?

The normal Web development approach can be used for both desktop and mobile devices. For testing and verification, there are the following alternatives:

- **“Quick and Dirty.” Desktop browser/IFrame:** A quick testing method that does not require special tools or access to a particular device is to simply use a resized version of a common desktop browser. Alternatively, developers can use a tailored IFrame for presenting content. IFrames are an inline HTML element that allows another HTML document to be embedded inside the main document. There are also online services for emulating mobile device behavior; for example, see [www.yospace.com](http://www.yospace.com).
- **“Heavy Duty.” SDK emulator:** Software Development Kits include emulators for testing browser behavior. The emulator provided in [S60 Platform SDKs for Symbian OS, for Java](#) does provide all Nokia Web Browser functionality (excluding MiniMap and text rendering), but is a smaller, lighter download. The [S60 3rd Edition SDK for Symbian OS, for C++](#) is a larger package, but it provides full browser functionality. Web pages loaded as local files need to be placed in a specific location and opened with the SDK file manager. For the Java SDK the default local file location is

C:\S60\Devices\S60\_3rd\_MIDP\_SDK\bin\epoc32\wincsw\c\Data. For the C++ SDK the default location is C:\Nokia\Epoc32\wins\c\nokia.

- **“The Gold Standard.” On-device testing:** The best way to test the full functionality of the design is to access the material over a live connection. This allows the developer verify the site design and functionality from the mobile display, using the actual device keys and input methods. On some devices it may be possible to test the files locally by transferring them to the memory card.

For rudimentary testing purposes, [a sample IFrame](#) is provided on the Forum Nokia Web site. Developers can use the IFrame to quickly see how their site would look on a mobile browser display. The IFrame size is defined to be the standard QVGA (320x240 pixels) size. Note that while an IFrame is good for quick testing and demonstration purposes, it should always be augmented with at least one of the other methods listed above.

Taking mobile browser issues into consideration does not have to mean that existing processes are turned upside down — rather, these browser issues can be adapted to fit into the development and testing methods and processes already used in the developer's organization.

## 4 Basic Web development issues

This chapter discusses some basic Web development issues from the Nokia Web Browser perspective.

This chapter works in tandem with an example Web site that illustrates and supplements the information presented here and provides code examples. You can view the Web site side by side with a normal desktop browser and the Nokia Web Browser on your mobile device. You are free to use any pieces of code to develop your own site.

- [Open the example site](#) (see [Article 3](#))
- [Download the source files as a zip package](#)

### 4.1 Getting started: Building simple HTML structures

Due to the True Web layout rendering, the Nokia Web Browser displays complete Web pages and renders any normal HTML structure without difficulties. When starting development work, build the HTML structure in a logical order and make sure that the content is semantically marked. Use HTML to create the structure for the Web page and keep the code as simple as possible. Cascading Style Sheets (CSS) can be used to fine-tune the layout and visual appearance.

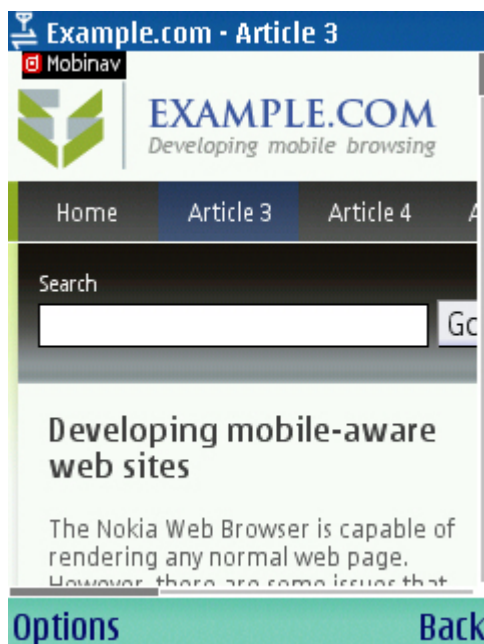
Because Web pages are transferred to a mobile device over a limited network connection, it is important to write efficient and condensed code. Make a habit of thinking economically — small adjustments and optimizations here and there result in lower total file sizes and faster download times for the mobile browser.

Remember to test Web pages with a mobile device early on in the development process. One testing method is to remove all style information and switch off the graphics; if the Web site is still readable and functional at a basic level, it will probably survive any browser. Use different validators (such as the World Wide Web Consortium [W3C] service at <http://validator.w3.org/>) for ensuring compliance to markup standards.

### 4.2 Utilizing the “hot spot”: Designing effective navigation

When the Nokia Web Browser opens a Web page, the initial view is on the upper left corner of the page. This “hot-spot” area is very important, as it is the first thing the user sees and focuses on. The most important objects, such as navigation and basic site information, should be placed in or near this hot spot. The initial view should tell the mobile user where he or she has landed (company name or logo, site name, or short summary) and how to navigate further on the site.

On a mobile device, the hot-spot real estate is limited. For example, on the standard QVGA display, the hot spot covers an area of 240x320 pixels (or 320x240 pixels in landscape mode). It is not necessary to fit everything within the hot spot, but it is recommended to at least start the elements from the hot spot. For example, a horizontal or vertical site navigation element can continue and trail off-screen, as long as the user is able to quickly recognize the element and follow it. Make sure that the navigation element is visually consistent — a background color or a shape acts as a visual clue and helps the user follow the element.



On the example site, the initially displayed hot-spot area on the Nokia Web Browser includes a number of critical components:

- Company logo and site name
- Navigation
- Search field
- First heading text of the content
- Special mobile navigation tool (explained below)

In the example site, the navigation bar is implemented with a standard, unordered list, using text elements and CSS to provide a mouse over effect.

Tip: To place focus on a specific link after the page loads, add a special “#start” fragment tag on the link element. This is an effective way to direct the user’s attention to a specific place on the page.

#### 4.3 Supporting the mobile user with navigational aids

Using the hot spot for navigation placement is good for the mobile user, but not always possible in creative site design. The example site introduces a special navigational aid: a mobile navigation tool that can be placed in the hot spot. Clicking on the link opens a supportive navigation list from which the user can quickly navigate elsewhere within the site or within the page.

The tool is implemented with a script written in the JavaScript™ programming language. Currently, the script dynamically scans the document for elements marked with id=“navigation” and fetches the navigation content. The script also creates links for the Heading elements used within the current page. Because the script fetches the content dynamically from the page contents, there is no need for constant upkeep or maintenance of a separate navigation list: the navigation tool is automatically updated as long as the new elements follow the structure used by the script.

The script can be customized to include the necessary elements required by the site design and the designer. Simply examine the script source code and comments within the script for help. Note that the tool should only support normal navigation, not replace it completely. Some users may have switched off the script, in which case the presence of traditional navigation is essential.

The traditional navigation aids are very useful for mobile browsers as well. For example, the traditional “Back to top” text link proves very useful with mobile browsers. Remember to provide alternative text description for important graphic elements, as some users may have turned off images when browsing with a mobile connection.

#### 4.4 Using CSS for presentation

The Nokia Web Browser supports CSS. Use CSS to separate the layout and visual appearance from the content and structure provided by HTML.

With the Nokia Web Browser, there are a couple of issues that should be taken into account: the decision where to place the style information and using the media type to tailor presentation format.

##### 4.4.1 Embedded versus external CSS

CSS information can be attached to a Web page using several alternative methods (or a combination of methods):

- External CSS: Style sheet information is stored in an external file, allowing the possibility for several HTML pages to refer to the same style sheet.
- Embedded CSS: Blocks of style sheet information are included in the HTML document itself, in the `<head>` element before the actual content in the `<body>` element.
- Inline CSS: Style information is written to individual elements.

When loading a Web page, the Nokia Web Browser first displays the HTML Web page without including external files (such as style sheets or scripts). When the document is fully loaded, the page is rendered again, now including the external files. Therefore, if the CSS information is included in the HTML document (embedded or inline), it is already included in the first rendering phase. External CSS causes a short duration of a blank screen before the second rendering phase; however, the external CSS file is cached to the browser and can be utilized faster with subsequent pages using the same style sheets.

##### 4.4.2 Tailoring presentation format

If deemed necessary, it is possible to tailor the presentation of content for the mobile screen. A different CSS can be used for a full-size, desktop display and another for the mobile display. (Chapter 5, “Site layout design,” describes how this can be achieved by using a JavaScript “sniffer” that detects the screen size.)

CSS allows the use of different media types for formatting the content differently, for example, by using the “handheld” attribute. The Nokia Web Browser 3.0 does not support that media type distinction, but it may still be a good idea to label any alternative mobile CSS styles with the media type, because support may be added in later browser versions.

#### 4.5 Font size recommendations

Different fonts and typefaces are re-rendered in the browser to use the sans-serif style device font.

For the standard QVGA-size display, the following font sizes are recommended:

- Heading 1: 18px, bold
- Heading 2: 16px, bold
- Heading 3: 14px, bold

- Heading 4: 12px, bold
- Body text: 12px, normal

The choice of font styling can often play an important part of a Web design, and to have this control taken away by the browser might at first seem a serious drawback. However, the twin factors of coping with small screen size, coupled with the usability advantage offered by sans serif screen fonts makes the feature more understandable.

## 5 Site layout design

This chapter examines issues regarding site layout design and some effective tips that reduce effort from the mobile user.

This chapter is accompanied with an example Web site to illustrate and supplement the information presented in the chapter and to provide code examples. View the Web site side by side with a normal desktop browser and the Nokia Web Browser on your mobile device. You are free to use any pieces of code to develop your own site.

- [Open the example site](#) (see [Article 4](#))
- [Download the source files as a zip package](#)

### 5.1 Detecting the mobile browser

The Nokia Web Browser renders Web pages in their full layout and utilizes a MiniMap feature for efficient use of a large Web page. However, sometimes it may necessary or feasible to recognize that the user is using a mobile browser. This information can be used for various purposes, for example:

- Providing an alternative CSS stylesheet (for an alternative layout or text sizes, for example)
- Omitting elements or objects from the page shown on the mobile browser
- Providing an alternative navigation
- Offering lighter versions of graphics to reduce download speeds

On the example site (section [Article 4](#)), the layout of the page has been optimized to support simple vertical scrolling when accessing with a mobile browser. The user is notified of the alternative layout and a control to switch to the full layout is provided. On the example site, the functionality is implemented with JavaScript objects (`screen.width`, `screen.height`) that detect if the user is using a QVGA-sized browser. After the detection, the browser is instructed to use an alternative CSS stylesheet.

Upon initial page loading, the page uses the `getScreenSize` function to check the display size on the mobile device:

```
if (getScreenSize(240, 320)) {
    generateMobinav();
    setActiveStyleSheet('Mobile');
}
```

If the mobile display size corresponds to the setting (in this case, QVGA), the page generates the additional navigational tool (function `generateMobinav`) and automatically switches the page to use the Mobile stylesheet. The function below loads the alternative stylesheet (note that the name of the stylesheet is used as an argument and originates from the function call).

```
/* =v==== STYLE SHEET SWITCHER =====v= */
function setActiveStyleSheet(title) {
    if (document.getElementsByTagName) {
        var i, a, main;
        for(i=0; (a = document.getElementsByTagName("link")[i]); i++) {
```

```

        if(a.getAttribute("rel").indexOf("style") != -1 &&
a.getAttribute("title")) {

            a.disabled = true;

            if(a.getAttribute("title") == title) a.disabled = false;
        }
    }
    return false;
}

```

After this, the alternative stylesheet declared in the HTML page is used:

```

<link rel="alternate stylesheet" href="styles/mobile.css"
type="text/css" media="all" title="Mobile">

```

Note that if you decide to use an alternative layout for the mobile browser, let the user know it and provide them with the possibility to switch to the full version.

## 5.2 Two-column layouts

Many Web sites utilize a two- or three-column layout for presenting the contents. When using several columns, remember that on the Nokia Web Browser wraps text elements for the maximum available screen width, in order to better facilitate vertical scrolling.

An example of a two-column layout is presented on the example site for this chapter (see [Article 4](#)). An example featuring a three-column layout is presented in [Article 5](#) on the example Web site. Note that using a multi-column layout on a small display can be challenging. Consider what kind of content you want to place on the left-side column: remember that the mobile user first sees the upper left corner of the page, and the initial view should include both a visual clue of the navigation as well as a first glimpse of the most relevant content.

## 5.3 Tips for making mobile browsing efficient

When considering the mobile user and the use context, viewing and using Web pages should be as efficient as possible. In an optimal case, only the absolutely necessary information is presented and the need for user input is minimized.

Some useful tips for efficient design are presented below. For practical examples, refer to the example site and the source code.

- **Expanding content areas:** allow the user to browse through the content and read what he or she is really interested in.
- **Efficient forms:** dynamic drop-down menus with pre-selected alternatives reduce the need for user input.
- **Cookies:** re-use information that is already entered by the user; do not make the user enter the same information over and over again.
- **Shortcuts:** Provide shortcuts, such as a Back to top –link, for the user.

## 6 Media formats

This chapter takes a look at a few advanced media formats.

The chapter is accompanied by an example Web site that illustrates and supplements the information presented in the chapter and provides code examples. You can view the Web site side by side with a normal desktop browser and the Nokia Web Browser on your mobile device. You are free to use any pieces of code to develop your own site.

- [Open the example site](#) (see [Article 5](#))
- [Download the source files as a zip package](#)

### 6.1 Video content in the mobile browser

The example site demonstrates the behavior of video content in the Nokia Web Browser. After the link has been clicked on, the `.rm` (RealVideo 8) video on the site is first downloaded to the device and then automatically opened in the video player. The Nokia Web Browser shipping with S60 3rd Edition does not have the capability of playing inline video; the Feature Pack 1 browser has added support for inline video playback without launching an external player.

The files can be in any video format supported by the S60 platform, including the following:

- 3GPP
- MP4
- Real Video7, 8, 9, 10

The Video Plug-in implemented in S60 3rd Edition, FP1 does not support locked content that is protected by a Digital Rights Management (DRM) scheme.

In the future, embedding video content on the Web page will be easier. Flash Lite 3, announced by Adobe in early 2007, will include support for video and bring mobile devices one step closer to the browsing experience provided by desktop browsers.

### 6.2 Designing Flash for the mobile browser

Flash content, such as animated banners, is an everyday sight on Web pages. While Flash support on desktop browsers is widespread, mobile browsers may be unable to show the same information, effectively losing some of the site's content. Possible solutions to this problem include:

- Deciding not to display Flash content on the mobile display by removing the elements
- Substituting static images for Flash content
- Using Flash Lite

Nokia Web Browser on S60 3rd Edition, FP1 adds support for Flash Lite, making it possible to display embedded Flash content on the mobile display. When designing content, developers should consider the advantages of using Flash Lite instead of full Flash. In many cases, the functionality provided by Flash Lite is sufficient, and the same Flash Lite content can be displayed on both the desktop browser and the mobile browser. However, Flash should not be used for critical components such as navigation.

### 6.2.1 Substituting Flash content

The ideal way of substituting Flash content in browsers that do not support Flash would be the use of the `<object>` element in the HTML markup. However, the Nokia Web Browser currently does not support this method and an alternative solution is needed. Note that the browser is in development and the issues are addressed in future releases.

The example site features a JavaScript™ function for checking the Flash support on the browser and substituting content. Try viewing the example site on different browsers — the bottom label of the banner on the left-hand side reads “Flash,” “Flash Lite,” or “Picture,” depending on what is supported by the browser.

First, the function checks if the device platform is the S60 platform and if the browser supports Flash Lite:

```
function checkFlash() {
    if(navigator.platform && navigator.plugins &&
document.getElementById("flash")) {
        if(navigator.platform == "Series60") {
            for(var i = 0; i < navigator.plugins.length; i++) {
                if(navigator.plugins[i].name == "FLASHLITE") {
                    var flashEnabled = true;
                    break;
                }
            }
        }
    }
}
```

Next, if Flash Lite is detected, the browser is instructed to replace the Flash object with a secondary Flash Lite object. If the browser does not support Flash Lite, a static image is displayed as the final alternative:

```
        if(flashEnabled) {
            document.getElementById("flash").innerHTML = "<object
type='application/x-shockwave-flash' data='flash/ad_skyscraper_lite.swf'
width='120' height='600'><param name='movie'
value='flash/ad_skyscraper_lite.swf' /><img src='flash/placeholder.jpg'
alt='Picture of Nokia N77' /></object>";
        } else {
            document.getElementById("flash").innerHTML = "<a
href='http://europe.nokia.com/phones/n77'><img
src='flash/placeholder.jpg' alt='Picture of Nokia N77' /></a>";
        }
    }
}
```

### 6.3 Embedding audio

From a design and usability point of view, embedding large audio files into HTML pages is not recommended. However, there may be situations where this feature is required. Please bear in mind that there may be a considerable time lag before the audio begins to play. It is usually a good idea to allow the user to switch off the sound.

With the Nokia Web Browser, embedding audio for playback is straightforward, as both “embed” and “object” methods work.

The Browser Audio Plug-in (BAP) downloads and plays audio files when the user selects them. BAP also plays background sound embedded in a Web page. The volume level is stored in the Central

Repository and can be configured by the user. The files can be in any audio format supported by the S60 platform, including the following:

Real Audio Voice/ MP3/ MIDI/MPEG4/ AMR-NB/ AAC/ AMR-WB/ EAAC

In addition, the FP1 browser also adds support for WAV file playback.

When the current page defines the audio to be played, the browser begins to play it automatically. If multiple windows are open, only audio from the top window is played. Audio requests from other windows are ignored.

If the user tries to open an audio file while another audio file is playing, the browser stops the initial file and closes it. The newly requested file is played instead.

#### 6.4 Embedding SVG

Embedding Scalable Vector Graphics (SVG) is also a simple process for the Nokia Web Browser, although plug-ins may be required for some desktop browsers. Both static and interactive SVG content can be displayed, however, since interaction usually requires JavaScript, there are issues of portability across platforms. SVG is an open source, lightweight graphic solution that is well suited to mobile devices.

## 7 Progressive enhancement — the future Web design philosophy

The diversity of Web technologies and users will inevitably increase in the future. The dominance of one major browser application will decline as Web users access their favorite content with a multitude of applications and from a variety of mobile devices, such as laptops, PDAs, and mobile communication devices. The profile of the Web user will also be upgraded: in the future, the new Web user may use a mobile device as his/her primary or only access to the Web. This evolution requires a rethinking of the overall Web design philosophy.

The traditional Web design approach is the graceful degradation strategy. The design driver is that Web sites are developed for the latest technologies and newest browsers; at the same time, there is the challenge of trying to keep the site somewhat presentable while losing features (“degrading”) in older browsers. In practice, this approach is often unsuccessful due to lack of development and testing resources and the attitude that the user will eventually update to the latest browser software anyway. The core assumption is that browsers will progressively get faster and more powerful, incorporating support for all the available technologies.

Progressive enhancement reverses the design strategy by focusing on the lowest common denominator. Everything begins with a basic markup document to which the designer adds layers (“enhancements”) of presentation and behavior, using technologies such as CSS, JavaScript, or Flash. All the progressive elements are externally linked so that browsers with limited capacities do not have to handle code that would only slow them down or clog a limited network connection.

The principles of progressive enhancement are:

- Basic content and functionality should be accessible with any browser or connection type; an enhanced browsing experience is provided to advanced browsers;
- Sparse semantic markup and separation of content and presentation;
- Enhanced presentation and functionality provided with layered and unobtrusive elements, such as externally linked CSS and scripts;
- Respect for user preferences.

## 8 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](#).