Device and feature detection on the mobile web

04 Jul 2010

Using device and feature detection to improve user experience on the mobile web

This article introduces the topic of device and feature detection as it applies to the design and development of mobile websites. The article discusses common mobile design approaches, why and how to detect device properties, and the options available once a device or property has been identified. The article includes sample source files, which can be downloaded or viewed online. To access the online version, please visit http://www.developer.nokia.com/device-detection/ using a mobile or desktop browser. The page demonstrates various device detection techniques, and provides a sample of the data returned from them. Viewing the page on a variety of devices will illustrate differences in browser capabilities. Note: The list of properties returned is not a complete list of what is available. Please see each respective technology for full information.

Introduction to device detection

Device detection enables identification of device properties and characteristics, with the aim of determining the best content, layout, markup or application to serve to a given device. Device detection enables developers to identify device features and characteristics such as screen size, browser type (or version), media support, and the level of support for Cascading Style Sheets (CSS), HTML and JavaScript technology.

Why use device and feature detection?

The ability to identify a device, browser or feature enables the developer to perform a wide number of relevant actions. These may include:

- Serving a mobile formatted site rather than the desktop site;
- Swapping style sheets to adapt layout and content to the device's specific HTML, CSS and JavaScript technology capabilities;
- Modifying the amount or nature of content to suit the device capacity. For example, a JavaScript based slide show may be served to a higher-end device, and a simple list served to a lower-end device;
- Serving smaller or larger images to suit screen size, or swap SVG graphics for bitmap images on less capable devices;
- Enhancing functionality on more capable devices by applying progressive enhancements (based on feature and object detection);
- Increasing or decreasing font size, margins and padding to scale the actionable areas on a touch device.
- Serving different sizes and/or formats of media such as video to suit the device codecs or capabilities.

Approaches to mobile site design

Deciding the best implementation approach during development of a mobile web presence can be challenging and should be based on examination of a variety of factors. These may include:

- Your users. What devices do they use? Do they already use the mobile web? What specific use cases would you like to address or enable using your mobile presence?
- The implementation costs and resource requirements (including long-term maintenance).
- The specific strategy and business goals of your mobile presence.
- The approach taken by your competitors.

There are several common approaches to providing web-based mobile services, each with distinct advantages and disadvantages. These approaches can also be combined to improve users' overall experience.

Do nothing

Serve your desktop site to all devices and let the browser render what it can on its own. While this is an option, there are quite a few disadvantages:

- Serving a desktop site to all users does not address the fact that mobile users often have very different behaviours and expectations when accessing a site on a mobile device.
- While higher-end mobile browsers may be able to successfully render desktop-formatted content, browsers on lower-end devices will certainly struggle with this. On these devices, some content may not render, and attempting to load large amounts of content may crash the browser.
- If you do not take the necessary steps to support mobile devices, your desktop web site may be mobile-adapted (aka transcoded) by the network over which it is delivered. Transcoding dynamically reformats the content for mobile browsing and reduces data usage.

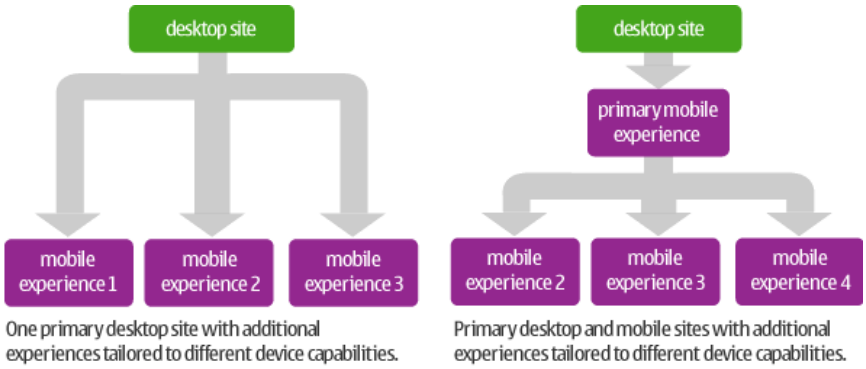
The "do nothing" approach is still quite common, however is not recommended as a long-term solution.

Providing a generic mobile site

An excellent first step in extending your site to mobile devices, is to create a generic mobile site which presents content formatted with the small screen in mind. This will enable you to provide a superior browsing experience, tailored to small devices, and to deliver content and functionality that has been adapted to the mobile context. It will not however take into consideration the capabilities of each mobile device or browser. Given the ever-increasing number and diversity of devices on the market, it is also not viable to design multiple mobile sites—each to address a different group of device capabilities.

Designing with mobile and adaptation in mind

The recommended approach is to design with mobile diversity in mind from the very beginning, and adapt your desktop site (or a separate mobile site) to suit a variety of device capabilities. This in effect, simulates the existence of a large number of sites, each serving the best content for a given device or range of capabilities.



Content adaptation and device grouping strategies

Delivering a great experience to multiple devices will typically involve adaptation of both content and functionality. The end goal: To deliver the best content, functionality and overall experience for a specific browser or device. In doing so, there are many parameters, that must be considered and each will reveal potential adaptation targets.

Browser parameter	Relevant factors	Potential targets for adaptation
Design and layout capabilities	* How well are HTML and CSS supported?	*Use of floats, relative and absolute positioning, positioned background images or CSS sprites. *Use of CSS3 selectors such as rounded corners.
Scripting	*How well are JavaScript technologies and libraries such as jQuery supported?	*Features dependent on JavaScript and related libraries such as jQuery. *DOM manipulation e.g. adding, modifying and removing classes, content, styles and so on. *Features dependent on more recent JavaScript 1.6 array functions.
Browser	*Is there support for platform-specific APIs or HTML 5 APIs such as location?	*Implementation of features that take advantage of device, browser or platform specific APIs. *Which DocType and meta-tags to serve.
Hardware	*What is the screen size and dpi? *Does the device support touch? *Is there support for both landscape and portrait orientations? *How does this device rate when it comes to performance? *Is this browser subject to extreme memory limitations? *Are there limits to the size of the media that can be downloaded onto the device?	*Default font size, and values used for padding and margins, which will affect the size of controls for touch manipulation. *Use of large images (or images in general), which impact both browser memory and network usage. *Use of video and other memory-intensive media assets. *Use of CPU or GPU-intensive features such as on-screen transitions or animations. *Amount of content served on a given page or view, and the ability for users to request more content (through pagination or Ajax-based mechanisms).
Media	*Does the device support Flash, video, audio or document formats such as PDF? If so, what formats?	*Type and version of media served.

Given the number of parameters to consider, the task may appear daunting. After all, it's not possible to design content adaptations for every single browser or device. A common solution is to create device categories or groupings—each representing a set of parameters that you have decided to support. In effect, the groups map to different "experiences" that you will serve to different users.

Device grouping

The following table illustrates a sample device grouping and the possible content adaptations that may apply to devices within this group.

Group	Characteristics	Content adaptation characteristics
High-end touch smartphones	Eg. S60 5th Edition, N900	*Strong support for XHTML, JavaScript and CSS. Some CSS3 support. Some jQuery support. Support for DOM functions. *Large touchscreen display (over 360 pixels wide in portrait orientation). *Strong data table support including setting column width, use of background colours and manipulation of content using JavaScript technology. *Strong caching support. *Support for HTML5 capabilities. *Extra JavaScript APIs or vendor-specific CSS. *Serve large images, maximum width 340 pixels. Use occasional float-based layouts to improve information design. *Progressively enhance styling, functionality and behaviour using JavaScript. *Enable users to customise content choices. Save these preferences using cookies. *Use of CSS3- or HTML5-specific capabilities.
High-end nontouch smartphones	E.g. S60 3rd Edition FP2, Series 40 devices using the WebKit browser	*Strong support for XHTML, JavaScript and CSS. *Large display (typically 240 pixels wide in portrait orientation). Devices also often support landscape mode. *Strong data table support including use of background colours. *Strong caching support. *Serve mid-sized images, maximum width 220 pixels. *Progressively enhance styling and behaviour using JavaScript but avoid processor-intensive transitions or animations. *Enable users to customise content choices. Save these preferences using cookies.
Mid-range smartphones and feature phones	E.g. S60 3rd Edition FP1, Midrange Series 40 devices	*Supports XHTML MP and basic CSS. *Unreliable JavaScript capabilities. *Displays range from 176 pixels to 240 pixels wide in portrait orientation. Landscape orientation support is rare. *Basic data table support. Table formatting unreliable. *Unpredictable caching. *Serve midsized images. Maximum width 170 pixels. *Remove background images and replace with native colour fills. *Use data tables sparingly. Consider replacing these with lists where possible. *Implement pagination for larger articles.
Low-end devices	E.g. Legacy S60 and Series 40, Series 30 devices	*Supports XHTML MP. Basic or non-existent CSS capabilities. *No support for JavaScript. *Displays range from 120 pixels to 240 pixels wide in portrait orientation. Landscape orientation support is rare. *Unreliable support for data tables. *Assume no caching. *Serve smallest images. Maximum width 100 pixels. Consider omitting images altogether in certain contexts.

*Remove all backgrounds (colours and images).
*Remove all data tables. Substitute for lists where possible.
*Reduce the amount of content within each view and within the entire site. Paginate longer content and adapt drill-down hierarchies.

Important: Today's smartphone may be considered on par with a featurephone in a few years time. Remember to review and/or update your device categories regularly as new devices and capabilities are released into the marketplace. It's also important to remember that there is no one method to determine device groupings. For example, if your content serves lots of video, your groupings may largely revolve around support for this media format.

- Your high-end category may be split into two groupings: touch devices with a large display that support HTML 5 video, and similar devices with no HTML 5 support.
- A midrange grouping would include devices that can only accept smaller video files.
- A final low-end grouping may include devices, which are completely excluded due to a complete lack of video support but may receive an alternate form of content where possible.

Content adaptation

As indicated in the previous table, adaptation can impact content in many ways. Here are some recommendations:

Adjust the quantity of content

- The amount of content should be increased or decreased to suit the device. On less robust devices, content should be paginated to reduce the amount on each page and to provide users with control over how much content they wish to download.
- The number of choices in navigation lists and menus should be increased or decreased to suit the device. Determine which menu items are most important, most frequently visited, or can be subdivided and paginated. You can present the most important items first, and then provide the user with the option to download more menu items or to click through to a separate page. For example, a shopping site might start with three categories: menswear, ladies wear and shoes. Clicking each link would serve a new page with further choices for each category.
- Examine the content on your site to determine what is most necessary for each mobile experience. Serve the most necessary content to simpler devices and progressively increase the content for midrange and high-end devices. In all cases however, keep in mind that users may be paying for every byte of content. Serve only content that has been requested by the user.

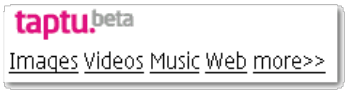


Figure: The Taptu site contains many categories however only displays the most popular topics as primary navigation along with an option to click to view the full list.

Adapt image size

- Avoid serving purely decorative images (unless they are integral to the brand message) and adjust the size of images to suit the device. This is one of the simplest forms of adaptation to execute but can go a long way to improve download speeds and reduce memory issues and the potential cost for users—all of which result in a better user experience.
- Services such as tinySrc, which work in conjunction with a device database are available to dynamically and automatically swap images on different devices. Images can also be substituted using client-side JavaScript technology however server-side approaches are recommended as they ensure users only download the graphics that are most suitable for the device.
- When specifying images to suit different devices, keep in mind the impact of screen size, colour depth and resolution (ppi) on your images. A graphic that looks great on a larger device may be illegible if simply scaled to suit a small screen. Where possible provide alternate images that have been specifically chosen or cropped to suit a particular size.
- Remember as well that if you do not provide alternate images of your own, your user's network operator may automatically (and often aggressively) compress or transcode the images to reduce data consumption. This can lead to pixilated images and a poor experience.

Figure: A device's ppi can dramatically affect the legibility of an image.

Figure: While the photos on the New York times website are legible at multiple sizes, alternate versions of the banner advertisement have been provided to ensure legibility is maintained at all sizes.

- Through changes to CSS, adjust the size of text and UI controls (such as buttons, navigation bars and form input elements) to suit the device. On devices with smaller screens, these controls merely have to be legible but on touch devices they should also be large enough to tap or manipulate. There should also be enough space between the controls so that users can easily tap their option of choice.
- Note as well that on older devices with cursor navigation, the exact pointer movement and location may be hard to control. It is therefore important to leave adequate space between links and UI controls to ensure the user can successfully select them.

Are there situations where content adaptation is not appropriate? The short answer is no. Unless your desktop site is extremely simple, well-implemented content adaptation will always improve the experience of your mobile users. It's important however to carefully consider which features and functionality to implement on mobile. Here is an example:

You are an international airline brand. Your desktop web site provides the ability to search through thousands of flights and partner hotel listings. The site uses Ajax to display query and search results on the fly, and dynamic maps to further illustrate the products on offer. Offering all this functionality on a mobile device will not be realistic however a subset of this functionality could be offered as follows:

- High-end touch smartphones: Provide flight status, timetables, mobile check-in and the ability to complete relatively simple transactions (such as hotel reservations) on mobile. Where possible, enhance the experience using mapping, video clips or photographs of the destinations. Detect the user's location then provide a local phone number enabling completion of more complex transactions by phone.
- High-end non-touch smartphones: Provide flight status, timetables, mobile check-in and the ability to complete relatively simple transactions (such as hotel reservations) on mobile. Where possible, detect the user's location then provide a local phone number enabling completion of more complex transactions by phone.
- Midrange featurephones: Provide flight status, timetables and mobile check-in capabilities. Enable users to register for SMS-based flight alerts. Provide a local phone number enabling completion of more complex transactions by phone. The user's location would, in this case be detected via triangulation or manually specified.
- Low-end devices: Provide a simple flight status lookup and enable users to easily register for SMS-based flight alerts. Provide a local phone number enabling completion of more complex transactions by phone. The user's location would, in this case be detected via triangulation or manually specified.

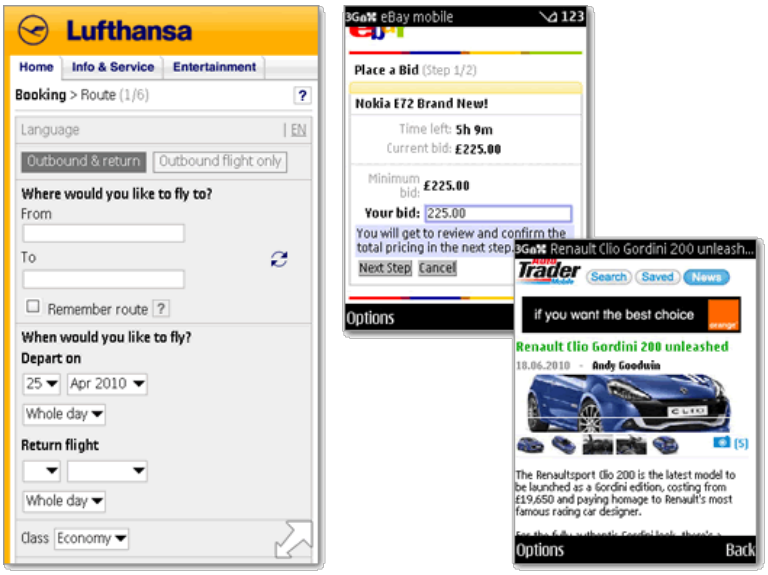


Figure: When well designed and optimized with users and device capabilities in mind, mobile sites can include sophisticated functionality.

Minimising the need for adaptation in the first place

When you serve content to mobile devices, three factors will immediately impact your design—the smaller screen size, the often-flexible orientation (which also impacts screen size), and the variations in rendering ability and capability support among browsers. How you choose to structure and create your default HTML markup can dramatically impact the amount of additional content adaptation you will have to undertake to address these issues.

Write clean, well-structured HTML markup

Structure your pages, templates and content using simple, standards-based HTML. Use native HTML elements to specify as much of the content as possible, and only use divs (combined with classes or IDs) to create major divisions within the page such as the primary content area, navigation area or footer.

For example, instead of creating a div called 'header' for the title, use an h2. Because most browsers will by default distinguish headers from paragraph and list copy (typically by increasing their size and weight), using an h2 automatically implements valuable default styling that you will not have to apply. To further customise the header, apply styling to the h2 and additional classes to headers that differ from your base styling.

Writing clean, well-structure markup has several additional and very important benefits.

- Structuring and specifying content in this way tends to result in very simple, lightweight files, which will be easy for the browser to load, parse and display. By comparison, large complex mark-up may impact memory, CPU and performance.
- Well-structured, semantic markup provides a level of detail that search engines can take advantage of. For example, many search engines will apply additional weight to the content within a <title> or <h1> tag when determining what the page is about.
- The World Wide Web Consortium (W3C) also recommends this approach: Because CSS gives considerable power to the "class" attribute, authors could conceivably design their own "document language" based on elements with almost no associated presentation (such as DIV and SPAN in HTML) and assigning style information through the "class" attribute. Authors should avoid this practice since the structural elements of a document language often have recognised and accepted meanings and author-defined classes may not.

Tip: Nokia Mobile Web Templates use clean, well-structured markup resulting in content that remains legible even when no CSS is applied.

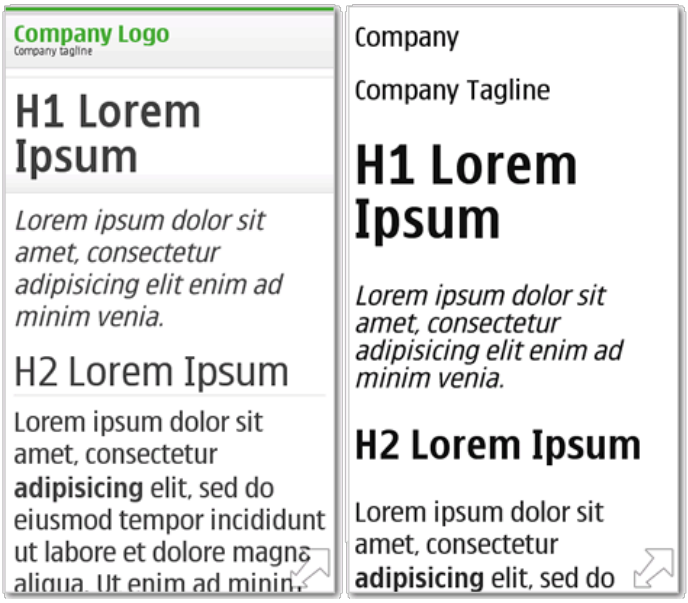


Figure: An example of the Nokia Mobile Web Templates with and without CSS styling.

Keep the layout fluid

Where possible, do not assign fixed values for layout or styling. Use percentages to define width, and ems to define margins, padding and font sizes. This can be tricky and there will be times where pixel values make more sense (for example, if positioning certain elements precisely) however if you keep your site as flexible as possible, this will limit the level of formal adaptation you must undertake.

Tip: Nokia Mobile Web Templates for High End Devices follow this approach to styling and markup. This enables most template elements to scale and adapt automatically to suit different screens sizes and orientations.

Tip: Use background colours or images (where supported) to extend the design
CSS background colours are supported on the vast majority of Nokia devices and can help visually break up and apply contrast to areas of your content without changing the HTML. The use and positioning of background images is also widely supported although backgrounds may tile and display less reliably on older devices.

Example: Using an h2, a bit of device detection (to determine browser capabilities), and two separate style sheets, you could extend the design of a header using backgrounds in the following way:

Style sheet 1

- For devices that support CSS3, specify a CSS-based background gradient.
- For devices with strong CSS and HTML support, but no CSS3, specify a tiled image-based background gradient. Ensure the design degrades gracefully by choosing a text colour that will work well on its own if the background image does not render. For example, don't use white text on a black background. If the background rendering fails, the white text will be invisible.

Style sheet 2

- For devices with poor CSS and HTML support, specify a background colour. Here again, ensure that the design degrades gracefully.

Because you have used an h2 instead of a div and class, devices that have poor or nonexistent CSS support will still understand the markup and render the header using that browser's default font size and style for h2s. This will differentiate the header from other elements on the page such as p and div.

While the above examples will require device detection to determine which style sheet to serve, this will be the only adaptation required to accomplish quite a few design changes—all of which will degrade gracefully.

Plan to progressively enhance The previous examples already provide a hint of this final guideline. The goal is to begin with a baseline of content that can be read by most devices, then implement enhancements on more capable devices based on detection of their capabilities. This technique is called progressive enhancement. For more discussion of this topic see the following articles:

- Opera: Graceful degradation vs progressive enhancement
- Wikipedia: Progressive enhancement
- Smashing magazine: Progressive enhancement, what it is, why do it?

Common approaches to device detection

Device detection can be accomplished server-side, before the content is delivered to the client, or client-side—once once the content and related files have been loaded to the device itself. Server-side detection is typically the most reliable however client-side detection can serve as a useful fall-back.

Server side adaptation

Advantages
Offers far greater flexibility as it enables developers to serve (and users to download) only the media and content that is best and most relevant for that particular device.
Can be extremely efficient when combined with device- or group-based caching on the server.

Typically allows easier design-time minification (compression) and optimisation of assets (e.g. CSS files, image sprites, scripts etc)

Disadvantages
While fairly reliable, server-side detection is not foolproof as it relies a great deal on the accuracy of a variety of data.

Client-side adaptation

Advantages
Is simple to execute using a combination of JavaScript and CSS.
Client-side detection can provide a useful fallback and can be used to implement last-minute tweaks related to real-time behaviour such as a changes in orientation.

On more capable devices, client-side Ajax and CSS can be used to download additional content and enhancements without having to refresh the entire page.

Disadvantages
Because JavaScript and CSS execute locally, (and depending on implementation) the entire page may have to download before the script can process and manipulate the DOM.
There may be a visible lag between the initial loading and the execution of the client-side adaptation.

Requires JavaScript, media queries or media types, all of which have variable levels of support.

Tip: For an example of client-side detection, see the Nokia Mobile Web Templates for High End devices, which use JavaScript to detect the device platform version and alternate style sheets to apply small design tweaks.

The following server-side and client-side detection techniques will be discussed in the next section. These approaches are often most effective when combined to ensure a higher level of detection accuracy and a greater level of flexibility in regards to content adaptation.

Server side adaptation approaches

- User Agent (UA) header string lookup
- UA string combined with device database lookup
- User Agent Profiles (UAProf) lookup

Client-side adaptation approaches

- Feature detection based on JavaScript technology
- CSS media types
- CSS media queries

Server-side User Agent (UA) and header lookup

User Agent lookup is a lightweight device-detection technique that involves examination of the device's UA string which is contained in the HTTP request headers. This approach has been used for some time in desktop web development to detect older browsers that may require different style sheets than their more contemporary counterparts.

Example of a UA string UA String

```
Nokia6600/1.0 (4.03.24) SymbianOS/6.1 Series60/2.0 Profile/MIDP-2.0 Configuration/CLDC-1.0
```

Device	Nokia6600/1.0 (4.03.24)
Operating system	SymbianOS/6.1 Series60/2.0
Profile*	Profile/MIDP-2.0
Configuration*	Configuration/CLDC-1.0

* Properties describing the devices Java MIDP/CLDC capabilities.

How it works

With each HTTP request, the UA string is sent by the device browser, and is therefore available to the server. Each user agent string contains information, that can enable identification of the browser and in many cases, the individual device model.

Once the string is obtained, it can be evaluated via regular expressions (regex) or, simple string manipulation can be used (either in languages such as Java™ , PHP, or even within .htaccess or similar files) to identify key predefined portions of the UA string. These key strings (such as 'symbian', 'smartphone', 'midp', 'wap', 'phone') help identify (and subsequently target) the device, browser, or device grouping these have been assigned to.

User Agent (UA) and header lookup

Advantages
Quick and relatively simple to implement.
Detection of the neighbouring headers can reveal important information such as screen resolution, language and encoding support (e.g. gzip). This type of detection typically matches only one or two parameters in the UA string. False positives and false negatives are possible.

Disadvantages
Certain browsers misreport the UA string or enable their users to change the reported UA string. This can cause false identification of the browser.
Firmware updates, regional variations and operator device differentiation can result in devices having more than one known UA string.

Some transcoders will alter the UA string, falsely identifying the device.

All you can reliably tell from UA strings is the browser name and identity. No data is provided regarding its capabilities.

Detecting other headers

UA is not the only HTTP header. Header data such as cookie and language can also be useful to determine basic browser capabilities. In fact, devices can typically be identified as 'mobile' from even the presence of a profile header, as shown in the sample code below.

Sample: Lightweight detection script

The code below—implemented as part of the WordPress Mobile Pack and available separately here—provides an example of a lightweight algorithm that detects the presence of a mobile device and works for a large percentage of mobile browsers. The code is primarily based on a list of approximately 90 well-known mobile browser UA string snippets, with a couple of special cases for Opera Mini, the W3C default delivery context and certain other Windows browsers. The code also looks to see if the browser advertises WAP capabilities as a hint.

The initial function checks for the presence of the x_wap and/or http_profile headers. Finding either of these identifies the device as mobile.

```
function lite_detection() {
  if (isset($_SERVER['HTTP_X_WAP_PROFILE']) ||
      isset($_SERVER['HTTP_PROFILE'])) {
    return true;
  }

// The following code retrieves the UAheader and attempts to match one of the common UA string prefixes (primarily identifying opera

  $user_agent = strtolower($_SERVER['HTTP_USER_AGENT']);
  if (in_array(substr($user_agent, 0, 4), lite_detection_ua_prefixes())) {
    return true;
  }

function lite_detection_ua_prefixes() {
  return array(  'w3c ','w3c-','acs-','alav', 'alca', 'amoi', 'audi', 'avan', 'benq', 'bird', 'blac', 'blaz', 'brew', 'cell', 'cldc
'dang', 'doco', 'eric', 'hipt', 'htc_', 'inno','ipaq','ipod','jigs','kddi','keji','leno','lg-c','lg-d','lg-g','lge-','lg/u','maui','
'mits','mmef','mobi','mot-','moto', 'mwbp', 'nec-','newt', 'noki', 'palm', 'pana', 'pant','phil','play','port','prox','qwap','sage',
'sams','sany','sch-','sec-','send','seri','sgh-','shar','sie-','siem','smal','smar','sony', 'sph-','symb','t-mo', 'teli','tim-','tos
'tsm-','upg1','upsi','vk-v','voda','wap-','wapa','wapi','wapp','wapr','webc','winw','winw','xda ','xda-',);
}

// The following code attempts to match the term wap within the http_accept header, which identifies acceptable content types.

  $accept = strtolower($_SERVER['HTTP_ACCEPT']);
  if (strpos($accept, 'wap') !== false) {
    return true;
  }

// The following code, retrieves the user agent header and attempts to match common strings identifying mobile platforms and operati

  if (preg_match("/(" . lite_detection_ua_contains() . ")/i", $user_agent)) {
    return true;
  }

function lite_detection_ua_contains() {
  return implode("|", array('android','blackberry','hiptop','ipod', 'lge vx','midp','maemo','mmp''netfront', 'nintendo DS','novarra'
'openweb','opera mobi','opera mini','palm','psp','phone','smartphone',  'symbian','up.browser','up.link','wap','windows ce', ));
}

// Finally, this cod retrieves all headers and attempts to match a common string identifying Opera Mini.

  if (isset($_SERVER['ALL_HTTP']) && strpos(strtolower($_SERVER['ALL_HTTP']), 'operamini') !== false) {
    return true;
  }
  return false;
```

```
}
```

Related articles:

- Mobiforge: Lightweight device detection
- Tutorial: Detecting device capabilities
- Mobile device detection

Server-side UA string combined with device database lookup

Device databases are large data repositories containing named properties or characteristics from thousands of known devices. The two most common databases are the open source WURFL and the commercial DeviceAtlas service. Both incorporate community-generated data, gathered via UA Profiles and online test suites. DeviceAtlas also obtains data directly from handset manufacturers.

Device database lookup typically matches the provided client UA string with a large dataset of known UA strings from a device capabilities database. Known device capabilities such as screen width, height, available codecs, global positioning system (GPS) support and many others are then returned (if available).

Mature device databases such as WURFL and DeviceAtlas ship with excellent tools and APIs (typically provided for common/popular languages and frameworks such as PHP, ASP.NET, Java language and Python).

Device database lookup

Advantages
Enables identification of thousands of devices from all over the world.
While they can be a little more involved to setup than simple UA string lookup, mature device databases ship with excellent tools and APIs that enable developers to be up and running very quickly.

Disadvantages
Requires initial resources to cover setup time and (in the case of Device Atlas and WURFL) the annual user license.

Requires hosting of the device dataset on your server. To remain accurate, this dataset must be updated regularly.

May impose an additional load on your server.

Due to the vast number of possible devices and parameters, device databases will inevitably contain a certain level of inaccuracy.

Database lookups match to a specific device rather than a group of devices. It is then up to the developer to correlate this information to the appropriate device grouping.

Server-side User Agent Profiles (UAProf) detection

UAProfs are XML (RDF-based) documents that contain information about the capabilities of a mobile device. A UAProf file describes common information including vendor, model, screen size and multimedia capabilities.

How it works

With each server request, the mobile device sends an HTTP request header containing the URL of its UAProf. This information is most often found within the x-wap-profile or Profile property.

Developers can then parse the UAProf RDF/xml data to retrieve the necessary device parameters and characteristics.

Example: UAProf for a Nokia 6230

Tip: UAProf URLs for most Nokia devices can be found in the Nokia Developer Devices section (within each individual device profile.)

UAProf detection

Advantages
Can assist in identifying the name and basic capabilities (e.g. screen size, operating system) of totally unknown devices that may not yet figure in device databases.

Disadvantages
UAProfs were more common in the past than they are today. Not all devices currently support UAProf (including the iPhone).
Many UAProf properties aren't particularly useful for web development or can be obtained more reliably in other ways.

UAProf lookup can and does fail. Not all advertised UAProf URLs are actually available.

There is no industry-wide data quality standard for the data within each field in a UAProf. Schema's can vary from device to device requiring the need to parse and query multiple times. Schema and data errors can also cause parsing to fail.

Retrieving and parsing UAProfs in real time is slow and can add substantial overhead to any given web request, necessitating the creation of a Device Description Repository to cache the UAProfs and a workflow to refresh UAProfs to check for deprecation.

UAProf headers can often be wrong, indicating a completely different device.

Detection based on JavaScript technology

Client-side JavaScript technology can be extremely useful for detecting relevant and available DOM objects and/or events such as screen.width, screen.pixelDepth, navigator.userAgent, navigator.cookieEnabled and applicationCache support.

Feature detection, (first introduced earlier when discussing progressive enhancement) is far safer than simply identifying the browser as it will conclusively identify support for the DOM features you plan to target. This method will of course fail on browsers that do not support JavaScript technology or where JavaScript has been disabled.

Tip: Nokia Mobile Web Templates use client-side device detection based on JavaScript technology to identify the browser UA string and implement simple tweaks to the template such as adjustments in font size and margins.

Advantages	**Disadvantages**
Future-proofs your site by ensuring only supported content and functionality are displayed.	Can be error-prone. JavaScript may be disabled or not supported at all on certain devices. Level of DOM feature support can also vary even if it technically shown to be 'supported'.

Code example:

```
// Detects and returns a variety of useful device properties such as device platform, screen width, screen height, colour depth and
Also returns the UA header string.

<script type="text/javascript">
   $(function() {
    var properties = [  "navigator.platform",
      "navigator.userAgent",
      "screen.width",
      "screen.height",
      "screen.colorDepth",
      "screen.pixelDepth",
      "window.innerHeight",
      "window.innerWidth",
        ];
    for (index = 0; index < properties.length; index++) {
     var property = $("<dt>").html(properties[index]);
     var value = $("<dd>").html(eval(properties[index]));
     $("#js-test").append(property).append(value);
    }

    });
</script>
```

The above example uses jQuery, which may not be supported on all devices. For a working example of detection based on standard JavaScript technology, please see http://www.developer.nokia.com/device-detection/ . The source files are also included in this article's sample files.

Note: As the online sample demonstrates, it is also possible to use JavaScript client-side, to detect the browser UA string using navigator.userAgent. Given the many advantages of server-side detection and adaptation, there is typically little benefit to an additional client-side UA string lookup.

CSS media types

CSS media types are parameters, that enable style or style sheet switching based on recognition of the device's declared media type. There are ten media type categories, each named to reflect the target device. The media type for mobile devices is 'handheld'. Others include screen, print, tv and braille.

Media types can be defined in several ways.

Within the document HTML

```
<link rel="stylesheet" href="handheld.css" media="handheld"/>
// loads a style sheet called handheld.css when the media type matches to 'handheld'.
```

Within the CSS and combined with an import rule

```
@import url("reset.css") handheld;
// Imports a style sheet called reset.css when the media type matches to 'handheld'.
```

Media types

Advantages	**Disadvantages**
Simple to implement.	Can be error-prone. Media types are not well supported throughout the industry. Media types only identify the device as mobile. They do not provide information regarding the identity of the device, or its characteristics. Media types are only useful to implement CSS-based changes. They cannot be used to adapt the HTML markup or content structure. Important: Nokia S60 devices do not read or recognise the handheld media type and identify all devices as 'screen'.

Related articles

- A list apart: Return of the mobile style sheet

CSS media queries

CSS3 provides enhanced support for media-dependent style sheets through the use of CSS media queries. Media queries enable developers to customise the specified style or style sheet based on common parameters (or features) such as screen or device width. A media query consists of a media type (see iv) and at least one expression that—when resolving to 'true'—limits the named style sheets' scope through specification of one of the named parameters.

Linking to a custom style sheet using a media query

```
<link rel="stylesheet" href="handheld.css" media="only screen and (max-device width:480px)"/>
```

When placed in the head of the document, the above markup loads the handheld.css style sheet only if the media type is identified as 'screen' and the device width does not exceed 480 pixels. This method will load a completely new style sheet so is most appropriate when making significant changes to the CSS or in cases where there are many distinct style sheets being produced (for example one for 320-pixel-wide devices, one for 480-pixel-wide devices and one for 176 pixel-wide-devices).

Switching standalone styles using a media query

```
@media handheld and (orientation: portrait) {
 .navigation { width: 80%; }
 #footer { margin: 1em; }
}
```

When placed within a style sheet, the above markup changes the width of any elements specified with the navigation class to 80% and sets the margin to 1em for the div specified with an id of footer, but only if the media type is identified as 'handheld' and the orientation resolves to 'portrait'.

Media queries placed within a style sheet are ideal when small tweaks are required to very distinct elements, which do not warrant the loading (and subsequently managing) of a completely different style sheet. Note however that adding many media queries such as these will of course increase the size of the style sheet—potentially unnecessarily if this additional markup will only be relevant to a few devices.

The most immediately useful parameters for mobile development tend to be width, height, device-width, device-height, orientation, aspect-ratio, device-aspect-ratio, color and resolution. A full list can be found on the W3C website.

Media queries

Advantages
Simple to implement.
Media queries typically degrade well on unsupported devices as the markup is ignored. Can be error-prone. Media queries are not yet well supported and this lack of support is amplified due to the reliance on the specification of a possibly unsupported media type.

Disadvantages
Media queries are only useful to implement CSS-based changes. They cannot be used to adapt the HTML markup or content structure.
For a list of media type and query support, visit the W3C's mobile CSS Media Type test suite (warning: very large download).

For a working example of CSS media queries, please see http://www.developer.nokia.com/device-detection/ . The source files are also included in this article's sample files.

Related resources:

- A list apart: Return of the mobile style sheet
- Quirksmode: The orientation media query
- Mozilla: Media queries

Additional best practices

If, as part of your mobile strategy you choose to automatically redirect users to a mobile site, there is one more thing to consider. With all the care in the world, device detection may still not conclusively or accurately identify the device. One of your chosen parameters may be inaccurate, the user may be using a new device with an unfamiliar user agent, or the operator may have modified the device.

There is also the issue of choice. You cannot anticipate what a user will be thinking when accessing the site. It may be a random visit, or a completely intentional one where the user knows exactly what s/he's looking for and where to find it. Don't completely remove a user's choice allow him or her to switch between desktop and mobile optimised sites based on his or her preference. There are several recommended approaches to provide this type of choice.

Redirection + manual link

If detecting the mobile device and automatically redirecting the user to the appropriate experience, provide a 'switch to desktop site' link in your mobile site's footer area. This link will enable the user to manually navigate to the desktop site. If the user chooses to click this link, store their choice using a cookie but provide an equivalent 'switch to mobile' link on your desktop site to enable them to return if needed. The 'switch to mobile' link is best positioned at the top left of the page. This will ensure it is visible as soon as the desktop page loads on a mobile device.

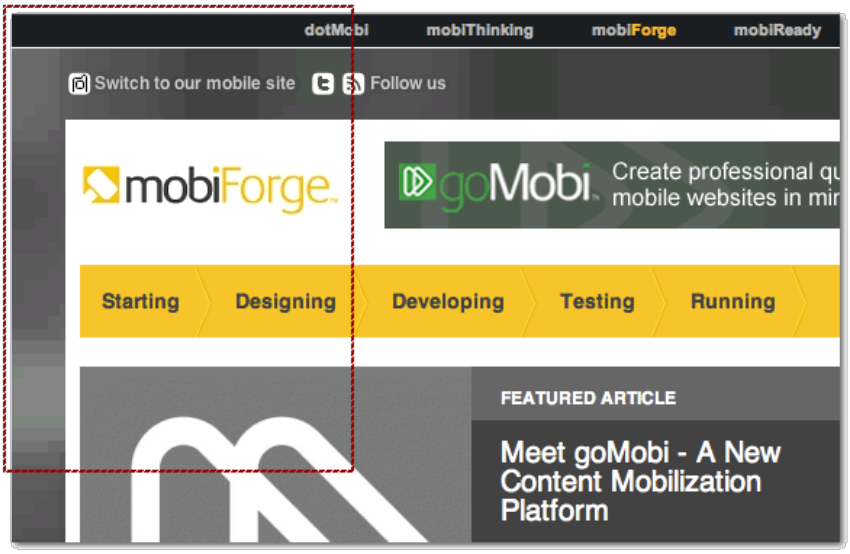


Figure: On the mobiForge web site, the 'Switch to our mobile site' link is positioned to be immediately viewable when loading a desktop site into a mobile viewport.

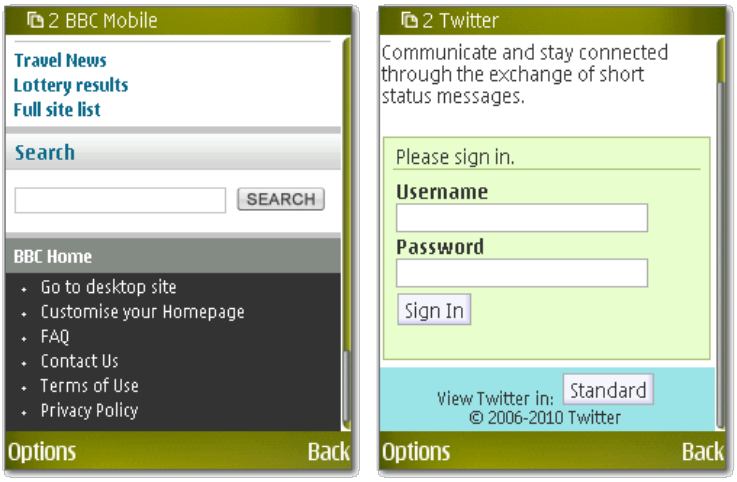


Figure: The BBC uses the phrase 'Go to desktop site' while Twitter provides users with a 'Standard' and 'Mobile' option.

Landing page + manual link

Create a landing page, that enables users to choose between the mobile and desktop site. Once the has been made, store the choice using a cookie. This approach should be combined with a manual link (as described above) to ensure that users can access the alternate site if needed.

Promoting your mobile presence Advertise the entry point to your mobile site using a domain extension such as .mobi or a variant of your own domain such as:

- m.mydomain
- mobile.mydomain
- mydomain.com/mobile

Be sure to consider all possible entry points and redirect those that are unused to the official advertised domain. For example, typing m.lufthansa.com redirects users automatically to mobile.lufthansa.com, which is the advertised domain for the Lufthansa mobile presence.

See the following articles on MobiForge for an in-depth example of mobile site discovery approaches and examples of a lightweight mobile site switching algorithm:

- mobiForge: a very modern mobile switching algorithm part 1
- mobiForge: a very modern mobile switching algorithm part 2

Downloadable sample page

The attached zip file contains a sample source files for device detection.

File:Device-detection-sample.zip