Browser Detection and Cross Browser Support

Bob Clary, Netscape Communications
Published 17 July 2002
Revised 10 February 2003

Improper browser detection can lead to web maintenance nightmares. Rethinking the basics of when and how to detect user agents is crucial to creating maintainable, cross browser web content. This article reviews several approaches to browser detection, their usefulness in specific circumstances to arrive at a common sense approach to browser detection.

When reporting browser statistics, you should aggregate traffic for all Gecko based browsers. Please see Browser Statistics for more information on how to accurately report Gecko based users.

Visit Gecko Central for more specific information on supporting Gecko based browsers. Visit CSS Central and DOM Central for more information regarding cross browser web development.

Introduction

In an ideal world, we could author HTML, XML, CSS and JavaScript and only worry about the W3C and ECMA standards. However, we don't quite live in such a world yet. Due to bugs, incomplete implementations of the standards and legacy browsers, web developers must be able to determine which browser a visitor is using and provide the appropriate content and scripting code path.

Although browser detection is perhaps the most common scripting task that every web developer faces, it seems that the variety of different strategies in use for detecting browsers is unlimited. As a member of the Netscape Evangelism team who has spent over a year investigating existing web content, I can say without a doubt that most compatibility problems found on the web today are due to a lack of understanding of the standards combined with inadequate and inappropriate browser detection strategies.

This article is intended to provide an overview of browser detection strategies and best practices. For more specific Netscape Gecko™ recommendations, please see the Netscape Gecko Compatibility Handbook.

Netscape Gecko

Although many web developers are aware of Netscape 6 and Netscape 7, far fewer are aware that Netscape 6 and 7 are members of an entire family of user agents based upon Netscape Gecko that includes the commercial browser CompuServe 7, and open source browsers such as Mozilla, Galeon, and Kmeleon.

Netscape Gecko was designed from the ground up to be compliant with the W3C HTML, W3C CSS, W3C XML, W3C DOM, and ECMAScript (JavaScript) standards. It also includes compatibility features which allow it to reasonably handle legacy content which was developed for earlier generations of browsers such as Netscape Navigator 4 as well as features which provide compatibility with Internet Explorer 5 and 6. Unlike other browsers, Netscape Gecko is truly a cross platform browser and provides identical support on all operating systems where it is supported.

The easiest way to support Netscape Gecko is to create content which only uses the standards.

Unfortunately, no other browser supports the standards as completely as Netscape Gecko which means that web developers and authors are forced to continue to provide support for other browsers which do not support the standards as fully. Fortunately, other browsers such as Internet Explorer 5 and 6 for Windows, to a lesser extent Internet Explorer 5 for Macintosh and Opera 6 also support the standards to a degree. These other browsers also appear to be moving towards more complete and rigorous support for the standards and there is hope that in the future web developers and authors will be able to dispense with browser detection at least with regard to features governed by standards.

We are still faced with the question of how to develop standards based content while supporting the differing implementations of modern browsers while at the same time supporting (to a lesser degree) older and less capable browsers. Browser detection is key to accomplishing this task.

Browser Detection History primer

To understand why many common browser detection strategies are inappropriate, we must first look back on how these strategies came into being.

In the earliest days of the web, HTML was very simple, not standardized and did not include any support for client side scripting. HTML itself was not standardized until HTML 2.0 was introduced in late 1995 and it did not even include tables. Browser vendors such as Netscape and Microsoft competed to add compelling features to the HTML they supported in their browsers in order to provide the richest most compelling content to their users and to entice web authors to support them. The abilities of browsers to support the latest and greatest content changed on an almost daily basis.

Web authors were faced from the beginning with a variety of browsers, some of which supported the latest and greatest version HTML and some which did not. The solution was either to provide the lowest-common denominator of HTML or to use browser detection techniques on the web server to send customized content to each browser depending on what level of support the browser provided. Server side browser detection using user agent strings was born.
User agent strings are defined in the HTTP protocol and are available to web servers (see RFC 1945 - Hypertext Transfer Protocol -- HTTP 1.0 and RFC 2068 - Hypertext Transfer Protocol -- HTTP 1.1).

The most common approach at this time was to distinguish user agents by vendor and version using the reported user agent string. Although this approach was considered reasonable at the time, this approach caused problems for browser vendors right from the beginning. The original Netscape browsers used a user agent string which began with the code name for the Netscape browser followed by it's version number, e.g. Mozilla/version followed by a comment token which gave additional information regarding the operating system being used, etc. Since the earliest browser detection techniques were based upon looking for a Netscape based browser and only provided customized content to browsers which used the Mozilla/version user agent string, other browser vendors standardized on using Mozilla/version to signal that they were compatible with a particular Netscape version. Since other browsers pretended to be Netscape browsers and encoded their version information in a non-standard fashion in the user agent comment area, the task of determining which browser was being used became more complicated than it should have been.

Netscape Navigator 2 introduced the ability to run JavaScript in web browsers. As browser evolution continued, differences in the implementation of scripting and the objects supported by browsers appeared. Web authors were no longer limited to detecting browsers on their web servers, but could now execute scripts client side (in the browser itself) which could be used to distinguish browsers.

One of the earliest approaches to client side browser detection involved testing whether the browser supported particular objects. An example of this approach involved testing for the existence of the document.images object.

While object based detection was used in some circumstances, many web authors continued to use the vendor/version approach to distinguishing web browsers in their client side scripts. Since the user agent string was exposed as a property of the navigator object (e.g. navigator.userAgent), many web authors used the same logic in their client side scripts as they had used earlier in their server side browser detection. In addition to navigator.userAgent other properties such as appName and appVersion were available in the navigator object which could be used in browser vendor/version detection strategies.

The classic example of this vendor/version client side detection strategy can be found in the Ultimate Browser Sniffer. This script and variants of it can be found today on many web sites where it is a common source of detection problems.

Netscape Navigator 4 and Internet Explorer 4 introduced the ability to manipulate HTML content in a browser (Dynamic HTML or DHTML) rather than on the web server and began the introduction of support for CSS to style content. This generation of browser, in addition to sharing several features which were not available in earlier versions, each implemented their own (incompatible) competing abilities to manipulate content in a web page.

Since each vendor's browser implemented different objects to perform DHTML, web authors began to use object detection to distinguish vendor/version through the existence of particular JavaScript objects. The existence of document.layers was sufficient to be sure that the browser was Netscape Navigator 4 while the existence of document.all was sufficient to be sure that the browser was Microsoft Internet Explorer 4. An implicit assumption by many web authors around this time was the there were only two types of browser available... Netscape Navigator and Microsoft Internet Explorer.

These strategies of classifying browsers by vendor/version, assuming that the only browsers being used where either Netscape Navigator 4 or Internet Explorer 4 failed when alternative browsers such as those based upon Netscape Gecko were introduced. Many of the problems reported in the press regarding Gecko's inability to display content were directly related to inadequate, inappropriate browser detection strategies.

A final note on vendor/version strategies. A web developer who fully utilizes the browser detection and distinctions in the Ultimate Browser Sniffer will produce code which uses code forks for many browsers and versions. Imagine attempting to maintain a web site which uses many of the browser variables available from the Ultimate Browser Sniffer.

browser vendor	is_nav, is_ie, is_opera, is_hotjava, is_webtv, is_TVNavigator, is_AOLTV
browser version number	is_major (integer indicating major version number: 2, 3, 4 ...) is_minor (float indicating full version number: 2.02, 3.01, 4.04 ...)
browser vendor AND major version number	is_nav2, is_nav3, is_nav4, is_nav4up, is_nav6, is_nav6up, is_gecko, is_ie3, is_ie4, is_ie4up, is_ie5, is_ie5up, is_ie5_5, is_ie5_5up, is_ie6, is_ie6up, is_hotjava3, is_hotjava3up, is_opera2,

browser vendor	is_nav, is_ie, is_opera, is_hotjava, is_webtv, is_TVNavigator, is_AOLTV
	is_opera3, is_opera4, is_opera5, is_opera5up
JavaScript version number	is_js (float indicating full JavaScript version number: 1, 1.1, 1.2 ...)
OS platform and version	is_win, is_win16, is_win32, is_win31, is_win95, is_winnt, is_win98, is_winme, is_win2k, is_os2, is_mac, is_mac68k, is_macppc, is_unix, is_sun, is_sun4, is_sun5, is_suni86, is_irix, is_irix5, is_irix6, is_hpux, is_hpux9, is_hpux10, is_aix, is_aix1, is_aix2, is_aix3, is_aix4, is_linux, is_sco, is_unixware, is_mpras, is_reliant, is_dec, is_sinix, is_freebsd, is_bsd, is_vms

Detecting browsers using this level of detail is unworkable, unmaintainable and violates the basic principles of web authoring! I strongly advise everyone to avoid this trap.

Problems caused by inappropriate Browser Detection

Excluding Unknown Browsers

If you only provide tests for specific browsers in your detection logic, your site will not be usable if a visitor uses a different browser. Consider the following example:

```
// WRONG APPROACH - do not use!
if (document.all)
{
  // Internet Explorer 4+
  document.write('<link rel="stylesheet" type="text/css"
src="style-ie.css">');
}
else if (document.layers)
{
  // Navigator 4
  document.write('<link rel="stylesheet" type="text/css"
src="style-nn.css">');
}
```

Note how the above example only provided stylesheets for Internet Explorer and Navigator 4 and even then only if the visitor has JavaScript support turned on in their browser. Users of Netscape 6, Netscape 7, CompuServe 7, Mozilla, Opera will not be able to view the site properly.

Misidentifying Browsers

A common mistake web authors make is to assume that if a browser is not Netscape Navigator 4, it must be Internet Explorer and vice versa. For example:

```
// WRONG APPROACH - do not use!
if (document.all)
{
  // Internet Explorer 4+
  elm = document.all['menu'];
}
else
{
  // Assume Navigator 4
  elm = document.layers['menu'];
}
```

Note how the above example assumed that any browser that was not Internet Explorer was Navigator 4 and attempted to use Layers. This is a common source of problems when using browsers based upon Netscape Gecko as well as Opera. A similar error can be seen in the following example:

```
// WRONG APPROACH - do not use!
if (document.layers)
{
  // Navigator 4
  elm = document.layers['menu'];
}
else
{
  // Assume Internet Explorer 4+
  elm = document.all['menu'];
}
```

Netscape 6 was the first commercial browser released based upon Netscape Gecko. Due to a lack of communication and understanding, many sites have created inappropriate detection strategies based upon the Netscape 6 user agent string. Netscape 6's user agent string follows the HTTP standards for specifying the version of the user agent. (see Mozilla user-agent strings and Netscape Gecko User Agent Strings)

```
Mozilla/5.0 (...) Gecko/20001108 Netscape6/6.0
```

The first vendor/version (Mozilla/5.0) indicates that Netscape 6 is a fifth generation browser and is not identical to earlier browsers. All Gecko based browsers currently report Mozilla/5.0 as their primary version although no other browser does so at the moment. Hopefully when other browser vendors achieve the same level of standards support as Gecko and begin to drop support for legacy browsers, they too will begin to report version 5. Assuming that only Netscape Gecko will use Mozilla/5.0 will cause your browser detection logic to fail as soon as another browser vendor releases a browser which reports Mozilla/5.0.

The second vendor/version (Gecko/20001108) identifies Netscape 6 as a particular release of Netscape Gecko which was built on November 8, 2000. If you must detect Gecko using the user agent string, Gecko/CCYYMMDD is the most appropriate string to search for.

The third vendor/version (Netscape6/6.0) identifies this particular instance of a Gecko browser as Netscape 6.0. Many sites keyed off of the existence of the string Netscape6 in the user agent and used tests similar to:

```
if (navigator.userAgent.indexOf('Netscape6') != -1)
```

```
{
  // Netscape 6 code
}
```

Note how this type of detection misses any other Gecko based browser. Unfortunately, the Netscape 6 user agent string was not sufficiently generalizable due to it's use of the string Netscape6 as a description of the vendor. Netscape 7 corrects this error and introduces another chance for user agent string based detection to fail.

```
Mozilla/5.0 (...) Gecko/200207XX Netscape/7.0
```

Note how Netscape 7 no longer uses the string Netscape6 as the Vendor. Any user agent string detection strategy for Gecko based upon the existence of the string Netscape6 will fail to detect Netscape 7.

Using JavaScript Objects to determine vendor/version

As we already discussed, a common approach in the past was to use objects to classify browsers by vendor/version. A common type of detection which originally was written to support only Netscape Navigator 4 and Internet Explorer 4 might look like:

```
// WRONG APPROACH - do not use!
if (document.all)
{
  // IE4
  height = document.body.offsetHeight;
}
else if (document.layers)
{
  // NN4
  height = window.innerHeight;
}
else
{
  // other
  height = 0;
}
```

With the introduction of the W3C DOM, the standard method document.getElementById became available in Internet Explorer 5 and later in Netscape 6 (Gecko). Many web authors decided that the easiest way to add support for Netscape Gecko was to add another code fork as follows:

```
// WRONG APPROACH - do not use!
if (document.all)
{
  // IE4
  height = document.body.offsetHeight;
}
else if (document.layers)
{
  // NN4
  height = window.innerHeight;
}
else if (document.getElementById)
{
  // They think this is Netscape Gecko
  // but could be wrong!
  height = window.innerHeight;
}
else
{
  // other
  height = 0;
}
```

The approach is incorrect since it assumes that the only other browser besides Internet Explorer 5+ that implements document.getElementById is Netscape Gecko. This is not true today and will become even less true as more browsers which support the W3C DOM Standards are introduced in the future.

Recommendations

Target the standards and not particular browsers

While the period from 1994-2000 was dominated by incompatible non-standard browsers from Netscape and Microsoft, today the dominating factor in web development are the standards proposed by the World Wide Web Consortium (W3C). Standards are important for web developers due to the increased flexibility, power of presentation, support for users with disabilities to name just a few reasons.

Targeting your web content to particular vendors ignores the possibility that other browsers which support the same standards may be introduced in the future. A common problem on the web today is the assumption that the only browsers in the world are Netscape Navigator and Microsoft Internet Explorer. This ignores the existence of Opera as well as the newer handheld devices which are being used to access the web today and in the future.

Today, both Netscape (Netscape 6 and above) and Microsoft (Internet Explorer 5.5 and above) provide browsers which due to their support for the standards are more similar than they are different. This similarity will only increase as each browser implements more and more of the standards.

Netscape 6 and above, Internet Explorer 6 on Windows, and Internet Explorer 5 for Macintosh all support DOCTYPE switching. This is a technique where these browsers can be switched from Quirks mode (which emulates buggy implementations in earlier generation browsers) to Standards mode (which more strictly adhers to the Standards). For new content, we recommend that you use a DOCTYPE which will invoke Standards mode in Netscape Gecko and Internet Explorer 6. This will ensure that your designs work similarly in these browsers as well as in any other browsers which support the Standards.

Provide support for unknown browsers

Always provide content and code paths for unknown browsers. The recommended approach is to assume that any unknown browser supports the basic standards of HTML and CSS and to a certain extent JavaScript and the W3C DOM. This will guarantee that your content will be supported today and in the future by any browser which supports the standards.

Limit the use of vendor/version specific features

Authoring content which is standards compliant is the easiest way to support the widest range of user agents and to decrease your maintenance costs. While it is not always possible to avoid vendor/version specific differences between browsers, the use of such features and the distinction between browsers based on vendor/version should be strictly limited to those areas where it is still required.

Limit the use of User Agent String based Detection

Server side browser detection requires the use of user agent strings. We recommend that the use of user agent string based detection be limited to server side situations and except in those circumstances where it is absolutely required such as when details of the Netscape Gecko branch are required.

There are legitimate reasons to use the user agent string (or the navigator object) to determine exactly what vendor, version or operating system is being used. Many financial sites (banks, online stock trading firms, etc) have very strict policies with respect to what browsers they support. This is due to the history of security exploits that have been discovered in older browsers. If you have the need to only allow specific versions of browsers to use your secure site, then the user agent string and the related information from the navigator object can be very useful.

You can also use detailed information regarding a browser's version to work around bugs in specific releases of a browser. However, this can quickly become a maintenance nightmare if you are not careful. I recommend that you only provide work arounds for bugs on a temporary basis and as soon as bugs are corrected in newer releases of the browser that you require your visitors to upgrade their browser.

Provide downlevel pages for older browsers

No commercial web site today makes complete support for Netscape Navigator versions 1, 2 or 3 or Microsoft Internet Explorer 3 a requirement. The reasons are that the capabilities of such browsers are far too limited compared to more modern browsers, the added development and quality assurance requirements add too much to the development cost of web sites and the market share of such browsers does not justify the expense of supporting them.

One of the most important decisions you can make in order to improve the quality of your site and decrease development, maintenance and quality assurance costs is to provide only limited support for older browsers such as Netscape Navigator 4 and Internet Explorer 4. One of the more common approaches in major web sites is to provide a downlevel version of a web page to older browsers while providing a richer page which uses advanced CSS and JavaScript to more modern browsers. This can be accomplished through server side browser detection either as part of a scripted solution or as part of the web server's native ability to serve different content to different user agents. This approach does not necessarily require you to author separate pages for downlevel and modern browsers. A common solution is to author content in a neutral format such as XML and use XSLT transformations to generate the necessary HTML for each class of browser.

Whether a particular browser should be provided with downlevel content depends to an extent on what features of CSS or JavaScript are being used in the advanced content. Netscape Navigator 4 and Internet Explorer 4 should both be considered downlevel browsers for most pages due to their limited support for CSS and the more recent DOM based standards. If your pages make use of advanced JavaScript which manipulates or creates new content using the W3C DOM Core, then Opera 5 and 6 should be considered downlevel as well due to their limited support for the W3C DOM.

The future belongs to developers and browsers which support standards. If you fail to take advantage of the coming change in browsers, your competitors will eat your lunch. Once that happens, the only place your web site will be found is on the web archive.

Use non-script based detection methods where possible

Older browsers have many limitations which result in their ignoring more advanced features. By judiciously using these limitations in older browsers you can include modern content while at the same time supporting older browsers.

HTML provides several methods of detecting support for various features such as support for scripting and FRAMES. Use these natural abilities of HTML to extend the range of browsers your content supports.

Using NOFRAMES to support NON-FRAMES capable browsers

```
<HTML>
  <HEAD>
    <TITLE>FRAMES</TITLE>
  </HEAD>
  <FRAMESET ROWS="30,*">
    <FRAME SRC="foo.html">
    <FRAME SRC="bar.html">
    <NOFRAMES>
      <BODY>
        <P>
          This page requires frames. See <a
href="noframes.html">sitemap</a>.
        </P>
      </BODY>
    </NOFRAMES>
  </FRAMESET>
</HTML>
```

Using NOSCRIPT to support non-scriptable browsers

Some browsers may not support scripting while some users may have scripting support turned off in their browser. Use the NOSCRIPT tag to provide these users with alternative unscripted versions of your pages or to at least notify them of the need to use scripting in order to view your content properly.

Since browsers such as Netscape Navigator 4 and Internet Explorer 4 do not support some of the most recent additions to the JavaScript (ECMAScript) standard, it is often necessary to limit the use of advanced JavaScript features such as exception processing. One approach is to require that users of browsers which do not support the level of JavaScript used in your content to turn off JavaScript in order to be able to use your content. You can do this by providing an error message for users of older browsers as well as alternative content contained in NOSCRIPT tags.

```
<HTML>
  <HEAD>
    <TITLE>NOSCRIPT</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript">
      window.onerror = function ()
      {
        // redirect user to a page describing the limitations
        // of their browser and requesting that they turn off
        // JavaScript in order to view your site.
      }

      // Netscape Navigator 4 will throw an error
      // on any JavaScript which attempts to use
      // try ... catch exception processing.
      try
      {
        // Code to implement fancy Menu
      }
      catch (errors)
      {
        // handle Exceptions
      }
    </SCRIPT>
    <NOSCRIPT>
      <!--
      if javascript is not enabled, then the browser
      will display the contents of the NOSCRIPT tag
      which in this case is a simple MENU implemented
      as an unordered list.
      -->
      <UL>
        <LI><A HREF="choice1.html">Choice1</A></LI>
        <LI><A HREF="choice2.html">Choice2</A></LI>
      </UL>
    </NOSCRIPT>
  </BODY>
</HTML>
```

Using SCRIPT LANGUAGE to choose the browser where it will be executed

The choice of scripting language is determined by the LANGUAGE attribute of the script tag.

Internet Explorer 4 and above can support a variety of script languages. The most common are VBSCRIPT and JavaScript. Internet Explorer also uses JSCRIPT as a synonym for JavaScript. Since other browsers do not support the language attribute VBSCRIPT or JSCRIPT you can use these languages when you wish certain scripts to only be executed by Internet Explorer 4 and above.

```
<HTML>
  <HEAD>
    <TITLE>SCRIPT Languages</TITLE>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript">
      // JavaScript Code to implement fancy Menu
      // Visible to all JavaScript capable browsers
    </SCRIPT>
    <SCRIPT LANGUAGE="JScript">
      // JavaScript Code that uses proprietary
      // Internet Explorer features not available in
      // other browsers.
    </SCRIPT>
    <SCRIPT LANGUAGE="VBScript">
      // VBScript Code that uses proprietary
      // Internet Explorer features not available in
      // other browsers.
    </SCRIPT>
  </BODY>
</HTML>
```

Using Netscape Navigator 4's CSS limitations

It is possible to use Netscape Navigator 4's limitations for CSS support to automatically exclude certain CSS rules from ever being seen by Navigator 4.

For example, Navigator 4 does not understand the @import directive in CSS and will not load any external CSS Style sheets specified via @import. This technique can be used to provide basic common CSS rules for all browsers (including Navigator 4) while providing more advanced rules in a external CSS file for more modern CSS compliant browsers.

```
<STYLE type="text/css">
/* Navigator 4 CSS rules */
</STYLE>

<STYLE type="text/css">
/* Advanced CSS rules ignored by Navigator 4 */
@import "advanced.css";
</STYLE>
```

A similar technique is available for hiding CSS rules from Navigator 4 using the fact that Navigator 4 will ignore CSS rules after an occurence of /*/*/ in a stylesheet.

```
<STYLE type="text/css">
/* Navigator 4 CSS rules */
```

```
/*/*/
/* Advanced CSS rules ignored by Navigator 4 */
</STYLE>
```

DevEdge uses this technique to hide advanced CSS from Navigator 4.

Use feature oriented object detection

Object detection is a powerful method of providing cross browser support in your web content. Although you can use object detection as just another means of distinguishing between vendor/version the technique shows it's true power when used to detect features rather than browsers.

Feature oriented object detection is the testing for the existence of specific objects before attempting to use them in a script. The classic example is:

```
if (document.images)
{
  // code which processes the images
}
```

The advantage to feature detection is that it will work regardless of the vendor/version. We can rewrite the earlier example which illustrated problems with using objects to determine vendor/version to use feature detection instead.

```
if (document.body && typeof(document.body.offsetHeight) == 'number')
{
  height = document.body.offsetHeight;
}
else if (typeof(window.innerHeight) == 'number')
{
  height = window.innerHeight;
}
else
{
  height = 0;
}
```

Note how the previous example does not make assumptions about which browser is being used. Instead, it only tests for the objects which it wishes to use. Since the numeric values could potentially be zero, the script tests the type of the objects to make sure they are numbers instead.

Gecko and Navigator 4

Gecko is the replacement for Navigator 4 and as such retains many features from Navigator 4. The basic differences between Navigator 4 and Gecko can be easily summarized by:

Differences between Gecko and Navigator 4

Gecko is standards conformant

Gecko supports many more standards than Navigator 4 and implements them correctly unlike Navigator 4.

Gecko does not support Layers

Navigator 4 introduced the Layer API which it used to manipulate content and which formed the basis of DHTML in Navigator 4.

Layers however were not accepted by the W3C either in HTML or in the DOM. Since Gecko's mission was to be the most standards conformant browser possible, Layers were not supported. This lack of backwards compatibility has been the cause of many problems for web authors, but can easily be overcome through proper authoring and browser detection strategies. As the use of Navigator 4 decreases and more authors use the standards in their content, Layers and the problems they cause will disappear.

Gecko and Internet Explorer

Gecko implements a number of Internet Explorer only proprietary features especially with respect to their DHTML object model. Gecko's support for a number of IE's features has steadily increased since the introduction of Netscape 6 in November 2000. The best approach to take advantage of these IE compatibility features in Gecko is to use object based feature detection. This will automatically use any such features in Gecko if they are available in the version of Gecko being used. See the Client Object Cross Reference for more details on which Internet Explorer's objects and properties are supported by which version of Gecko.

A number of Internet Explorer features are not supported by Gecko. These include the window.event object, behaviors, filters, transitions, and ActiveX.

How (and when) to use the navigator object when detecting Gecko

Unless you specifically need to detect if Gecko is being used, do not use these methods. These methods should only be used in circumstances which can not be handled by using object feature detection such as when specific versions of Gecko must be excluded for security reasons.

Note: For client side detection, we recommend using the navigator object and it's properties. All of the information reported in the navigator is also available in the user agent string which can be used in server side situations.

Product

navigator.product is specific to Gecko browsers and will always return 'Gecko'. The is a quick and simple means of determining that a browser is based upon Netscape Gecko.

CVS Branch Tag

Beginning in Gecko 0.9.0 (Netscape 6.1 in Gecko 0.9.2), navigator.userAgent contains the CVS branch tag of the source which was used to create the version of Gecko being used in the browser. The branch tag is contained in the comment area of the user agent string and follows the string 'rv:'. In the following example, the branch tag is a.b.c.

```
Mozilla/5.0 (...; rv:a.b.c) Gecko/CCYYMMDD Vendor/version
```

Gecko browsers which are built from the same branch share the same basic version of Netscape Gecko and can be treated similarly when dealing with HTML, CSS, JavaScript, etc. For example, Netscape 6.2, 6.2.1, 6.2.2, 6.2.3 and CompuServe 7 are all built from the 0.9.4 branch and therefore share similar behavior in many ways.

Netscape 6.0	contained M18 rather than the rv value
Netscape 6.1	0.9.2
Netscape 6.2	0.9.4
Netscape 6.2.1	0.9.4
Netscape 6.2.2	0.9.4.1
Netscape 6.2.3	0.9.4.1
CompuServe 7	0.9.4.2
Netscape 7.0	1.0.1
Netscape 7.01	1.0.2

As you can see, all versions of Netscape 6.2 and CompuServe 7 were created from the 0.9.4 branch. The distinction between 0.9.4, 0.9.4.1, 0.9.4.2 is minor.

Note: The branch tag is a string and can contain more than single digits in any particular level. For example, it is conceivable that someday there will exist branch tags similar to 2.2.0 and 2.12.36. Since these values are strings, it is not possible to use relative string comparisons to determine which branch tag came later. In our example, branch 2.2.0 was created before 2.12.36 however comparing these values as strings shows '2.2.0' > '2.12.36'. The JavaScript function geckoGetRv() provides one solution to this problem by converting the branch tag in the user agent string into a floating point number where each level of the branch tag is considered as a number from 0-99.

0.9.2	0.0902
0.9.4	0.0904
0.9.4.1	0.090401
0.9.4.2	0.090402
1.0.1	1.0001
1.0.2	1.0002
2.2.0	2.02
2.12.36	2.1236

geckoGetRv() returns values which can be compared using greater than, less than, etc. geckoGetRv() is not an official part of Netscape Gecko however is provided as an example of approaches you can take to compare the different branch tags today and in the future.

Build Date

navigator.productSub is specific to Gecko browsers and will return a string containing the date the browser was built in the format CCYYMMDD (e.g. '20020801' for August 1, 2002). If you are concerned about a specific security issue in Gecko and know for example that all Gecko browsers contain a fix for the issue after a certain date, you can check that the navigator.productSub value is after that date.

You can also distinquish between point releases using a combination of the branch tag and the build date. For example, Netscape 6.2.2 and Netscape 6.2.3 both have branch tags 0.9.4.1, but Netscape 6.2.2 has navigator.productSub == '20020314' while Netscape 6.2.3 has navigator.productSub == '20020508'.

vendor/version

All Gecko browsers report the vendor and vendor's version both in the navigator object and the user agent string. The vendor and version information is not as useful as the branch tag and the build date however and we do not recommend their use. However, if you wish, you can distinguish the different types of Gecko browser using these values. As we saw earlier, the vendor/version appear in the user agent string following the Gecko version.

```
Mozilla/5.0 (...; rv:a.b.c) Gecko/CCYYMMDD Vendor/version
```

The vendor is available in the navigator object as navigator.vendor while the vendor's version is available as navigator.vendorSub.

Netscape 6.0	Netscape6
Netscape 6.01	Netscape6
Netscape 6.1	Netscape6
Netscape 6.2	Netscape6
Netscape 6.2.1	Netscape6
Netscape 6.2.2	Netscape6
Netscape 6.2.3	Netscape6
CompuServe 7.0	CS 2000 7.0
Netscape 7 Preview Release 1	Netscape
Netscape 7.0	Netscape
Netscape 7.01	Netscape

Examples

If you are like me, you learn best from examples. Studying how other authors use browser detection and cross browser coding techniques in the best way to learn.

Example 1 - object based feature detection

This example illustrates the use of feature detection. Note that Gecko 1.0 (Netscape 7) and later implement the proprietary Internet Explorer feature clientWidth while Netscape 6 did not. In this example, Netscape 7 and Internet Explorer 5+ will automatically use clientWidth while Netscape Navigator 4, Netscape 6, CompuServe 7 and Opera will use innerWidth.

Compare how you would have had to code this using vendor/version based detection approaches.

```
if (windowRef.document.body &&
typeof(windowRef.document.body.clientWidth) == 'number')
{
  // Gecko 1.0 (Netscape 7) and Internet Explorer 5+
  width = windowRef.document.body.clientWidth;
}
else if (typeof(windowRef.innerWidth) == 'number')
{
  // Navigator 4.x, Netscape 6.x, CompuServe 7 and Opera
  width = windowRef.innerWidth;
}
```

Example 2 - object based feature detection

Cross Browser Support

This example also illustrates the use of feature detection and shows the complications that can arise from the non standard implementations in other browsers.

```
function moveElement(id, x, y)
{
  // move the element with id to x,y
  // where x,y are the horizontal
  // and vertical position in pixels

  var elm = null;
  if (document.getElementById)
  {
    // browser implements part of W3C DOM HTML
    // Gecko, Internet Explorer 5+, Opera 5+
    elm = document.getElementById(id);
  }
  else if (document.all)
  {
    // Internet Explorer 4 or Opera with IE user agent
    elm = document.all[id];
  }
  else if (document.layers)
  {
    // Navigator 4
    elm = document.layers[id];
  }

  if (!elm)
  {
    // browser not supported or element not found
  }
  else if (elm.style)
  {
    // browser implements part of W3C DOM Style
    // Gecko, Internet Explorer 4+, Opera 5+

    if (typeof(elm.style.left) == 'number')
    {
      // Opera 5/6 do not implement the standard correctly
      // and assume that elm.style.left and similar properties
      // are numbers.
      elm.style.left = x;
      elm.style.top  = y;
    }
    else
    {
      // Gecko/Internet Explorer 4+
      // W3C DOM Style states that elm.style.left is a string
      // containing the length followed by the unit. e.g. 10px
      // Gecko will allow you to omit the unit only in Quirks
      // mode.
      // Gecko REQUIRES the unit when operating in Standards
      // mode.
      elm.style.left = x + 'px';
      elm.style.top  = y + 'px';
    }
  }
  else if (typeof(elm.left) == 'number')
  {
    // Navigator 4
    elm.left = x;
    elm.top  = y;
  }
}
```

Standards only

Consider how simple this function is if you code it according to the W3C standards.

```
function moveElement(id, x, y)
{
  // move the element with id to x,y
  // where x,y are the horizontal
  // and vertical position in pixels

  var elm = document.getElementById(id);

  if (elm)
  {
    elm.style.left = x + 'px';
    elm.style.top  = y + 'px';
  }
}
```

Ask yourself this: "Is supporting non-standard browsers worth the development and maintenance costs?"

Example 3 - Detecting specific Netscape Gecko branches

```
// return the rv value of a Gecko user agent
// as a floating point number.
// returns -1 for non-gecko browsers,
//          0 for pre Netscape 6.1/Gecko 0.9.1 browsers
//          number > 0 where each portion of
//          the rv value delimited by .
//          will be treated as value out of 100.
//          e.g. for rv: 3.12.42,
//          getGeckoRv() returns 3.1242
//
function geckoGetRv()
{
  if (navigator.product != 'Gecko')
  {
    return -1;
  }
  var rvValue = 0;
  var ua      = navigator.userAgent.toLowerCase();
  var rvStart = ua.indexOf('rv:');
  var rvEnd   = ua.indexOf(')', rvStart);
  var rv      = ua.substring(rvStart+3, rvEnd);
  var rvParts = rv.split('.');
  var exp     = 1;

  for (var i = 0; i < rvParts.length; i++)
  {
    var val = parseInt(rvParts[i]);
    rvValue += val / exp;
    exp *= 100;
  }

  return rvValue;
}


// determine if the browser is any Netscape Gecko
// branch >= 1.0.1 or Netscape 6.2.x/CompuServe 7
// built after August 1, 2002

var rv    = geckoGetRv();
var found = false;

if (rv >= 0)
{
  // Gecko browser
  if (navigator.productSub > '20020801')
  {
    if (rv >= 1.0001)
    {
      found = true;
    }
    else if (rv >= 0.0904 && rv < 0.0905)
    {
      if (navigator.vendor == 'Netscape6' || navigator.vendor == 'CS
2000 7.0')
      {
        found = true;
      }
    }
  }
}
```

Example 4 - The International Herald-Tribune

This site illustrates many of the techniques described in this article. They use downlevel pages for less capable browsers combined with object based feature detection to produce a compelling and interesting site.

Examples From DevEdge

- xbDOM
- xbMarquee
- xbPositionableElement
- xbAnimatedElement

Conclusion

As we have seen in this article, the browser detection story is still quite complicated due to differences between the modern browsers such as Netscape Gecko/Internet Explorer 6 and the older or non-standard browsers such as Netscape Navigator 4. You may say to yourself "If only all browsers were as good as Netscape Gecko and Internet Explorer 6, then my job would be so much easier!".

I would like to leave you with this thought. In the past, users did not have the choice of picking a browser which implemented the standards however today they do have a choice. There is no compelling reason for anyone in the world to continue to use a browser which does not support the standards. However, as long as web developers continue to code work arounds for these older browsers, users will not have a compelling reason to upgrade. By ceasing to support older browsers, you can provide a reason for users to upgrade. This will benefit not only them, but yourself as well. Supporting only standards based browsers can reduce development and maintanance costs as well as increase the dynamic and compelling content which will attract visitors and increase your revenue. The choice is yours... Decide to support the standards today!

Links

- Netscape Gecko Compatibility Handbook
- Mozilla user-agent strings

- Object Cross Reference - navigator
- Netscape Gecko User Agent Strings
- RFC 1945 - Hypertext Transfer Protocol -- HTTP 1.0
- RFC 2068 - Hypertext Transfer Protocol -- HTTP 1.1
- Client Object Cross References
- Quirks Mode in Mozilla
- CSS Enhancements in Internet Explorer 6
- W3C
- W3C Markup
- W3C Technical Recommendations
- W3C HTML 4.01
- W3C CSS 1
- W3C CSS 2
- W3C DOM Core 2
- W3C DOM HTML 2
- W3C DOM Style 2