About Element Positioning

[This documentation is preliminary and is subject to change.]

Windows Internet Explorer supports the ability to position HTML elements in x- and y-coordinates and to overlap elements in planes along the z-axis, which extends toward and away from the viewer in a Web document. These capabilities allow authors to precisely place elements, images, controls, or text on to a web page. By using scripts to manipulate the position coordinates and other dynamic styles, authors can move elements around a page, creating animated effects. The combination of dynamic styles, positioning, transparent Microsoft ActiveX Controls, and transparent images presents authors with a rich set of animation options.

- What Is Positioning?
- Absolute Positioning
- Fixed Positioning
- Relative Positioning
- Positioning Considerations
- Controlling Content Visibility
- Element Visibility

What Is Positioning?

Cascading Style Sheets (CSS) Positioning defines the placement of elements on a page and is an extension of cascading style sheets as specified in the W3C Working Draft on Positioning HTML with CSS. By default, elements flow one after another in the same order as they appear in the HTML source, with each element having a size and position that depends on the type of element, the contents of the element, and the display context for the element as it will render on the page. This default flow model for HTML layout doesn't allow a high level of control over the placement of elements on the page. By applying a small set of CSS attributes to the elements that are defined for the page, CSS can control the precise position of elements by giving exact coordinates. It is also possible to specify placement relative to the position of other objects on the page.

Just like any other HTML or CSS attribute, the CSS attributes used to control an element's position are available for scripting. The position of these elements on the page can thus be dynamically changed with script. As a result, the position of these elements can be recalculated and redrawn after the document is loaded without reloading the page from the server.

Controlling an elements position can make use of a variety of layout techniques. By employing specific parameters the site designer is in complete control over how much of an element is displayed on a page (if at all).

There are three ways to position an element in x- and y-coordinates. The type of positioning to use depends on the layout of the content and the purpose of the document. Absolute positioning means that the element is precisely placed relative to the parent coordinate system, regardless of any other content. Fixed positioning gives elements precise placement relative to the browser window, outside the flow of other content. Relative positioning places the item with respect to other elements on the page. Relative positioning depends on the default flow of the document, and reflows content should the user resize the browser.

Microsoft Internet Explorer 6 and greater: It is recommended that your !DOCTYPE be set to strict mode to enable rendering compliance with the W3C specification for Cascading Style Sheets, Level 2.1 (CSS2.1).
Note Windows Internet Explorer 7 and greater: Fixed positioning is available through strict mode.

Absolute Positioning

An absolutely positioned element is always relative to the next positioned parent. If there isn't a parent element, the containment block is used instead. Values for `left` and `top` are relative to the upper-left corner of the next positioned element in the hierarchy. For example, to place an image at the top left corner of the document, set the attributes to 0.

```
<img src="sample.gif" style="position:absolute; left:0px; top:0px">
```

This positions the image within the border of the HTML element. Be aware that the HTML element has a default border of 1. If you do not want two borders, set the border of the `body` to 0 to position the image at the 0,0 coordinates of the client area.

To see how a positioned parent affects the absolute position, consider the following example.

```
<div style="position:relative;left:50px;top:30px;height:100px;width:100px">
  Some text inside the DIV that will be hidden by the image because the image will be positioned over this flowing text.
</div>
```

This example places the `img` element at the upper-left corner of the `div` element, which is itself positioned on the page.

Setting an absolute position pulls the element out of the flow of the document and positions it without regard to the layout of surrounding elements. If other elements already occupy the given position, they do not affect the positioned element, nor does the positioned element affect them. Instead, all elements are drawn at the same place, causing the objects to overlap. You can control this overlap by using the `z-index` attribute to specify the order in which elements are stacked at the same location.

The contents of a positioned element flow within the given dimensions as default HTML would flow. For instance, text wraps normally based on the width of the element, and various inline elements contained within the positioned elements are placed next to each other in source order according to the constraints of size and shape of the container (that is, the positioned element).

Fixed Positioning

Beginning in Internet Explorer 7, web developers can use fixed positioning, a subcategory of absolute positioning. Similar to the containing boxes of absolute positioning, fixed position elements are independent. When applied, they are not relative to preexisting parent or child elements. The distinction involves element placement. Rather than localization on the `body` of a page (as with absolute positioning), the contents of a fixed positioned element flow within the given dimensions of the browser window (also known as the viewport).

The following example includes a typical layout representing fixed attributes.
In this example, the fixed containers are out of the flow of the page contents. Their positions remain relative to the viewport even as the browser window is resized. Once defined, they do not move from their assigned locations; making it possible for them to block elements beneath them. This makes careful placement a necessary consideration in your page design.

**Relative Positioning**

Setting the CSS `position` attribute to "relative" places the element in the natural HTML flow of the document and offsets the position of the element based on the preceding content. For example, placing a piece of text within a paragraph with "relative" positioning renders the text relative to the text in the paragraph that precedes it.

Should the user resize the window, the "xyz" still appears three pixels above the natural baseline of the text. You can set the left attribute in a similar way to change the horizontal spacing between "name" and "xyz". If the contents of the `span` were absolutely positioned, the "xyz" would be placed relative to the HTML element (or the next positioned element in the hierarchy), and the "xyz" would be barely visible at the upper corner of the client area - probably not the effect the author intended!

Text and elements that follow a relatively positioned element occupy their own space and do not overlap the natural space for the positioned element. Contrast this with an absolutely positioned element where subsequent text and elements occupy what would have been the natural space for the positioned element before the positioned element was pulled out of the flow.

It is quite possible that relatively positioned elements will overlap with other objects and elements on the page. As with absolute positioning, you can use the `z-index` attribute to set the `z-index` of the positioned element relative to other elements that might occupy the same area. By default, a positioned element always has a higher `z-coordinate` than its parent element so that it will always be on top of its parent element.
Positioning Considerations

The type of positioning to use depends on the layout of the content and the purpose of the document. Relative positioning depends on the default flow of the document and reflows content when the user resizes the browser. Fixed positioning, with its direct relation to the browser window also responds by reflowing when the window is resized. However, absolute positioning precisely places images and text no matter what the user does to the display.

Here is an example of nesting an absolutely positioned element within a relatively positioned element. The desired effect is to center text in a rectangle. In the past, you might use tables and attributes to center the content inside a table cell. However, this layout restricts you to a static table. Using positioning, this content can be worked into a larger layout, and then you can add scripting that might, for instance, have each of these elements fall into place from somewhere outside the document as the user loads the page!

```html
<html>
<head>
<title>Center the DIV</title>
<script language="JScript">
function doPosition() {
  two.style.top = document.all.one.offsetHeight/2 -
    two.offsetHeight/2;
  two.style.left = document.all.one.offsetWidth/2 -
    two.offsetWidth/2;
}
</script>
</head>
<body onload="doPosition()">
<p>Some text in the beginning.</p>
<div id=one
    style="position:relative;top:10px;height:200px;width:200px;"
    background-color:green">
  Some text in the outer DIV
  <div id=two style="position:absolute;left:50px;width:100px;color:red;"
    border:red 2px solid">text in the inner DIV - color should be red
  </div>
</div>
</body>
</html>
```

Click to view sample.

In the example above, the outer div flows with the contents of the document that precedes it, meaning it is positioned 10 pixels immediately after the first paragraph. The inner div has an initial absolute position, but this position is modified by the script function "doPosition" when the document is loaded. The offsetWidth and offsetHeight properties are used to calculate the new absolute position for an element. The example can also use the posLeft or pixelLeft property to center the images. These properties give alternate access to the left property, letting you set the position using a floating point number or an integer. There are similar properties that provide alternate access to the top, width, and height properties.

Combining Dynamic Positioning Techniques

The previous example can be expanded to manipulate multiple items on the page. If you were to animate this script, you might rework the scripting function on the documents onload event to have the inner piece of text "fly in" from offscreen. This function could be based on a timer that would move the
inner div from an initial top and left coordinate somewhere off the visible portion of the page, and, over a given amount of time, move it to a position that would be in the center of the outer div. The following example makes the div element visible and animates the content to glide across the screen. By setting an interval using the setInterval method on the window object, you can move one or more elements each time the interval elapses.

```html
<html>
<head><title>Glide the DIV</title>
<script language="JScript">
var action;
function StartGlide() {
    Banner.style.pixelLeft = document.body.offsetWidth;
    Banner.style.visibility = "visible";
    action = window.setInterval("Glide()",-50);
}
function Glide() {
    document.all.Banner.style.pixelLeft -= 10;
    if (Banner.style.pixelLeft<=0) {
        Banner.style.pixelLeft=0;
        window.clearInterval(action);
    }
}
</script>
</head>
<body onload="StartGlide()">
<p>With dynamic positioning, you can move elements and their content anywhere in the document even after the document has loaded!</p>
</body>
</html>

Click to view sample.

Note Dynamically changing an element from nonpositioned to positioned is only supported in Microsoft Internet Explorer 5 or later.

For more ideas on how to integrate positioning with other Dynamic HTML (DHTML) techniques, see Introduction to Filters and Transitions to learn how to incorporate visual filter effects.

Controlling Content Visibility

- Clipping Regions and Overflow
- Z-Index Ordering

In addition to controlling where on the page the element is positioned, the content of positioned elements can be restricted from the user's view in several ways. The display and visibility attributes control whether the element appears on the screen at all, and the clip and overflow attributes control how much of the content the user can see. Authors can also control the visibility of overlapping elements by manipulating the z-index ordering.

Clipping Regions and Overflow
You can set a clipping region for a positioned element by using the `clip` attribute. The clipping region defines a rectangle within which the element is visible. Any portion of the element that extends outside the clipping region is clipped, that is, not displayed. The clipping region does not alter the HTML, but simply changes how the element is displayed.

For example, the following document uses `clip` to define a clipping region for the absolutely positioned `img`. The region, a 50-by-50 pixel square, displays only a portion of the image; the rest is clipped from view.

```html
<html>
<head>
<title>Clipping an Image with clip</title>
<meta http-equiv="X-UA-Compatible" content="IE=8" />
<style type="text/css">
img {
    clip: rect(0px,50px,50px,0px);
    position: absolute;
    top: 50px;
}
</style>
</head>
<body>
<h2>Clipping an Image with clip</h2>
<img alt="ielogo" src="ie8_logo.jpg">
</body>
</html>
```

Click to view sample.

Be careful when you define a clipping region - the parameter order (top, right, bottom, left) is important. For example, setting `clip:rect` to `(0, 0, 50, 50)` causes the region not to display because the top and right have been defined as zero. The correct definition for a 50-by-50 clipping region based off the top left corner of the positioned object is to set `clip:rect` to `(0, 50, 50, 0)`.

You can change the clipping region dynamically by using the `clip` property, as in the following example.

```javascript
document.all.MyDiv.style.clip = "rect(0, 50, 75, 0)";
```

Overflow occurs when there is more content in a positioned element than can fit within the area defined for it. By default, any extra content is displayed but flows beyond the end of the area and therefore may overlap other elements in the document. You can prevent this overflow by using the `overflow` attribute to either hide the overflow or enable scroll bars to let the user view it by scrolling.

For example, the following document uses the `overflow` attribute to apply scroll bars. Only the
portion of the text that fits within the 100-by-100 area is initially displayed, but the user can scroll the rest into view by using the scroll bars.

```html
<html>
<head><title>Scroll the Overflow</title></head>
<body>
<div style="position:absolute;top:50px;left:50px;height:100px;width:100px;overflow:scroll">
some content some content some content some content some content
some content some content some content some content some content
some content some content some content some content some content
some content some content
</div>
</body>
</html>

Click to view sample.27

You can hide any overflow by setting the **overflow** attribute to hidden. Similarly, you can let the overflow flow beyond the end of the area by setting it to visible.

You can change the overflow dynamically by using the **overflow** property, as in the following example:

```html
if (document.all.MyDiv.style.overflow != "scroll")
    document.all.MyDiv.style.overflow = "scroll";
```

**Z-Index Ordering**

The z-index specifies in what order elements should be drawn when two or more elements occupy the same area. Setting the z-index is useful whenever you have absolutely or relatively positioned elements that overlap other elements in the document.

You set the z-index by using the **z-index** attribute. Setting this to a positive value causes the element to be stacked on top of other elements; negative values cause it to be stacked below. The following document uses z-index to stack text on top of the image.

```html
<html>
<head><title>Stack the Image</title></head>
<body>
<p style="position:absolute;top:0px;left:0px">Text Over Image</p>
<img src="sample.jpg" style="position:absolute;top:0px;left:0px;z-index:-1"/>
</body>
</html>
```
The element with the greatest $z$-index is always placed at the very top of the stack, and the element with the least $z$-index at the very bottom. If two elements have the same $z$-index, their source order determines the stacking (the last element is stacked highest).

**Note** Input from pointing devices, such as the mouse, does not penetrate through overlapping elements even if the elements are not visible. This is also true for positioned elements with a negative $z$-index unless the parent is a scrolling container (that is, its overflow property is set to auto or scroll), or the parent is manually positioned (that is, its position property is set to absolute, relative, or fixed).

You can change the $z$-index dynamically by setting the zIndex property, as in the following example.

MyImg.style.zIndex = 2;

In general, all elements are windowless and will participate in $z$-order overlapping. However, some objects are windowed. ActiveX Controls that have not been specifically written to be windowless will not overlap with other objects. iframe elements represent a window object, and window objects do not participate in $z$-order. Another exception is the select element which is also a windowed object in Internet Explorer 6 and lower. As of Internet Explorer 7, the select element is windowless and supports the $z$-index attribute and the zIndex property.

**Note** Although they are associated with different parts of the rendered page, fixed positioned elements are also subject to the relative stacking order of absolute positioned objects.

**Element Visibility**

The visibility of a positioned element determines whether the user can see the element. Visibility is useful for temporarily hiding an element, such as when time is needed to calculate its position or when carrying out simple transition effects. You can set the visibility by using the visibility attribute. The following example uses the attribute to make the div element invisible.

```
<p>A paragraph above the DIV element</p>
<div id=MyDiv style="position:absolute;top:50px;left:50px;height:100px;
width:100px;visibility:hidden"></div>
<p>A paragraph below the DIV element</p>
```

You can set the visibility of an element from script by using the visibility property. For example, assume that the div in the previous example is loaded. Initially, it is invisible, but you can change this to visible by using the following:

```
MyDiv.style.visibility = "visible";
```

An element that is not visible continues to affect the document layout; that is, the space that the
element would occupy remains in the document even though it is empty. This is unlike the display attribute, which does not reserve space. The following sample demonstrates the difference between visibility and display.

Click to view sample.

Send comments about this topic to Microsoft

Build date: 5/3/2012

Links Table

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="http://go.microsoft.com/fwlink/p/?linkid=203728" alt="Link" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms535242(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="http://go.microsoft.com/fwlink/p/?linkid=203760" alt="Link" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms530778(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>5</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms531177(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>6</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms535205(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>7</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms535259(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>8</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms535240(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>9</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms531188(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>10</td>
<td><img src="http://go.microsoft.com/fwlink/p/?linkid=237806" alt="Link" /></td>
</tr>
<tr>
<td>11</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms531140(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>12</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms535895(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>13</td>
<td><img src="http://go.microsoft.com/fwlink/p/?linkid=237800" alt="Link" /></td>
</tr>
<tr>
<td>14</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms534304(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>15</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms534199(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>16</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms531143(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>17</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms531129(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>18</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms531183(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>19</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms530765(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>20</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms536749(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>21</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms535873(v=vs.85)" alt="Link" /></td>
</tr>
<tr>
<td>22</td>
<td><img src="http://go.microsoft.com/fwlink/p/?linkid=237801" alt="Link" /></td>
</tr>
<tr>
<td>23</td>
<td><img src="http://msdn.microsoft.com/en-us/library/ms533035(v=vs.85)" alt="Link" /></td>
</tr>
</tbody>
</table>
Absolute to Fixed Position

The problem of absolute positioning in IE8 may be resolved by using the "fixed" value for the position attribute rather than "absolute." However, I have not tried this with printing, as the problem described previously is due to the image element appearing at the end of the document. Elements are "absolutely" positioned relative to the containing element, which in print media, would be the "page" containing the element, or the final page in this case. Another possible solution might be to place the Image Element near the beginning of content, so that it is rendered as part of the first page.  {Note that since both previous posts are rather old, this reply is meant to help future readers resolve similar issues.}
**problem with absolute position in IE8 while printing.**

When I was working with absolute position in IE8 with print, I encountered a problem.

I have an image at the end of the page, while printing I will set an absolute position (top 0px and left 0px) so that the image should come in as a header. It works fine if the page content is with in 1 page. If the page content exceeds the page then the image is getting placed in the top of 2nd page or 3rd page depending upon the content (in terms of pages). The absolute position is considered as last page's absolute position instead of 1st page. This happens only while printing, while displaying it display’s correctly.

Why is that so? Is their any alternate solution to this?

The absolute position works fine with IE6 while displaying as well as while printing.

© 2012 Microsoft. All rights reserved.
About Font Embedding

46 out of 72 rated this helpful - Rate this topic

[This documentation is preliminary and is subject to change.]

Unlike traditional print-based designers, Web page designers are limited to specifying fonts installed on a user's computer. This effectively limits the fonts that can be used on a Web page to those that are typically installed on a user's computer. For many designers, this is a problem because few fonts are consistently installed on users' computers. The Microsoft OpenType and TrueType specifications provide a solution to this problem, document font embedding.

Font embedding has long been a feature of Microsoft applications, such as Microsoft Word and Microsoft PowerPoint. It enables fonts used in the creation of a document to travel with that document, which ensures that a user views the document exactly as the author intended. Windows Internet Explorer has supported embedded fonts since Microsoft Internet Explorer 4.0.

Embedded Font Technology

In the early years of the World Wide Web, browsers had full control over the fonts used to display content and users were able to override the defaults fonts and specify alternate choices. Later, the font object and Cascading Style Sheets (CSS) font-family property was introduced to allow Web designers more control over the appearance of their content. For example, the font-family property lets designers specify a prioritized list of fonts for the browser to use to display content. If the preferred font is not installed on the user's computer, the browser will attempt to use the second font in the list. If the user does not have of the preferred fonts installed, a default font is used to display the content.

Internet Explorer supports font embedding, which allows fonts to be temporarily installed on a user's computer so that a Web page is displayed exactly as the Web designer intended.

Although the mechanism to specify fonts is defined by CSS, the font format itself is not called out. In conjunction with the font community, Microsoft adapted the embedding technology used by Microsoft Office for use on the Web, creating the Embedded OpenType (EOT) font format. Based on OpenType, the EOT format encapsulates a font and binds it to a specific web page or site. This meets the OpenType definition of document font embedding.

The EOT format has been documented and submitted to the World Wide Web Consortium (W3C). For more information, see Embedded OpenType (EOT) File Format.

Different Levels of Font Embedding

OpenType and TrueType fonts include permissions defined by the original publisher of the font to detail when and how a may be embedded in a document. The OpenType and TrueType specifications define four primary levels of font embedding:

- **Print and preview** fonts can be embedded in a document, provided the user reading the document cannot edit the content of the document.
- **Editable** fonts can be embedded within content that can be edited by the user.
- **Installable** fonts within a document may be permanently installed by the user reading the
document or a client application. In practice, installable fonts are treated like editable fonts by most client applications.

- **No embedding** permissions prevent fonts from being embedded in a document and are used by a small proportion of available fonts.

Most fonts are licensed from their original publishers, who may place additional restrictions on their use and distribution. For information about licensing arrangements for a given font, contact the original publisher.

**Embedded Font Examples**

The following demonstrations, located on the Microsoft Typography Web site, show how font embedding is a dramatic improvement over the traditional use of fonts on the Web.

**Example 1: Healthy Eating Recipe**

**Design process.** This page was designed around a simple HTML table. CSS features were used to specify fonts and ensure the correct alignment of the text and symbols. At this stage the fonts used were all installed locally.

When the copy and layout were set, Microsoft WEFT was used to create "font-objects" that were linked to the page. The WEFT tool analyzes the font usage of Web pages, gathers the required characters from each font used, and creates the compressed font objects. It also modifies the HTML page by writing in the CSS code that links the font objects to that page. If you use View Source on the demonstration page, you'll see the format of the code that was added.

```html
@font-face {
  font-family: Goudy Stout;
  font-style: normal;
  font-weight: 700;
  src: url(GOU DyST0.eot);
}
```

The preceding code instructs Internet Explorer 4.0 to use the GOU DyST0.eot font object whenever there is text on the page specified in the Goudy Stout font. The following table shows the fonts used by the page, the number of unique characters used, and the size of the font object that contains them.

<table>
<thead>
<tr>
<th>Font used</th>
<th>Unique characters</th>
<th>Object size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pie</td>
<td>9</td>
<td>4.27K</td>
</tr>
<tr>
<td>Goudy Stout Bold</td>
<td>15</td>
<td>7.43K</td>
</tr>
<tr>
<td>Garamond Bold</td>
<td>15</td>
<td>10.3K</td>
</tr>
<tr>
<td>Garamond</td>
<td>19</td>
<td>10.3K</td>
</tr>
<tr>
<td>Script MT Bold</td>
<td>49</td>
<td>3.97K</td>
</tr>
</tbody>
</table>

**How the page looks in other browsers.** Browsers that don't support font embedding ignore the code that links the font objects to the page. Those using browsers that support font embedding will
see text displayed in the selected font if it's installed. Since Goudy Stout and Garamond come with Office, it's possible the fonts may be installed. If the first-choice font isn't available, the browser displays the text using the second- or third-choice fonts, if they are installed. Users of older browsers see all the text displayed using their default font.

One thing to bear in mind is that symbol fonts appear as ordinary characters in browsers other than Internet Explorer—even CSS browsers do not display the symbols properly. For this reason, using symbol fonts in this way is recommended only for content specific to Internet Explorer.

**Example 2: A Blot On The Copybook**

**Design process.** This demonstration consists of four pages and shows how to use one font object across any number of pages. The French Script MT font object contains all the letters and punctuation used in the story and is referenced by all four pages.

Unlike the characters used in the text of the story, the pictures appear only once. For this reason, four font objects were created, each containing only the pictures used by the page that references it.

The third font is a custom font created specifically for this demonstration. The phrase "A Blot on the Copybook" was drawn as a single character using Macromedia Fontographer, and mapped to the letter Q. The phrase "fin" was also drawn and mapped to the letter K.

<table>
<thead>
<tr>
<th>Font used</th>
<th>Unique characters</th>
<th>Object size</th>
</tr>
</thead>
<tbody>
<tr>
<td>French Script MT</td>
<td>52</td>
<td>11.1K</td>
</tr>
<tr>
<td>Copyblot</td>
<td>2</td>
<td>4.23K</td>
</tr>
<tr>
<td>Dingblots (p. 1)</td>
<td>3</td>
<td>2.56K</td>
</tr>
<tr>
<td>Dingblots (p. 2)</td>
<td>27</td>
<td>8.42K</td>
</tr>
<tr>
<td>Dingblots (p. 3)</td>
<td>17</td>
<td>5.47K</td>
</tr>
<tr>
<td>Dingblots (p. 4)</td>
<td>13</td>
<td>3.57K</td>
</tr>
</tbody>
</table>

**How the page looks in other browsers.** Like the preceding demonstration, this one uses an embedded symbol font. Two pages also use a custom font that maps the phrases "A Blot on the Copybook" and "fin" to the Q and K keys, respectively. Using custom and symbol fonts in this way means that the pages are specific to Internet Explorer.

**Example 3: Typographic Ornament**

**Design process.** These pages use HTML tables to achieve checkerboard patterns, which are colored and filled with various symbols taken from the Border Web and Kingston inline fonts.

<table>
<thead>
<tr>
<th>Font used</th>
<th>Unique characters</th>
<th>Object size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kingston Inline</td>
<td>36</td>
<td>17.1K</td>
</tr>
<tr>
<td>Runic MT Condensed</td>
<td>33</td>
<td>9.85K</td>
</tr>
<tr>
<td>BorderWeb</td>
<td>10</td>
<td>8.36K</td>
</tr>
</tbody>
</table>
How the page looks in other browsers. Like the previous demonstrations, these pages also use embedded symbol fonts, making them specific to Internet Explorer.

**Example 4: Progressive Rendering**

**Design process.** Internet Explorer 4.0 does not render a page until the font objects referenced by that page are downloaded, decompressed, and temporarily installed. In some cases you might want to have a page displayed using an installed font that is dynamically replaced by the font stored within the font object.

The code required to implement this progressive rendering is very compact and takes the following form.

```javascript
<SCRIPT LANGUAGE="JavaScript">
function createbreak() {
    alert("Click here to continue the demo");
    document.styleSheets(0).href="ftembed.css";
}
</SCRIPT>

The "styleSheets(0)" section of the preceding code is the element of the style sheet collection you want to replace. In this example, a null style sheet corresponding to styleSheets(0) is dimensioned so as to create a style sheet element. This requires you to add the following code in the head object of your HTML code.

```
<LINK rel=stylesheet href=null>
```

The font object links are contained in the external style sheet and are therefore referenced only after the page itself is downloaded. The alert code has been inserted so that users with fast connections can see how the progressive rendering effect works. You can remove this line of code if you implement it on your own pages.

<table>
<thead>
<tr>
<th>Font used</th>
<th>Unique characters</th>
<th>Object size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Braggadocio</td>
<td>18</td>
<td>3.16K</td>
</tr>
<tr>
<td>Curlz MT</td>
<td>36</td>
<td>10.4K</td>
</tr>
<tr>
<td>OCR A Extended</td>
<td>19</td>
<td>4.75K</td>
</tr>
<tr>
<td>Gradl</td>
<td>23</td>
<td>4.08K</td>
</tr>
<tr>
<td>Snap ITC</td>
<td>29</td>
<td>7.09K</td>
</tr>
</tbody>
</table>
Internet Explorer Security Alert

Depending on how Internet Explorer security settings are set, you might receive a warning every time a page accesses an embedded font. Although your security level settings can easily be modified, you should first access the Internet Explorer online help and read the sections discussing security.

The Font Download security setting allows you to control the behavior of this dialog box for each security zone. For more information, see URL Security Zones.

- On the View menu, click Options and then click the Security tab.
- Select Custom and click Settings.
- Scroll to the Downloads section.
- Change the Font Download setting from Prompt to Enable.

Using Microsoft WEFT

The Microsoft WEFT is a free utility that lets site designers create font objects that are linked to their Web pages.

The font objects created by WEFT differ from traditional font files in a number of ways. The font objects are compressed and usually only contain the subset of characters used by a particular site or document. Font objects are privately installed by Internet Explorer so that they can't be accessed by other applications or by Web sites that do not have permission to use them.

WEFT Style Code

WEFT adds a style section to the head part of each HTML page that uses one or more font objects.

<STYLE>
@font-face {
    font-family: Garamond;
    font-style: italic;
    font-weight: normal;
    src: url(Garamond1.eot); }
</STYLE>

The preceding code instructs Internet Explorer to use the Garamond1.eot font object whenever the Garamond Italic font is specified within the page. The browser uses the font object regardless of whether the font is specified using the FONT FACE tag, a linked or inline cascading style sheet, or some other method.

CSS Files

WEFT uses Internet Explorer to determine the fonts and characters used on each page. If a page references a linked style sheet, this style sheet usually determines the fonts Internet Explorer uses to display the page. WEFT does not modify linked style sheets. Instead, the tool modifies the HTML pages by adding code that links the font objects to pages that require them.

In some cases it is more efficient to reference the font objects within one or more linked style sheets. If you'd like to try this, skip the option that asks you if you want to publish the modified pages back to your Web site. Instead, cut and paste the relevant code from the modified pages (by default these are stored in the My Documents folder) into your linked style sheet.

A word of caution is needed. As style sheets cascade, a font specified in a linked style sheet might
be overridden by one specified as inline. Also, because a style sheet can be linked to any number of pages, care should be taken when choosing an appropriate character subset.

For more information about CSS, see the CSS Attributes: Index

Frequently Asked Questions

Q. Which fonts can I use?

WEFT supports OpenType and TrueType fonts in the TrueType format. However, WEFT does not let you embed certain categories of fonts, such as those with no embedding permissions or fonts containing serious errors.

Q. How can I check the embedding permissions of my fonts?

You can check the embedding permissions of any font using the Microsoft Font Properties Extension. WEFT also has a font-checking feature that reports the permissions of fonts installed on your system.

You might come across old TrueType fonts or converted Macintosh TrueType fonts that do not have any embedding permissions encoded within them. WEFT treats these as "no-embedding" fonts. In the unlikely event that you encounter such a font, your best option is to contact the original publisher of the font and ask about an upgrade.

In addition, fonts are typically published with certain restrictions regarding use and distribution. The original publisher can provide detailed information regarding licensing requirements and opportunities.

Q. Should I embed common core fonts such as Arial or Times New Roman?

WEFT identifies common fonts that your readers are likely to have installed. These include Windows core fonts such as Arial, Times New Roman, and Courier New. You can still embed these fonts; however, Internet Explorer 4.0 downloads, decompresses, and temporarily installs font objects, even if the real font is present on the user's computer.

Send comments about this topic to Microsoft

Build date: 5/3/2012

Did you find this helpful?  Yes  No

Community Content

I can't embed to IE browser

I found this, hope it is helpful for all:

Name: Andy  
Website: http://hyundaioto.com.vn

I can't embed to IE browser

When I used css style: @font-face, it's work on Firefox but IE was not.

http://hyundaioto.com.vn

Weft Is failed

Your Weft technology is failed, neither proper support available for the same nor good example i tested it many times but it does not work any time if the font is not available on the client machine so there is no need of WEFT.

Crashes

Crashes using Windows 7

Unfortunately, the weft tool crashes on my Win 7 64bit

Unfortunately, the weft tool crashes on my Win 7 64bit

Additional informations

Microsoft Font Properties Extension  
http://www.microsoft.com/typography/property/fpedit.htm

WEFT stands for Web Embedding Fonts Tool  
http://www.microsoft.com/typography/WEFT.mspx
An AJAX driven Joomla! 1.5.x component to place a contact form anywhere on your web page with any number of custom fields of different types, including attachments.

In the component pack you also get:
- mod__aiContactSafe : a modul to display the contact form in any position of your template
- plg__aiContactSafe Form : a plugin to place the contact form in an article.
- plg__aiContactSafe Link a plugin to place a link to the contact form in an article.

You can install and/or upgrade any of them from the Control Panel of aiContactSafe and see more information about all of them in the articles of algisinfo.com.

Some of the features of aiContactSafe are:
- CAPTCHA with the possibility to specify code words for each language or random strings, easily integrated in your design by settings the colors and background color to use
- profiles ( multiple forms with different custom fields )
- unlimited custom fields of different types ( Textbox, Checkbox, Checkbox list, Combobox, Radio list, Editbox, Date, Email, Email list, Hidden, Separator, Joomla contact, File )
- add a prefix or suffix to the field
- redirect to specified page when the message is successfully sent
- any number of attachments working with or without AJAX
- SEF support through ARTIO JoomSEF or sh404SEF, or you can use the Joomla's native SEF
- multilingual support ( language files & Joom!Fish )
- language files for English, Arabic, Bulgarian, Czech, Danish, German, Greek, Hungarian, Spanish, French, Italian, Dutch, Polish, Brazilian Portuguese, Portuguese, Romanian, Russian, Slovak, Serbian (Cyrillic), Swedish, Turkish, Ukrainian, Lithuanian ( some not completed )
- specify contact information
- detailed instructions for each administration page in English ( and some of the translation mentioned above )
- block users by IP
- block messages that contain words from a list you can define
- automatically ban IP that sends messages with blocked words in a specified period of time
- record the messages in the database with the IP from which they've been sent ( can be disabled from Control Panel )

Developer: Algis Info Grup
Website: Website

Editor's Note
This extension requires registration to download.
aiContactSafe - Joomla! Extensions Directory

- extended configuration capabilities
- MVC architecture
- email templates
- order the fields for each profile

REVIEWS (357)

🌟🌟🌟🌟🌟 Its the best component contact

![User Avatar]

rfilgueiras
2012-06-26
Review s: 1

It's a very good component. I didn't have any problem with it.
Congratulations

🌟🌟🌟🌟🌟 Perfect App

![User Avatar]

stoltravolta
2012-06-26
Review s: 1

This is one of the best apps that I have found for Joomla. The support is also very quick and very helpful. The flexibility is amazing.
Yes it is a bit tricky to get the hang of it at first, but a couple of profiles later and it's second nature.
I would recomend this to anyone looking for decent contact forms.

🌟🌟🌟🌟🌟 Great extension

![User Avatar]

neotrix
2012-06-21
Review s: 1

The extension top of my sites.
Congratulations developer

🌟🌟🌟🌟🌟 Great Job and great support team

![User Avatar]

vaughan
2012-06-21
Review s: 1

I'm really pleased with this component, the best contact form by far, is free and has all the features you need, fully configurable AND A GREAT TEAM TO SUPPORT ALL DOUBTS RESOLVED, CONGRATULATIONS

🌟🌟🌟🌟🌟 Question answered prompt!!

![User Avatar]

dsalchow
2012-06-17
Review s: 1

great tool and great support. This component simple and perfect.

🌟🌟🌟🌟🌟 is highly recommended

![User Avatar]

Tryout
2012-06-16
Review s: 3

It's a really good component contact form easy to set up and use, good response from support, this is the best component I've used have tried some over time ... try it will not be disappointed

🌟🌟🌟🌟🌟 It's a breeze

![User Avatar]

fffedele
2012-06-15
Review s: 2

As a rookie, I was surprised to how quickly I could get it working and it works. Certainly has impressed me.

🌟🌟🌟🌟🌟 Highly flexible and useful
I definitely recommend this component to build custom contact form, it's highly customizable and flexible. Maybe what's not so intuitive is some logic about default fields or how to customize layout but it is still a great component.

Great contact component with every features you'll ever need. 1 of 1 people found this review helpful

aiContactSafe is a great contact component and it has every features you'll ever need. Thanks Alex for this great product.

sef support

It's really good, but can't support sh404SEF, but you write that... don't work sef url!

Perfect tool

Excellent form, parameterized and a very ready and very illuminating answer in the forum about the doubts that always arise during the installation.

Recommended for those who want to set various forms on your web page.

Take this opportunity to thank the developer for free distribution of this magnificent tool.

Great extension

This is an excellent extension. Congratulations Alex!!

Excellent!

Lots of options to choose! Thank for your extension.

Good support

I needed support for something very simple and the response was quick and with great patience -best of all-. The component is very simple and easy to use, css handling is limited but enough to have a neat and simple form without much effort. something very simple and the response was quick and with great patience and best of all ... The component is very simple and easy to use, css handling is limited but enough to have a neat and simple form without much effort.

Professional extension

This is a complete professional extension, simple and perfect. Great job Thank you

AIContactSafe is Great!
I have used this extension on two sites. It is relatively easy to figure out and when I had a question the developer was extremely prompt and courteous with an answer. Also there is an active support forum where I was able to search and get the answer to my other question.

The form is attractive on the page, too, and comes with a Captcha element--two more pluses.

Thank you to the developer for making such a nice extension.

---

Great extension but...

Hi,

the extension is great but I think to embed the form into the article one should NOT use ... (as described on your tutorial page, not working) but `{ aicontactsafe form pf=1}` (then it's working)

---

A must have!

Fully customizable & Absolutely Beautiful extension... A definite MUST HAVE 4 any Joomla website!.. Thanks 4 sharing!!!

---

Simply the Best

Have been using this extension for years on all my Joomla sites because it is easy to use. Has everything, captcha, required fields, multiple forms, any kind of input, email templates and it is code compliant!

By far the best in my book.

---

Excellent extension

I would like to recommend this extension to anyone needing a very flexible and extended, yet quite easily configurable and logically made solution for contact forms.

One of its greatest advantages to me is the possibility to use it as a plugin, with all parameters configurable from within the command line. This fact makes content-aware forms possible and easy to implement.

The only drawback that I've noticed so far is that it's not possible to send two different messages with one form (one to sender - like a confirmation - and the other to receiver).

But the best thing of all is the absolutely relevant, professional and quick support from the extension developer!
Controlling Presentation with Measurement and Location Properties in Quirks Mode

[This documentation is preliminary and is subject to change.]

Dynamic HTML (DHTML) exposes measurement and location properties that you can use to change the size and position of HTML elements on your Web pages. When you understand what these properties are and how they affect elements on a page, you can achieve greater control over the appearance of your Web pages. For example, you can use these properties to design pages that are similar to documents in other applications, such as Microsoft PowerPoint or Microsoft Word. This article includes the following sections, which explain how to use measurement and location properties to control the appearance of a Web page.

Note This document describes the layout and positioning properties of a page rendered in Windows Internet Explorer's Quirks mode. You do not need to define a `!DOCTYPE` for the examples on this page to function properly. For layout and positioning information under the strict rendering mode, please view Controlling Presentation with Measurement and Location Properties In Strict Mode.

**Requirements**

This documentation assumes that you are familiar with a client-side scripting language, such as Microsoft JScript or Microsoft Visual Basic Scripting Edition (VBScript). You should also be familiar with the DHTML Document Object Model (DOM) and Cascading Style Sheets (CSS).

**Layout Fundamentals**

Measurement and location properties are available through the DHTML Document Object Model (DOM) and as CSS attributes. You can use DHTML DOM properties to programmatically set CSS attributes. Properties exposed through the DHTML DOM return values based on how an element renders in the document. CSS attributes return values based on the preset values of other CSS attributes.

All CSS properties are exposed through the `style` and the `runtimeStyle` objects. You can use the `currentStyle` object to query the current value of a property.

**The Element Rectangle**

Every visible element on a Web page occupies an absolute amount of space in the document. The amount of space occupied by an element is defined by the element rectangle or box. An element rectangle includes all of the layout and display properties plus any content.

In the preceding graphic, the `margin`, `border`, and `padding` properties are shown surrounding the content of a generic element. "Element Width" represents the width of the element's content, and "Box Width" represents the width of the content plus the additional space occupied by the layout properties. The height of an element and its layout properties can be represented similarly.
Block Versus Inline Elements

All visible HTML Elements\(^{12}\) are either displayed in blocks or inline. A block element, such as a div\(^{13}\), typically starts a new line and is sized according to the width of the parent container. An inline element, such as a span\(^{14}\), typically does not start a new line and is sized according to the height and width of its own content.

Size, Layout, and Location of Elements

An element has layout when one of the following conditions is true:

- It has a width and/or a height specified
- It is an inline-block (display: inline-block)
- It has absolute positioning (position: absolute)
- It is a float (float: left, float: right)
- It is a table element
- It is transformed (style="zoom: 1")

Nearly all inline and block elements have layout. The exception is an inline element that is neither positioned nor has its height or width specified.

A positioned element has specific measurements and can be set to a location using the CSS layout attributes: top, right, left, bottom. The location of non-positioned elements are relative to their nearest ancestor with layout (offsetParent\(^{15}\)). Location is useful when moving one or more elements to relative or absolute coordinates within the document. It can also be useful for creating specific document styles.

Measurement Fundamentals

Height and width are the measurements used most frequently. Use the offsetHeight\(^{16}\) and offsetWidth\(^{17}\) properties to retrieve these measurements from any rendered inline or block element. To set these measurements, use the height\(^{18}\) and width\(^{19}\) attributes (or related properties). For a CSS measurement attribute to be useful, you must set the attribute's value before retrieving it.

Inline elements gain layout when the height or width are set. Inline elements with layout expose the same layout properties, such as border\(^{10}\), margin\(^{9}\), and padding\(^{11}\), as block elements. The following example shows how layout attributes affect the appearance of a web page when the height\(^{18}\) attribute is set on an element.

<!-- Styles render because this link has layout. --><!--a href="http://msdn.microsoft.com/"
   style="width:150px; border:1px solid; padding:10px; margin:5px;">
   MSDN Online</a>

<!-- Styles do not render because this link has no layout. --><!--a href="http://msdn.microsoft.com/"
   style="border:1 solid; padding:10px; margin:5px;">
   MSDN Online</a>

Absolute and Relative Length Units

When setting or retrieving an element's measurement and location values, you can use different length...
units\textsuperscript{20} to achieve a particular style. Using length units consistently simplifies measurement and location values. The mixing of length units requires you to determine the value of an absolute length unit programmatically, based on a relative unit for every system. For example, you would have to convert inches to pixels on every machine that renders your document.

Layout properties contribute to the dimensions of an element and should be considered when determining the dimensions of the content. The content of an element is sized to any specified measurements minus the border and padding measurements. Because the DHTML and CSS properties do not provide a measurement of the content without its padding, the padding properties must be specifically queried to retrieve an accurate measurement. While you can use the offset and client properties to determine the size of the content, it is easier to subtract the size of the border and padding properties from the width of the element.

Consider the following div\textsuperscript{13} element:

\begin{verbatim}
<div id="oDiv"
   style="padding:10px; width:250px; height:250px;
           border:2px outset; background-color:#CFCFCF;">
</div>
\end{verbatim}

The specified height and width of the preceding div\textsuperscript{13} is 250 pixels. You can formulate the width of the content as follows:

\begin{verbatim}
oDiv.style.width - oDiv.style.borderWidth - oDiv.style.padding
\end{verbatim}

However, since all three properties return variant data types, you must either convert the values to integers or use properties that return integer values. For example, to obtain the width of the element in pixels, you can use one of the following techniques in JScript:

\begin{verbatim}
var iWidth = oDiv.style.pixelWidth
\end{verbatim}

\begin{verbatim}
var iWidth = parseInt(oDiv.style.width) (width is specified in pixels)
\end{verbatim}

\begin{verbatim}
var iWidth = oDiv.offsetWidth
\end{verbatim}

If the values of the border and padding properties are set in the same length units as the width property, you can convert the variant value to an integer. To determine the content dimensions, use the border and padding properties, or the client and padding properties, and the element dimensions. When retrieving border and padding values, use the border\textit{Width}\textsuperscript{21} and padding properties, if the
values are uniform on all sides of the element. Otherwise, you must specifically query the `borderLeftWidth`, `borderRightWidth`, `paddingLeft`, and `paddingRight` properties to obtain an accurate measurement.

For example, to obtain the width of the content in pixels, you can use one of the following techniques in JScript, where iWidth is based on one of the previous example techniques.

- Use this formula if the border and padding sizes are not uniform on all sides.

```javascript
var iTotWidth = (iWidth - parseInt(oDiv.style.borderLeftWidth) - parseInt(oDiv.style.borderRightWidth) - parseInt(oDiv.style.paddingLeft) - parseInt(oDiv.style.paddingRight));
```

- Use this formula if the border and padding sizes are uniform on all sides.

```javascript
var iTotWidth = (iWidth - parseInt(oDiv.style.borderWidth) - parseInt(oDiv.style.padding));
```

- Use this formula instead of using the border properties.

```javascript
var iTotWidth = (iWidth - (oDiv.offsetWidth - oDiv.clientWidth) - parseInt(oDiv.style.padding));
```

**Note** These formulas will not work if padding or margin are specified using percentages.

**Measurement Example**

The following example uses the first formula from the preceding section to move and resize a positioned element based on the content of another element.

```javascript
<script>
window.onload=fnInit;
function fnInit(){
    var iWidth=oDiv.style.pixelWidth;
    var iHeight=oDiv.style.pixelHeight;
    var iTotWidth=(iWidth - parseInt(oDiv.style.borderLeftWidth) - parseInt(oDiv.style.borderRightWidth) - parseInt(oDiv.style.paddingLeft) - parseInt(oDiv.style.paddingRight));
    var iTotHeight=(iHeight - parseInt(oDiv.style.borderTopWidth) - parseInt(oDiv.style.borderBottomWidth) - parseInt(oDiv.style.paddingTop) - parseInt(oDiv.style.paddingBottom));
    var iTotTop=oDiv.offsetTop + parseInt(oDiv.style.borderTop);...
```
Location Fundamentals

The portion of Internet Explorer that displays the document is referred to as the client area. Beginning in the top left corner with x and y coordinates set at 0, the client area has no inherent margin or padding. The **BODY** element is the first container in the document and is the top-most parent. Like the client area, the **BODY** has no default margin or padding, and also begins at x and y coordinates of 0.

The distance between the element and its positioned or **offsetParent** defines the element location. Internet Explorer exposes the element location when the document renders or when a change to the content forces the document to redraw. Understanding how elements are located within the document is key to determining and changing the location of an element.

The general layout of elements is primarily based on document flow. Document flow is analogous to the **writingMode** attribute. In Western languages flow indicates the layout of content from left to right, top to bottom. Inline and block elements that do not have absolute positioning flow in this manner by default; unless otherwise specified. Elements having absolute positioning render outside of the document flow. The document flow is the order of the elements after their measurements are calculated. Changing the measurements or location of positioned elements does not affect adjacent elements in the document flow, but it might affect **child** elements.

Non-positioned inline and block elements render together and are part of the document flow. An element's location can change when the measurement of another element appearing earlier in the document flow changes. The measurement of an element changes when the content, layout, or font style of another element is updated after the document renders. Changing the measurements of a non-positioned element changes the location of adjacent elements in the document flow.

Top and Left Locations

The following example shows how to retrieve the location of an inline element and how the measurement of another element affects the location of that element.

```html
<script>
function getLocation(){
    alert("Left: " + oSpan.offsetLeft);
    oSpan1.innerHTML="Changed content.");
    alert("Left: " + oSpan.offsetLeft);
}
</script>
<div id="oDiv"
    style="padding:20px 5px 10px 10px; width:250px; height:250px;
        border:10px outset; background-color:#CFCFCF;">
</div>
<div id="oDiv2" style="position:absolute; border:2 inset;
    background-color:#000099;">
</div>
```
Offset Parents

Although you can retrieve the top or left location of any element that renders, the values of these locations are relative to the positioned or offsetParent. You cannot always rely on a single value when determining the distance between two elements.

The following image depicts the offsetLeft values of a span element and a table element. The offset value for both elements is 50 pixels. However, if you query the offset value of a table cell, the offsetLeft property returns only 3 pixels, because table is the offsetParent of the td element (Cell 1). To determine the distance from the table cell to the edge of the screen, add the two values together.

Special Cases

Certain elements, such as the table element, expose their own object model and a specific set of properties, such as the cellPadding property and the cellSpacing property. From the currentStyle object, you can retrieve the current value of the cellPadding property through the padding property. You must retrieve the cellSpacing property directly from the table element.

Nested Elements

Determining the distance between any nested element and its offsetParent, such as the BODY element, might require you to include the location of the offsetParent. For example, the top and left locations of a td element return the distance between the cell and the offsetParent, which is the table element. To determine the distance between a nested element and the BODY element, you must walk up the document hierarchy and add the left location values of all the offsetParents between the two elements.

Location Example

The following example shows how to programmatically retrieve the absolute distance between a td element and the BODY element.

```html
oSpan1.innerHTML="This is some dynamic content.";
}
</script>
</body>
</html>
```
```javascript
oSpan1.innerHTML = "This is some dynamic content.");

This content won't change, but this element's location will change.

<input type="button" VALUE="Locate Second Element" onclick="getLocation()" />
</body>
```
Positioned Elements

The measurements of an element with absolute positioning are first defined by any specified measurement properties, such as height or width, followed by its content. This is true for block and inline elements. However, a relatively positioned element retains its block or inline measurements unless otherwise specified.

The following example shows that the measurements of a block element and an inline element are the same when the position attribute is set to absolute.

You can use the same procedure to set or retrieve the measurements of positioned elements and non-positioned elements.

How to Use Measurement and Location Properties

Measurement and location values do not have to be static integers. They can be scripted values based on distances and dimensions of other elements, expressed lengths (as in more traditional media) or equations. When working with several elements, you can use the measurement of one element to set the location of another.

You can use the measurement and location properties discussed in this overview to construct more complex creations. For example, to center an element within its container, set its left coordinate to the sum of one-half the width of its container minus one-half the width of the element. The syntax is demonstrated in the following example.
Expressions are formula-derived values. You can use expressions to continually update property values. In the preceding example, the element is centered once, but does not remain centered if the browser window is resized. To preserve a particular layout, use a formula rather than a static value.

Use the `setExpression` method to add expressions to CSS attributes and DHTML properties. The following example shows how to center an element using an expression rather than a static value.

```html
<script>
function center(oNode)
    oNode.style.left=oParent.offsetWidth/2 - oNode.offsetWidth/2;
</script>
<div id="oDiv" onclick="center(this)"
    style="position:absolute;">Click Here To Center</div>
</script>
```

While the DHTML DOM provides an easy way to obtain the current dimensions and location of an element, you must use CSS to set these values in most cases. Except for elements that expose `height` and `width` properties, such as `img` and `table`, you can use various CSS attributes to set the size of an element. While many properties return values in pixels, you can use some CSS properties with specific length units, such as inches or centimeters.

For example, if an `H1` element has top and left positions of 2 inches, the `offsetTop` and `offsetLeft` properties return approximately 190 pixels, depending on the screen resolution. Since the `top` and `left` properties return a string value of `in`, you can use the `posTop` and `posLeft` properties to increment or decrement the location in inches rather than pixels. The `posTop` and `posLeft` properties use the same length units as their counterpart CSS properties, `top` and `left`.

The following example moves the `H1` element 1 inch every time the user clicks the element.

```html
<script>
function center(oNode)
    oNode.style[this].setExpression("left","getCenter(this)";
} 
function getCenter(oNode)
    var oParent=oNode.parentElement;
    return (oParent.offsetWidth/2 - oNode.offsetWidth/2);
</script>
<div id="oDiv" onclick="center(this)"
    style="position:absolute;">Click Here To Center</div>
</script>
```
When moving elements to specific locations in a document, it is sometimes necessary to account for the different properties of the box element. Height and width values include border and padding measurements. Moving one element to the visible corner of another element is relatively easy using the offset properties and techniques described in the preceding example. However, when positioning an element at a specific point in the content of another positioned element, you must include the size of the padding and border. You can use either the client properties or the padding and border properties to establish a location within the content of an element. You can also use a `TextRectangle` to establish a location.

The following example shows three different ways to position an element within the content of another element. First, CSS pixel, border, and padding properties are used to move an element to the content within a positioned element. Next, DHTML offset and client properties are used to move an element within the content of a positioned element without accounting for the padding. And finally, a `TextRectangle` is used to establish the position of the element.

```script
function moveMe(){
    // Move the object by one inch.
    oHeader.style.posLeft+=1;
    oHeader.style.posTop+=1;
}
</script>
<h1 id="oHeader"
    style="position:absolute; top:1in; left:1in;"
    onclick="moveMe()"
    >
    Header
</h1>

function fnMove1(){
    // Method 1: Use only CSS properties
    var iTop1=oDiv.style.pixelTop +
             parseInt(oDiv.style.borderTopWidth) +
             parseInt(oDiv.style.paddingTop);
    var iLeft1=oDiv.style.pixelLeft +
              parseInt(oDiv.style.borderLeftWidth) +
              parseInt(oDiv.style.paddingLeft);
    oMarker.style.top=iTop1;
    oMarker.style.left=iLeft1;
}

function fnMove2(){
    // Method 2: Use DHTML properties.
    var iTop2=oDiv.offsetTop + oDiv.clientHeight;
    var iLeft2=oDiv.offsetLeft + oDiv.clientWidth;
    oMarker.style.top=iTop2;
    oMarker.style.left=iLeft2;
}

function fnMove3(){
    // Method 3: Use DHTML, CSS, and a TextRectangle.
    var aRects=oDiv.getClientRects();
    var oRect=aRects[0];
    var oBnd=oDiv.getBoundingClientRect();
    oMarker.style.top=oBnd.top +
                      parseInt(oDiv.style.paddingTop) +
                      parseInt(oDiv.style.borderTop);
    oMarker.style.left=oBnd.left +
                       parseInt(oDiv.style.paddingLeft) +
```
Advanced Techniques

You can use DHTML expressions and the `runtimeStyle` and `currentStyle` objects to create more advanced measurement and location formulas. Expressions use formulas to return a possible value for any DHTML or CSS read/write property. To query the current value of a property, use the `currentStyle` object instead of the `style` object. To temporarily set a property value, use the `runtimeStyle` object instead of the `style` object. When a property value is cleared from the `runtimeStyle` object, the value reverts to the original property value set on the `style` object.

Expressions, DHTML, and CSS provide a robust model for delivering information to your audience. The following example uses these technologies to animate a list when the document loads. The `marginLeft` attribute includes an expression that returns a value based on a variable.

Once the document loads, the `setInterval` method is invoked to increment the variable until it reaches a target value. The amount is determined by the `marginLeft` property of a particular list item's sibling, available from the `previousSibling` property. When the variable increments, the `recalc` method is invoked with the `true` parameter, because the `marginLeft` attribute is not implicitly dependent on the variable. Adding the `marginLeft` value of the sibling list item causes the entire list to cascade.

```html
<style>
li { margin: expression(fnTabIt(this));}
</style>
<script>
window.onload=fnInit;
// Starting marginLeft increment.
var iDML=0;
// Target marginLeft increment.
var iDMLTarg=30;
```

```javascript
parseInt(oDiv.style.borderLeft);
} 
</script>

<div id="oDiv"
    style="position:absolute; top:150px; left:50px; border:10px outset;
    padding:10px; width:250px; height:250px; background-color:#EFEFEF;">
Move marker here.
</div>

<span id="oMarker"
    style="top:200px; left:200px; position:absolute;
    border: 2px outset; background-color:#CFCFCF;">
Marker
</span>

<input type="button" value="CSS" onclick="fnMove1()">
<input type="button" value="DHTML" onclick="fnMove2()">
<input type="button" value="TextRectangle" onclick="fnMove3()"
Click the following Show Me button for an interactive demonstration of the measurement and location properties. When you select a pair of properties, the dimensions and distances are displayed using indicators created with the same techniques discussed in this overview.

Click to view sample.²⁰

Related topics

CSS Overviews and Tutorials²¹
About the DHTML Object Model³
About the W3C Document Object Model⁵²
About Dynamic Properties⁵³
position

- Edit
- Watch

Table of contents

1. Summary
2. Syntax
3. Values
4. Examples
   4.1. Relative positioning
   4.2. Absolute positioning
   4.3. Fixed positioning
5. Notes
6. Browser compatibility
   6.1. Gecko notes
   6.2. Internet Explorer notes
7. Specifications
8. See also

- Tags
- Files

Summary

The position CSS property chooses alternative rules for positioning elements, designed to be useful for scripted animation effects.
A **positioned element** is an element whose *computed* position property is *relative*, *absolute*, or *fixed*.

An **absolutely positioned element** is an element whose *computed* position property is *absolute* or *fixed*.

The `top`, `right`, `bottom`, and `left` properties specify the position of positioned elements.

**Syntax**

```
position: static | relative | absolute | fixed | inherit
```

**Values**

**static**  
Normal behavior. The `top`, `right`, `bottom`, and `left` properties do not apply.

**relative**  
Lay out all elements as though the element were not positioned, and then adjust the element’s position, without changing layout (and thus leaving a gap for the element where it would have been had it not been positioned). The effect of `position: relative` on `table-*`-group, `table-row`, `table-column`, `table-cell`, and `table-caption` elements is undefined.

**absolute**  
Do not leave space for the element. Instead, position it at a specified position relative to its closest positioned ancestor or to the containing block. Absolutely positioned boxes can have margins, they do not collapse with any other margins.

**fixed**  
Do not leave space for the element. Instead, position it at a specified position relative to the screen’s viewport and doesn’t move when scrolled. When printing, position it at that fixed position on *every page*.

**Examples**

**Relative positioning**

To position an element relatively 20px from the top and left of its normal position, the following CSS is used.

```
#two { position: relative; top: 20px; left: 20px; }
```

Note how the other elements are displayed as if ”Two” were in its normal position and taking up space.

```
One

Two

Three

Four
```

**Absolute positioning**
Elements that are positioned relatively are still considered to be in the normal flow of elements in the document. In contrast, an element that is positioned absolutely is taken out of the flow and thus takes up no space when placing other elements. The absolutely positioned element is positioned relative to nearest positioned ancestor. If a positioned ancestor doesn’t exist, the initial container is used.

In the example below, the blue ancestor div is positioned relative (so it becomes the nearest positioned ancestor) and box Two is positioned absolutely:

```css
#ancestor { position: relative; background: #ddf; width: 500px; }
#two { position: absolute; top: 20px; left: 20px; }
```

If #ancestor had not been positioned relative, box Two would have appeared relative to the upper left corner of the page.

**Fixed positioning**

Fixed positioning is similar to absolute positioning, with the exception that the element's containing block is the viewport. This is often used to create a floating element that stays in the same position even after scrolling the page. In the example below the "One" box is fixed 80px from the top of the page and 20px from the left:

```css
#one { position: fixed; top: 80px; left: 20px }
```

When viewing the top of the page, the position box appears in the upper left, and after scrolling, it remains in the same place relative to the viewport:

**Notes**

For relatively positioned elements, the `top` or `bottom` property specifies the vertical offset from the normal position and the `left` or `right` property specifies the horizontal offset.

For absolutely positioned elements, the `top`, `right`, `bottom`, and `left` properties specify offsets from the edge of the element’s containing block (what the element is positioned relative to). The margin of the element is then positioned inside these offsets.

Most of the time, absolutely positioned elements have `auto` values of `height` and `width` computed to fit the contents of the element. However, non-replaced absolutely positioned elements can be made to fill the
available space by specifying (as other than auto) both top and bottom and leaving height unspecified (that is, auto). Likewise for left, right, and width.

Except for the case just described of absolutely positioned elements filling the available space:

- If both top and bottom are specified (technically, not auto), top wins.
- If both left and right are specified, left wins when direction is ltr (English, horizontal Japanese, etc.) and right wins when direction is rtl (Arabic, Hebrew, etc.).

### Browser compatibility

<table>
<thead>
<tr>
<th>Feature</th>
<th>Chrome</th>
<th>Firefox (Gecko)</th>
<th>Internet Explorer</th>
<th>Opera</th>
<th>Safari (WebKit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic support</td>
<td>1.0</td>
<td>1.0 (1.0)</td>
<td>4.0</td>
<td>4.0</td>
<td>1.0 (85)</td>
</tr>
<tr>
<td>fixed value</td>
<td>1.0</td>
<td>1.0 (1.0)</td>
<td>7.0</td>
<td>4.0</td>
<td>1.0 (85)</td>
</tr>
</tbody>
</table>

### Gecko notes

In Gecko 10.0 (Firefox 10.0 / Thunderbird 10.0), handling of absolutely positioned elements inside tables was fixed. Previously, if you had a positioned <table> element inside a positioned block element, such as a <div>, a position: absolute; styled element inside the table would be positioned relative to the outer <div> rather than the table, which is the nearest positioned ancestor. This was a bug, and has been fixed. See this blog post for a more verbose explanation of the details.

### Internet Explorer notes

In Internet Explorer, fixed positioning doesn't work if the document is in quirks mode.

### Specifications

- [CSS 2.1 Visual formatting #position](https://developer.mozilla.org/en/CSS/position)

### See also

display, float, top, right, bottom, left

### Tags (2)

- [Edit tags](https://developer.mozilla.org/en/CSS/position)
  - CSS Reference
  - css

### Attachments (4)

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Attached by</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute-positioning.png</td>
<td>2.11 kB</td>
<td>19:43, 28 Jan 2011</td>
<td>mmclark</td>
</tr>
<tr>
<td>Absolute Positioning Example</td>
<td></td>
<td></td>
<td>Actions</td>
</tr>
<tr>
<td>fixed-1.png</td>
<td>16.89 kB</td>
<td>20:13, 28 Jan 2011</td>
<td>mmclark</td>
</tr>
<tr>
<td>Fixed position example (before scrolling)</td>
<td></td>
<td></td>
<td>Actions</td>
</tr>
</tbody>
</table>

Fixed position example (after scrolling) [relative-positioning.png]

Relative Positioning Example

Images 4

Absolute Positioning Example [absolute-positioning.png]

Relative Positioning Example [relative-positioning.png]

Attach file

Page last modified 17:36, 29 Apr 2012 by Teoli

© 2012 Mozilla Developer Network

Content is available under these licenses • About MDN • Privacy Policy • Help

• Sign in

What’s this?

MDN has switched to BrowserID, a safe and simple way to sign in with just your e-mail address. Learn more about why Mozilla is using BrowserID.

Returning members: sign in with BrowserID and you'll be connected to your MDN profile (all your information is still here).

New members: sign in with BrowserID first, then you'll be able to set up your new MDN profile.

Sign in