Using Multiple JavaScript Onload Functions

By Lee Underwood

http://www.htmlgoodies.com/beyond/javascript/article.php/3724571/Using-Multiple-JavaScript-Onload-Functions.htm (Back to article)

JavaScript is one of the most popular programming languages in use on the World Wide Web. It can create interactive pages through the use of menus, forms and calendars. It can also be used to track a visitor's history on your site, play games and many other things.

When scripts are written they're used to accomplish a given task, e.g., create a rotating picture gallery, or to validate a form. This means that for each different type of usage, a separate script is necessary. Often, a script is called using an `onload` function.

Loading the Page

When a Web page is loaded, the code on the page — whether HTML, CSS, PHP, or some other type — is generally processed from the top down. As the browser comes upon a section of code, it's executed and the intended action is (hopefully) performed. Let's say you have the following script in the `head` section of a Web page:

```
function func1() {
  alert("This is the first.");
}
window.onload=func1;
```

When the page is loaded, the browser will stop at the script above and execute it. An alert window will pop up and the page will stop loading until the alert window is closed. Often, this isn't an issue, but if the script needs to access elements on the page that aren't yet loaded (i.e., below the current code), you will encounter problems. To resolve these, here are a few solutions.

Move it to the Bottom

One method is to place the script at the bottom of the Web page, just above the closing `</body>` tag. This does, however, present a few problems. First, there is a period of time that those elements which are you seeking to change with the script will be clearly seen by the visitor. Then, once the script is executed, they will "magically" change. With a fast loading page and a script that executes quickly, this might not be a problem, but if the original elements can be seen by the visitor, and are then changes, the visitor may perceive it to be some type of a hack or bug on the page, and leave the site immediately. Obviously, that's not good.

Secondly, either the script, or a link to it, will need to be included in the actual HTML `body` code. That's not unobtrusive JavaScript. It's best if you can keep everything separate. You'll find it's easier to work on any aspect of the page, and the page itself will load faster.

The `onload` Function

The `window` object in JavaScript has [an event handler] called `onload`. When this event handler is used, the entire page and all of its related files and components are loaded before the function listed in the `onload` event handler is executed, hence the term "*on load*."

As shown in the script above, it's easy to use, but what if you have more than one function you want to call using an `onload` event handler?

You would think you could just stack them like this:

```
window.onload=func1;
window.onload=func2;
window.onload=func3;
```

Unfortunately, it's not that simple. After the first `onload` event handler is executed, it will be replaced when the second `onload` event handler is executed. That, in turn, will be replaced immediately just as soon as the third `onload` event handler is executed. There are workarounds for this, though.

Let's Put Them in a Chain

One method that has been used quite a bit is the linking of multiple events. It places several function calls in a chain, using one `onload` event handler. The method would look like this:

```
<body onload="func1(); func2();">
```

Still, once again we run into the unobtrusive problem. In addition, you will need to add this to the `<body>` tag of every page that needs to use the called functions. That could be quite tedious for a large Web site, especially when changes or additions are needed.

Within Another Function

Another method is the following script:

```
function start() {
  func1();
  func2();
}
window.onload = start;
```

While this is good, there's one method that's even better.

The `addLoadEvent` Function

```
function addLoadEvent(func) {
  var oldonload = window.onload;
  if (typeof window.onload != 'function') {
    window.onload = func;
  } else {
    window.onload = function() {
      if (oldonload) {
        oldonload();
      }
      func();
    }
  }
}
addLoadEvent(nameOfSomeFunctionToRunOnPageLoad);
addLoadEvent(function() {
  /* more code to run on page load */
});
```

This function was written by Simon Willison. It has several advantages. For one, it's really unobtrusive. It can be placed in a file with your other scripts or in a separate file. (If it's in a separate file, just be sure to call it *after* the other files, so the functions will be available.)

Also, it works even if the `onload` event handler has been previously assigned. Simon explains it like this:

> "The way this works is relatively simple: if `window.onload` has not already been assigned a function, the function passed to `addLoadEvent` is simply assigned to `window.onload`. If `window.onload` has already been set, a brand new function is created which first calls the original `onload` handler, then calls the new handler afterwards."

Isn't that beautiful? So, if you already have a script that uses the `onload` event handler, you don't need to dig it out and change it, unless you want to. It also allows for extra code. Here's an example that calls two functions and adds a third, independent `addLoadEvent`:

```
function func1() {
  alert("This is the first.");
}
function func2() {
  alert("This is the second.");
}

function addLoadEvent(func) {
  var oldonload = window.onload;
  if (typeof window.onload != 'function') {
    window.onload = func;
  } else {
    window.onload = function() {
      if (oldonload) {
        oldonload();
      }
      func();
    }
  }
}
addLoadEvent(func1);
addLoadEvent(func2);
addLoadEvent(function() {
    document.body.style.backgroundColor = '#EFDF95';
})
```

After calling `func1` and `func2`, a function is created that changes the background color of the Web page. Each of these is executed in the order shown. Try it yourself and see how it works!

Conclusion

Sometimes, figuring out which approach to use in coding takes a bit of trial and error. Other times it's advantageous to search the Web and see what others have done. The `addLoadEvent` function should work just fine in this situation. We use it over at the JavaScript Source for all of our scripts requiring the `onload` event handler.

This article originally appeared on WebReference.com.