

# Markup as a Craft

By [Garrett Dimon](#)

Published on January 8, 2007

Markup is the technical foundation of front-end code. In one way or another, it influences or is influenced by design, content, accessibility, CSS, DOM scripting, and more. The quality of your markup will affect the quality of related code, and even the cost of implementing or maintaining that code. Your markup might be good now, but following the guidelines in this article will help bring it to the next level.

**“One of the most wicked defilers of beautiful markup is inline JavaScript. This makes the markup all but impossible to read, and provides lots of little dark corners for bugs to hide.”**

Carefully crafting your markup is like making a lucrative investment in the future of your code base. It's easier to create and maintain back-end code when it integrates with clean and simple markup. Your CSS can be elegant. Your DOM scripting can connect seamlessly. You'll have made a significant first step towards improved accessibility. And you might even pick up some extra search engine traffic. It's time to treat markup as a craft, and give it the love and attention it deserves.

## Simplify everything.

Before we get into the specifics, we need to do some quick housecleaning. Fewer lines of code or markup means fewer places for errors to hide, and that means that it's easier for someone else, or even you, to understand in the future. So, as with any solution, the pursuit of simplicity is an investment that is always worth making. This brings us to our first guideline.

### **Guideline Number 1: Simplify. Simplify. Simplify.**

The easiest way to achieve simplicity in your markup is to remove everything that's inappropriate or not necessary. This includes, among other things, tables for layout, font elements, inline style attributes, and inline JavaScript. There's no place in present-day markup for any of these. Progressive enhancement and CSS should make these methods all but extinct.

We'll start with the layout tables. They absolutely have to go. They are difficult to maintain, unnecessarily increase the page weight, introduce accessibility problems, and provide no semantic value. Thus we have the basis for our second guideline.

### **Guideline Number 2: Don't use tables for layout.**

We've disposed of the tables, but we also need to cleanse our markup of the style demons. These include, but aren't limited to, the `style` attribute, `font` elements, `strike` and `s` elements, and any other elements that smuggle presentational information into our otherwise pure markup such as purely decorative images or characters.

It's worth elaborating on the decorative images aspect. If we're using images that only provide decorative value and contain no content, they should be placed in the CSS, as background images. To generalize, only images that contain information such as charts, graphs, or written content need to be included in your markup. This helps eliminate unnecessary markup and separates presentation and structure.

### **Guideline Number 3: Markup should be purely semantic structure. Eliminate the elements that control presentational aspects.**

One of the most wicked defilers of beautiful markup is inline JavaScript. This makes the markup all but impossible to read, and provides lots of little dark corners for bugs to hide. If that isn't bad enough, the interweaving of JavaScript and markup creates a tangled mess of code that can negatively affect our development and maintenance costs. This brings us to guideline number four.

### **Guideline Number 4: Avoid using inline DOM scripting in markup, including event handlers.**

For the uninitiated, this may seem like a travesty, but rest assured, there are better ways to handle our DOM scripting. We can now put all of our scripts in external files and use the `getElementById` and `getElementsByTagName` functions to get references to the appropriate elements in our documents. Once we have those references, then our script can manipulate any aspect of the markup. Whether we're inserting new elements, adding event handlers, or moving elements around, it's all handled outside of the markup, and it allows us to keep things nice and separated.

So, with a sprinkle of [progressive enhancement](#) and a good working knowledge of the DOM, we can elegantly insert our client-side scripting behavior until we're blue in the face, because we'll be following best practices for separating our behavior from our structure. Going into the details of progressive enhancement is a bit outside the scope of this article, but there are plenty of articles that can help explain [the DOM and progressive enhancement](#). It is, however, worth mentioning that just because we're relocating the scripts in order to simplify our markup, it's equally important to make sure our external scripts are nice and tidy, as well. Otherwise, we're simply sweeping everything under the carpet and defeating the purpose.

At this point, our markup is relatively clean. We've removed the cobwebs and thrown away the trashy, antique tables and dated decorations, and we can move on to the more serious remodeling.

### **Layer carefully. Markup should stand on its own.**

While we're on the topic of inappropriately placing JavaScript in our markup, it makes sense to think about the flip side as well. Many poor souls have had maintenance nightmares in situations where markup was secretly buried inside of the external DOM scripts. This is generally a dangerous separation of behavior and structure, and we need to be mindful of the potential pitfalls.

The same is true of content. If we hide content in our script files, not only could we be creating maintenance nightmares, we could also be hindering the accessibility of that content, if we're not careful. Now, none of this is to say that we should never place markup or content in our external scripts, but we should do it with the utmost caution, and only after carefully considering the consequences. Now we have two more guidelines.

### **Guideline Number 5: Avoid putting markup in scripts.**

### **Guideline Number 6: Avoid putting content in scripts.**

Putting content or structure in our external scripts is like having skeletons in the closet. We're almost guaranteed that they'll come back to haunt us someday. If we think twice before creating our scripts, we should be in good shape.

### **Carefully craft your hooks.**

We know we shouldn't intertwine our presentational and behavioral aspects with our markup, and we know that we should be putting them in separate files. However, just because they're not inline doesn't

mean they can't trick us into muddying up our code. Since markup serves as a sort of hub for our other technologies, we need to think about how we create those hooks between DOM scripting and the markup, and CSS and the markup.

We could easily start putting IDs and classes all throughout our code so that we have the hooks to manipulate and change anything directly, but this could lead to a plethora of unnecessary bits of code that almost defeat the purpose of separating our different layers. It's important to give careful consideration to when and where we use classes and IDs.

CSS kindly gives us incredible flexibility when creating selectors. Through the use of some of the more advanced [CSS selectors](#) we can start to create selectors that actually need very few classes or IDs to get the job done.

For instance, maybe we could use `#primary div h3` instead of adding an ID to that `h3` element and referencing it as `#testimonials`. Of course, this might make the CSS a bit more complex, so we have to pay attention to the tradeoffs. However, since markup is our foundation, it's best to move the complexity out of the markup to the more appropriate areas.

### **Guideline Number 7: Try to avoid using unnecessary classes and IDs.**

So, now that we're keeping our hooks to a minimum, it's time for our next step in quality over quantity: naming. We want to avoid names that imply presentational aspects. Otherwise, if we name something `right-col`, it's entirely possible that the CSS would change and our "right-col" would end up actually being displayed on the left side of our page. That could lead to some confusion in the future, so it's best that we avoid these types of presentational naming schemes.

### **Guideline Number 8: Class and ID attributes should be named according to their content or purpose, not their presentation.**

Alright, we're starting to really make progress, but we need to consider one more aspect of defining our class and ID names. We should always take the time to give them good meaningful names. If we use names like `x1` and `a3`, we're not going to be happy with ourselves later. Instead, we should take some time and come up with names that tell us something like `summary` or `code`. Now we can put another guideline on the books.

### **Guideline Number 9: Class and ID attributes need meaningful names.**

We're making pretty good progress, and now we've got clean markup with some useful hooks carefully inserted for good separation of our different layers.

## **Know the code and your options.**

Since we just covered IDs and classes, it only makes sense to touch on the wonderful world of [microformats](#). With microformats, we don't even have to think of naming conventions for some common blocks of content such as events, addresses, reviews, and more. If we take advantage of microformats, we already know which classes to use for marking up common blocks of content. Best of all, they're an open standard that helps everyone, so we have another guideline.

### **Guideline Number 10: Use microformats when appropriate.**

While microformats are helpful, there won't always be a microformat for every piece of content we have, so it's important that we're familiar with [all of the elements at our disposal](#). This way, we can be sure we're always using the best element for the job. It's a quick one, but guideline number eleven is one of the keys to creating quality markup.

### **Guideline Number 11: Know all of the HTML elements.**

It's not enough to know which elements to use, we also need to be aware of which ones not to use, and our next guideline addresses that. There are [certain elements that have been deprecated](#) in favor of newer alternatives. For example, the `font` element has been deprecated in favor of controlling typography through CSS. The `strike` element, which implies a presentational strikethrough with its name, has been deprecated in favor of the `del` and `ins` elements, which offer more appropriate semantic meaning. So, we have another guideline.

### **Guideline Number 12: Don't use deprecated or inappropriate elements.**

### **Guideline Number 13: Communicate meaning through the use of semantic elements.**

Unfortunately, knowing and understanding the details of which elements to use in which circumstances doesn't mean that all of the current web browsers support all elements. For instance, our good friend IE6 doesn't support the `abbr` or `q` element. Thus, we might not always be able to use the best element for the job. So, it's a very subtle, but unfortunately important guideline that we have next.

### **Guideline Number 14: Be aware of and understand user agents and their nuances.**

We've talked fairly extensively about elements, but that's really only half the battle. Every element supports [a wide variety of attributes](#) that enable us to provide additional meaning to all that code we're writing. Some commonly overlooked or misunderstood attributes include `rel`, `rev`, `alt`, and `title`. Accurate use of these attributes can provide valuable information to our visitors.

### **Guideline Number 15: Know and understand all of the HTML attributes.**

We've covered most of the practices we can use to create better markup, so, to finish up, we're going to go over guidelines that can help us fine-tune our markup by addressing specific needs.

## **Addressing the details.**

As with any skill or craft, the basics might get us on our feet, but it's the attention to detail that really helps us get to the next level. The first detail we'll discuss is the order of the source code. Even though CSS enables us to control the visual order of the code in the browser, the actual order of the code is important, as well, for improved accessibility. While it should be simple to maintain a good source order thanks to CSS, it's not quite as simple as it could be. However, it's still valuable to consider source order when writing your markup. Good source order helps contribute to accessibility, as well as providing a logical hierarchy to the page. Now we have our first subtle guideline.

### **Guideline Number 16: Keep content in a logical order in the source.**

Of course, we all want our markup to integrate seamlessly with our content, and one of the most powerful places to do that is through our anchor text. Whenever we wrap text with anchor tags, there's incredible value in carefully crafting that anchor text and making sure that the most appropriate text is used. Good anchor text contributes to accessibility, usability, and even search engine optimization. So, by simply spending some time to make sure that the text of our links accurately describes the content of the link target, we'll be addressing our next guideline.

### **Guideline Number 17: Create meaningful links.**

The `alt` attribute is our next focus for refinement. As one of the most misunderstood attributes, the subtlety around [creating good alternate text](#) is fairly complex. Suffice it to say that simply including alternate text with images does not guarantee that it is appropriate. We have to consider not only the content of the image, but also the context of the image. So, our next guideline is about understanding how to [create appropriate alternate text](#) for your images.

### **Guideline Number 18: Create useful alt attributes.**

Along with alternate text, the use of headings is another area of markup that has an incredible amount of complexity and discussion behind it. As [Andy Budd pointed out](#), the headings part of the HTML spec says, “A heading element briefly describes the topic of the section it introduces. Heading information may be used by user agents, for example, to construct a table of contents for a document automatically.”

He goes on to explain the implication that headings should be used to describe the structure of a page rather than simply being used for stylistic information. If we remove all of the content from a page and left the headlines, would it create a logical outline for our content on the page? If not, we’re probably not making the most of our headings.

### **Guideline Number 19: Make the most of headings by using them to create an outline for the page’s content.**

Forms are a great place to make huge gains from cleaner and more semantic markup. By using a `label` element, we’re giving the document additional meaning while making the forms more accessible. Additionally, by using one input field instead of three for things such as dates and phone numbers, we’ll see several inherent benefits. We’ll have less code, added flexibility to support international formats, and increased accessibility when used with `label` elements.

Let’s take a second to address the accessibility benefit. Imagine if we have one label for a phone number, but three separate fields. Now, that label can only reference one of those fields, so the other two are orphaned and significantly less accessible. However, by using one field, we’re creating a much simpler one-to-one relationship between the label and the field—improved accessibility and less code in one fell swoop. Since forms are one of the more complex aspects of markup, they get a guideline all to themselves.

### **Guideline Number 20: Use good form when marking up forms.**

Right after forms, tables are one of the easiest, but most overlooked areas to add semantic meaning. Now, I know we talked about how evil tables can be when used for layout, but when used to format data, they’re incredibly powerful and meaningful. We have to make the most of all the things we can do with tables, and that requires us to be familiar with [the spec for tables in HTML](#). We have elements such as `tbody`, `thead`, `tfoot`, and others. We also have myriad attributes to help us add even more semantic meaning to our data. There’s a whole world of options, and it’s free for the taking by simply reading up on the spec. Like forms, tables are fancy enough to get their own guideline as well.

### **Guideline Number 21: Create semantically rich data tables.**

That’s it. We’re now well on our way to creating beautifully seamless markup that can communicate semantic meaning, be more accessible, be easier to read and maintain, and be just more enjoyable to write.

## **Summary**

With front-end technology, it all comes back to the markup. It’s not particularly difficult to write great

markup, but it does take attention to detail and an appreciation for some of the more subtle aspects. We can all be creating better markup with only a little reading and learning. It boils down to simply knowing what to do and when to do it.

## **Related Resources**

Here are two additional resources to help create an appreciation for and understanding of markup and HTML documents.

- [SimpleQuiz](#)
- [Document-level Details](#)

More Information Visit: <http://www.sitepoint.com/>