



Dive Into Accessibility

Table of Contents

Dive Into Accessibility	1
Introduction	2
Day 1: Jackie	3
Day 2: Michael	4
Day 3: Bill	5
Day 4: Lillian	6
Day 5: Marcus	7
Day 6: Choosing a DOCTYPE	8
Who benefits?.....	8
How to do it.....	8
Further reading.....	9
Day 7: Identifying your language	10
Who benefits?.....	10
How to do it.....	10
Further reading.....	11
Day 8: Constructing meaningful page titles	12
Who benefits?.....	12
How to do it.....	12
Further reading.....	13
Day 9: Providing additional navigation aids	14
Who benefits?.....	14
How to do it.....	14
Further reading.....	15
Day 10: Presenting your main content first	16
Who benefits?.....	16
How to do it.....	16
Further reading.....	17
Day 11: Skipping over navigation links	18
Who benefits?.....	18
How to do it.....	18
Day 12: Using color safely	20
Who benefits?.....	20
How to do it.....	20
Further reading.....	21

Table of Contents

Day 13: Using real links.....	22
Who benefits?.....	22
How to do it.....	22
Further reading.....	23
Postscript.....	23
Day 14: Adding titles to links.....	24
Who benefits?.....	24
How to do it.....	24
Further reading.....	25
Day 15: Defining keyboard shortcuts.....	26
Who benefits?.....	26
How to do it: home page link.....	26
How to do it: skip navigation link.....	27
How to do it: feedback link.....	27
Further reading.....	27
Day 16: Not opening new windows.....	28
Who benefits?.....	28
How to do it.....	28
Further reading.....	29
Day 17: Defining acronyms.....	30
Who benefits?.....	30
How to do it.....	30
How to do it: cascading style sheets.....	30
Further reading.....	31
Postscript.....	31
Day 18: Giving your calendar a real caption.....	32
Who benefits?.....	32
How to do it.....	32
Further reading.....	33
Day 19: Using real table headers.....	34
Who benefits?.....	34
How to do it.....	34
Very important note about layout tables.....	36
Further reading.....	36
Day 20: Providing a summary for tables.....	37
Who benefits?.....	37
How to do it: calendar.....	37
How to do it: layout tables.....	38
Day 21: Ignoring spacer images.....	39
Who benefits?.....	39
How to do it.....	39
Things not to do.....	40

Table of Contents

Day 21: Ignoring spacer images	
Further reading.....	40
Day 22: Using real lists (or faking them properly).....	41
Who benefits?.....	41
How to do it.....	41
How to do it: advanced.....	42
Postscript: un-bulleted lists.....	42
Further reading.....	43
Day 23: Providing text equivalents for images.....	44
Who benefits?.....	44
How to do it.....	44
Examples of bad alt text.....	45
Examples of good alt text.....	45
Further reading.....	45
Day 24: Providing text equivalents for image maps.....	47
Who benefits?.....	47
How to do it.....	48
Things not to do.....	49
Further reading.....	49
Day 25: Using real horizontal rules (or faking them properly).....	50
Who benefits?.....	50
How to do it.....	50
How to do it: advanced.....	50
Further reading.....	51
Day 26: Using relative font sizes.....	52
Who benefits?.....	52
How to do it: Radio.....	53
How to do it: Movable Type.....	54
How to do it: detailed explanation.....	55
Further reading.....	57
Day 27: Using real headers.....	58
Who benefits?.....	58
How to do it: Movable Type.....	58
How to do it: Radio.....	59
Further reading.....	60
Day 28: Labeling form elements.....	61
Who benefits?.....	61
How to do it: Movable Type.....	61
How to do it: Greymatter.....	62
Further reading.....	63

Table of Contents

Day 29: Making everything searchable.....	64
Who benefits?.....	64
How to do it.....	64
Further reading.....	65
Day 30: Creating an accessibility statement.....	66
Who benefits?.....	66
How to do it.....	66
Further reading.....	67
Conclusion.....	68
Further reading: books that I recommend.....	68
Accessibility statement.....	69
Access keys.....	69
Standards compliance.....	69
Navigation aids.....	69
Links.....	70
Images.....	70
Visual design.....	70
Accessibility references.....	70
Accessibility software.....	70
Accessibility services.....	70
Related resources.....	71
Accessibility books I recommend.....	71
Terms of use.....	72
GNU Free Documentation License.....	73
0. PREAMBLE.....	73
1. APPLICABILITY AND DEFINITIONS.....	73
2. VERBATIM COPYING.....	74
3. COPYING IN QUANTITY.....	74
4. MODIFICATIONS.....	75
5. COMBINING DOCUMENTS.....	76
6. COLLECTIONS OF DOCUMENTS.....	76
7. AGGREGATION WITH INDEPENDENT WORKS.....	76
8. TRANSLATION.....	77
9. TERMINATION.....	77
10. FUTURE REVISIONS OF THIS LICENSE.....	77
How to use this License for your documents.....	77
Translations.....	79
Chinese.....	79
Estonian.....	79
Finnish.....	79
French.....	79
German.....	79
Italian.....	79
Norwegian.....	79

Table of Contents

Translations

Polish.....	79
Romanian.....	80
Spanish.....	80
Swedish.....	80

Dive Into Accessibility

30 days to a more accessible web site

Copyright © 2002, Mark Pilgrim

This book lives at diveintoaccessibility.org. If you're reading it somewhere else, you may not have the latest version.

This book is free. Permission is granted to copy, distribute, and/or modify this book under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the last chapter of the book, entitled "GNU Free Documentation License".

Introduction

This book is entitled "Dive Into Accessibility: 30 days to a more accessible web site", and it will answer two questions. The first question is "Why should I make my web site more accessible?" If you do not have a web site, this book is not for you. The second question is "How can I make my web site more accessible?" If you are not convinced by the first answer, you will not be interested in the second.

To answer the first question, I will present character sketches of five people: [Jackie](#), [Michael](#), [Bill](#), [Lillian](#), and [Marcus](#). These people have several things in common:

1. They all have a combination of physical, mental, and technological disabilities which make it more difficult to use the Internet.
2. Although fictitious, they all represent real people with disabilities, and they use the Internet in ways that real people with disabilities use the Internet.
3. They all have difficulty reading your web site.

To answer the second question, I will present 25 tips that you can immediately apply to your own web site to make it more accessible. Although these concepts apply to all web sites, I will be focusing on implementation using popular weblogging tools. If you use some other publishing tool or template system, you will need to determine how to implement the tips in your tool of choice.

Each tip will focus on a single concept, explain the reasoning behind it, and show who will benefit once you implement it. This is why the character sketches come first, because they change the tone of the first question from "Why should I bother?" to "Who benefits?" Answer: "Marcus benefits." "How does Marcus benefit?" "Well, let's look at that..." And so forth.

Don't panic if you are not an HTML expert. Don't panic if the only web site you have is a personal weblog, you picked your template out of a list on your first day of blogging, and you've never touched it since. I am not here to tell you that you need to radically redesign your web site from scratch, rip out all your nested tables, and convert to XHTML and CSS. This is about taking what you have and making it better in small but important ways. Jackie, Michael, Bill, Lillian, and Marcus will thank you for your attention.

Day 1: Jackie

Jacquelin, who goes by Jackie, lives with her mother in Park Ridge, Illinois, outside Chicago. She is 19 years old, a senior in high school, and she is an A student. The reason she is an A student is partly because she has private tutors to help her, but mostly because she studies diligently and reads voraciously. She has all of her textbooks on audiocassette, which she listens to on a special tape player that can intelligibly play the tapes at 3 times their normal speed with minimal distortion. She has been blind for 8 years.

Since she was not born blind, Jackie understands sighted concepts like colors, and she still talks about colors with her mother in terms of things that were in her life "before". The one thing she does not talk about is the car accident that killed her father and left her blind; it is only referred to indirectly by prepositions: "before" and "after". "This is green like the walls of the living room before." "It's sort of like that pink sweater you wore before, only lighter." And so forth.

Jackie is two years behind her classmates in school, due to her difficulty adjusting to life immediately after. This fall, she will be attending the University of Chicago and majoring in comparative literature. She is excited about going to college, partly because she hopes to make new friends, and partly because she will be able to do so much more of her classwork online: reading class schedules, submitting papers, and instant messaging with her professors and classmates. She has few friends in her high school; she spends most of her time with her mother, and the rest online.

She spends over \$300 a month on audio books, music, and the usual array of geek gadgets. Most of her audio books are still on tape, although she is finding more and more interesting reading material that she can download and have her eBook reader read to her. Music on CD, and gadgets, gadgets, gadgets, all from online retailers.

It's not that online shopping is necessarily easy, but it's light years ahead of taking the train to the local mall and trying to get a salesperson's attention. Also, shopping online is something she can do without her guide dog, Arthur. She dislikes Arthur; he's not as good as her previous guide dogs, Lancelot and Guinevere, both of whom are now retired and live with her and her mother. She tells them apart by their collars; Arthur has a smooth collar, Lancelot's is spiked, Guinevere's is grooved.

Like the majority of blind people, Jackie knows very little Braille. She has a Braille label maker to mark her CDs, but she can not read Braille books, because they are written in grade 2 Braille, which she has never learned. When she shops and plays online, she uses the latest version of [JAWS](#), a screen reader that integrates with Internet Explorer on Windows. JAWS uses an advanced text-to-speech synthesizer to read web sites aloud. It also has a mind-numbing array of esoteric keyboard shortcuts for navigating through web sites, all of which Jackie has memorized. She can read well-designed web sites even more quickly than she can read her audio textbooks.

Day 2: Michael

Michael is 27 years old, retired, and lives with his girlfriend Christine in the Great Neck neighborhood of Long Island. When Michael was 22, he started a small company that specialized in laying ultra-high-speed fiber cable. Two years later, he cashed in at the right time and sold out to the same worldwide telecommunications conglomerate which, at this very moment, is on the phone explaining to Michael why he can not get anything better than 56K dialup access in his new \$4000-a-month apartment. Money, it seems, can buy everything but competence.

Michael sees the world differently than you do. This is not a business cliché or a philosophical statement; he really sees the world differently than you do. He has protanopia, an inability to distinguish red from green. (Reds, in particular, appear very dark, almost black.) All of his clothes are discretely labeled with letters, R for red, DG for dark green, and so forth. His girlfriend has compiled a compatibility matrix that specifies which clothes he is allowed to wear together. He follows these instructions to the letter, so to speak, although he does not understand why they matter.

While the customer service representative is pleasantly telling Michael that he is forever screwed, Michael is doing his usual online rounds. Now that he is retired, he spends most of his time collecting old arcade game boards, fixing them up, putting them in cabinets, detailing them, and reselling them on online auction sites. Actually, his girlfriend does most of the detailing, but Michael does everything else. He also owns a ferret named Ralph, who, despite the name, is female. Christine complained that male ferrets smell too much, so they compromised and did it her way. Ralph spends most of her time sleeping on the upper-left corner of the couch, but comes out to sit on the windowsill at morning rush hour to watch the traffic. Actually, Michael does this too. Christine does not; she sleeps in. It's a man-and-his-ferret kind of moment.

As you might expect, Michael's hobby requires quite a bit of online interaction with past and future customers, suppliers, arcade owners, and anyone else who might have a lead on an obscure arcade machine part. He curses his 56-but-really-28-today-K dialup access, and fires up his web browser. Some days he uses the text-only web browser [Links](#), which renders complex multi-column layouts as well as a graphical browser, but only in text — no images or fonts. Other days he uses [Opera](#), because it allows him to load pages in the background, and to easily turn images on and off. Mostly he leaves them off, to conserve bandwidth. Today is a Links day.

Traffic lights are red-yellow-green, from the top down. Traffic lights that are positioned sideways are tricky, and therefore dangerous. Michael, like the 8% of American men who are colorblind, has learned to quickly scan the surrounding area and follow the pole from the ground up to the lights. The light furthest along the pole from the ground is red, nearest the ground is green. Yellow is always in the middle.

Day 3: Bill

Bill lives in a small apartment in San Francisco, 5 miles away from his sister, his only living relative. He is 62 years old. The apartment is always impeccable, like a military barracks just before inspection. This is only natural, since Bill has been in the military since he was 18. He started as a soldier, moved up the ranks, was decorated three times in Vietnam, and remained in active duty until a knee injury forced him into a desk job 10 years ago. Then last year, he suffered a stroke and was finally forced to retire. His right arm does not move at all, and his left hand shakes for reasons that no one has been able to explain. 500 people came to his retirement party. He knew all of them personally.

Bill uses an old laptop that runs [Red Hat Linux](#). He browses the web with [Mozilla](#), and reads his mail with [Evolution](#), although he does not know this. The computer (and its applications) were a retirement gift from his sister, who is really good at this computer stuff. The laptop is one of those old laptops that is perpetually stuck in 1024 by 768 resolution, despite its 13 inch screen. Bill can see just fine, thank you very much, although his eyes do occasionally get confused and lose track of what they're reading. This started with the stroke, and his sister thinks it's getting worse, but she is obviously mistaken.

His one big computer-related indulgence was a keyboard extension that gives him a second set of arrow and PageUp/PageDown keys, so he can more easily reach them with his good hand. \$29.99, with a \$5 mail-in rebate, which he mailed in.

Before his retirement, Bill had never been online, but now he is finding more and more reasons to stay there. He spends four hours a day reading web sites and talking to old and new friends. Encouraged by his veteran buddies, and with technical help from his sister, he has started an email newsletter on veteran's rights. It already has 200 members.

Bill can type a surprising 10 words a minute (next year it will be 15) by steadying his left hand against the base of the keyboard and only moving his fingers. However, he can barely use a mouse at all, and mostly navigates with the arrow keys, the tab key, and a dizzying array of keyboard shortcuts that his sister has shown him. His sister also taught him how to activate a special program so he can move the cursor with his arrow keys, but this is so painfully slow and fraught with error that he rarely bothers.

Although he doesn't know it yet, next year the newsletter will turn into a weblog, and the year after that, the weblog will turn into a political action committee and become a major political force. Politics, after all, is all about who you know, and Bill knows everyone.

Day 4: Lillian

Lillian is an American immigrant, fresh off the boat, as they say, from Hong Kong. The boat was really a plane, and she came here 30 years ago, but they still say that. She is 54 years old, a widow, and lives in Kansas City with her daughter, her son-in-law, and her 2-year-old grandson. Together, they take up most of her time. Her daughter and son-in-law are fluently bilingual, but Lillian still struggles with English and prefers her native Cantonese. When she's not playing with her grandson, she tries to improve her English skills by reading the newspaper. She spreads it out on the kitchen table, turns on the 100 watt overhead light, and reads it with a magnifying glass.

Lillian works as an office assistant in a worldwide telecommunications conglomerate. By sheer coincidence, this is the same worldwide telecommunications conglomerate that is too incompetent to offer high-speed Internet access to wealthy 20-something retirees in Long Island, although Lillian does not know this. Their IT department has just completed a worldwide migration to Windows XP and Internet Explorer 6, and is expected to be grumpier than usual for the next 9 months. They have also implemented new Internet security policies: no Java, no Javascript, no Flash, no ActiveX controls, except on IT-approved sites, of which there are none. Technically, this means that all sites are in the Restricted zone in Internet Explorer's Security tab, that Restricted sites have all scripting turned off, and that you need administrative access to add a site to your Trusted zone.

Needless to say, Lillian does not have administrative access.

She does, however, have a 19-inch monitor, against the strenuous objections of Matt in IT, who reminds Lillian at every opportunity that he had to carry it up three flights of stairs because the elevator was out that day. He says it in a nice way, though. Lillian likes Matt; he's the nicest of the bunch, and he even once set her text size to "Larger" in Internet Explorer, so now her daughter's weblog is actually large enough to be readable. She reads it every day. But when she asked Matt why she couldn't make CNN.com any larger, Matt launched into one of his geek tirades with lots of big technical words, got very frustrated, and finally said there was nothing he could do.

Lillian wishes she could read more web sites, but if Matt can't fix it, no one can.

Day 5: Marcus

Marcus, son of a drug addict, has been blind since birth. He was born three months premature, was on a respirator for six weeks, and was given a 10% chance of survival. He is 31 years old.

Marcus works at an AT+T Relay Center, where he relays calls between deaf and hearing people. Here's how it works:

1. A deaf or hearing impaired person, call her Melissa, calls into the center using her TDD or computer.
2. Melissa types in the name and number of the person she wants to call, call him Todd.
3. Marcus calls Todd and announces that Todd has a call from Melissa via the Relay Center.
4. Whatever Melissa types, Marcus reads on his [ALVA refreshable Braille display](#), which takes Melissa's words and converts them into Braille in real time.
5. Whatever Todd says, Marcus types back to Melissa at an astonishing 110 words per minute.

Neither Melissa nor Todd knows or cares that Marcus is blind. The only indication is the muffled noise of the ALVA's pins popping up and down, like the clackety-clack of a manual typewriter in the distance.

There is no link whatsoever between Marcus's eyes and his brain. Despite this, he still complains of headaches whenever the sun streams in through the window near his cubicle, and he closes the blinds every afternoon.

Marcus also uses an ALVA at home, where he runs the text-only browser [Lynx](#) in a full-screen DOS window. He reads the web at home in much the same way that he reads his calls at work: in Braille, one line at a time. He hates screen reader software, and he wouldn't hear it even if he had it, since he always has talk radio on at top volume from the minute he gets home until the minute he goes to sleep.

He also talks on the phone while he listens to the radio and surfs the web. He talks to 100 people a week. He has no idea that this is unusual. If you played the "six degrees of separation" game, finding links between you and anyone else in the world, Marcus would be one of your six degrees. He is one of those people that generates buzz, that marketers think they can tap into but never can. He is the reason your phone company no longer offers unlimited local calling plans. He loves telling his friends about cool sites that he's seen. He intentionally uses the word "seen" because he knows it makes them feel uncomfortable.

Day 6: Choosing a DOCTYPE

You start your sentences with a capital letter; start your HTML with a DOCTYPE. It's just basic grammar.

Who benefits?

You benefit. Many of the tips in the rest of this series will require you to know what version of HTML you're using, because the instructions will be slightly different. So figure it out now, or add one if you don't have one.

How to do it

You may already have a DOCTYPE. View source on your home page; your DOCTYPE (if you have one) will be at the very top, even before the `<html>` tag.

- If you're using the default template of Movable Type, your DOCTYPE will probably include the phrase "XHTML 1.0 Transitional". This is fine.
- If you're using one of the default templates of Radio Userland, Manila, or Blogger, your DOCTYPE will probably include "HTML 4.01 Transitional". This is also fine.
- Other valid DOCTYPEs include phrases like "HTML 4.01 Strict", "XHTML 1.0 Strict", "XHTML 1.1", and a few others. These are all fine.

If you have a DOCTYPE, don't change it. However, if your source shows no DOCTYPE before your `<html>` tag, add this one:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

For technical reasons that I would rather not go into at the moment (see the "Further Reading" section below if you're interested), it is possible that you will see slight changes in your page layout after adding this DOCTYPE. If (and only if) this happens to you, you can compromise and use half a DOCTYPE instead, like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Note that every page of your web site should include a DOCTYPE, so you should check all your templates.

- Movable Type users should check the "Main Index", "Master Archive Index", "Category Archive", "Date-Based Archive", and "Individual Entry Archive" templates, plus any other archive templates you have created manually.
- Radio Userland and Manila users should check both the "Main template" and the "Home page template".
- Greymatter users should check "Main Index-Related Templates", "Archive-Related Templates", and "Entry-Related Templates".
- Blogger users should put a DOCTYPE in your main template. If your "Archive template" is a separate page (that is, if it has an `<html>` tag at the top), it should also have a DOCTYPE.

The important thing to know for the rest of the series is whether you're using HTML 4 (any variant), XHTML 1.0 (any variant), or XHTML 1.1. You'll see why tomorrow.

Further reading

- *A List Apart*: [Fixing Your Site With The Right DOCTYPE](#)
- *MSDN*: [Quirks mode in IE 6](#). (Note: this site does not work in some versions of Netscape and Mozilla. This is Microsoft's fault, not mine.)
- [Quirks mode in Mozilla](#).

Day 7: Identifying your language

You know what language you're writing in, so tell your readers... and their software.

Who benefits?

1. [Jackie](#) benefits. Her screen reader software ([JAWS](#)) needs to know what language your pages are written in, so it can pronounce your words properly when it reads them aloud. If you don't identify your language, JAWS will try to guess what language you're using, and it can guess incorrectly, especially if you quote source code or include other non-language content in your pages.
2. [Google](#) benefits, even if you are writing in English, but especially if you are writing in some other language. According to the [Google Zeitgeist](#), 50% of Google users search in languages other than English, and many of these users specify in their [Google preferences](#) to only search for pages in specific languages. Google's language auto-detection algorithms are better than most, but why make Google's job more difficult?

How to do it

First, get the right two-character language code. The code for English is "en"; the code for French is "fr"; German is "de". If you're writing in another language, [look up your language code here](#). Language codes are not case-sensitive.

Now put your language code in your `<html>` tag. Exactly how you do this depends on what version of HTML you're using. [Look at your DOCTYPE](#), then do one of the following:

1. If you're using any variant of HTML 4, change your `<html>` tag to this (use your own language code if not English):

```
<html lang="en">
```

2. If you're using any variant of XHTML 1.0, change your `<html>` tag to this (use your language code in both places):

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en"
xml:lang="en">
```

3. If you're using XHTML 1.1, change your `<html>` tag to this (again, insert your own language code):

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

Like the DOCTYPE, you should identify your language on every page of your web site.

One additional note: if you have more than one language on a page, you can identify the language on any enclosing element. For instance, if your web site uses HTML 4 and is primarily in German, but you quote an article in English, you could mark it up like this:

```
<html lang="de">
...
<blockquote lang="en">
...
</blockquote>
```

Further reading

- [List of language codes](#)
- [The lang attribute in the HTML specification](#)
- [Jim Thatcher on lang attribute support in screen readers](#). He describes their language auto-detection algorithms as "flaky", and he would know, since he designed the algorithm for [Home Page Reader](#). Other screen readers have added support for the lang attribute since his comments were written, but their auto-detection algorithms are still flaky, making it even more important and useful to identify your language.

Day 8: Constructing meaningful page titles

Every page of your web site should have a unique and meaningful page title.

- The home page title can simply be the name of your web site.
- Date-based archive pages should include the name of your web site, followed by the date (or date range) for the page. For example, on my weblog I have daily archives whose titles look like "[dive into mark/June 19, 2002](#)", and monthly archives like "[dive into mark/June 2002](#)".
- Category pages should include the name of your web site, followed by the name of the category. For example, all of my CSS-related posts are archived on a page entitled "[dive into mark/CSS](#)".
- Individual entry archive pages should include the name of your web site, followed by the entry title. I don't have separate pages for individual entries, but [Jonathon Delacour](#) does, and he gets this right. For example, his post of June 17, 2002, [Accessibility matters](#), is archived on its own page with the title "Jonathon Delacour: Accessibility matters".

The exact punctuation is not relevant, although some screen readers will read every punctuation character out loud by default. As a general rule, excessive punctuation sounds as dumb as it looks.

Who benefits?

1. [Jackie](#) benefits. [JAWS](#) has a special keyboard shortcut (INSERT + F10) which displays (and reads) a list of the currently open windows, by window title. In the case of web pages, this would be your page title. It also reads the window title while ALT-TABbing through open windows. Other screen readers, like [Home Page Reader](#), read the page title out loud as soon as you visit the page.
2. [Marcus](#) benefits. [Lynx](#) displays the page title in the first line of output, so it's always the first thing that Marcus reads in Braille.
3. [Bill](#) benefits. Because of his stroke, he sometimes gets confused and momentarily loses track of what he's reading. The page title in the window titlebar acts as a visual anchor; it stays in the same place, even as he scrolls the page. He can always glance back to it to jog his memory.
4. [Google](#) benefits. Google displays the page title in its search results, and [it ranks keywords higher when they appear in the page title](#). This is a Good Thing for you, especially for those individual entry pages. (Choosing good entry titles doesn't hurt either.)

How to do it

Movable Type has separate templates for the various types of index and archive pages. The default templates are quite accessible already; if you are using the default template, you don't need to make any changes.

1. Main Index: `<title><$MTBlogName$></title>`
2. Archive Index: `<title><$MTBlogName$> Archives</title>`
3. Category Archive: `<title><$MTBlogName$>: <$MTArchiveTitle$></title>`
4. Date-Based Archive: `<title><$MTBlogName$>: <$MTArchiveTitle$></title>`
5. Individual Entry Archive: `<title><$MTBlogName$>: <$MTEntireTitle$></title>`

Greymatter has a similar set of templates, but a different templating language. Greymatter does not have a separate template variable for the name of the weblog, so insert your own weblog title in each case.

1. Main Index Template: `<title>My Weblog</title>`
2. Archive Master Index Template: `<title>My Weblog Archives</title>`
3. Archive Log Index Template: `<title>My Weblog: {{month}} {{year}}</title>`

4. Entry Page Template: `<title>My Weblog: {{entrysubject}}</title>`

Manila (at least in the default configuration) lets you specify a title for each day, so you should use that in the page title instead of the date, since it is more likely to be relevant to the content.

1. Home Page Template: `<title>{siteName}</title>`

2. Template: `<title>{siteName}: {title}</title>`

Radio is a little trickier, you can still add the date to your date-based archives by using Radio's macro language. Be careful copying and pasting this macro; there should be no line breaks anywhere, and Radio cares. (Thanks to Jake Savin for these instructions.)

1. Home Page Template: `<title><%title%></title>`

2. Main Template: `<title><%title%><%local (d); if
radio.weblog.file.getArchiveFileDate (radioResponder.fileBeingRendered,
@d) {": " + string.dateString (d)} else {""}%></title>`

Unfortunately, I do not know how to customize page titles satisfactorily in Blogger. [Suggestions welcome.](#)

Keep in mind these are only suggestions. You can include the word "Archives" in the daily and monthly archive pages, or not. The exact punctuation really doesn't matter, as long as it's not excessive. You can put the site name at the end rather than at the beginning. It's a good idea to include your site name somewhere in your page titles, though; it's an important contextual clue, especially when people are switching between multiple open windows.

Further reading

- *Jake Savin*: [Adding a date to your Radio archive pages](#)

Day 9: Providing additional navigation aids

You may be familiar with the `<link>` tag in relation to external stylesheets. But did you know you can also use a similar syntax to point to your home page, and to previous and next pages in a series? For instance, on daily archive pages, you could point to the previous day's posts, and the next day's (if any). If you have individual pages for each entry, you could point to the previous and next entry.

```
<link rel="home" title="Home" href="http://url/of/home/page" />
<link rel="prev" title="Title of previous page"
href="http://url/of/previous/page" />
<link rel="next" title="Title of next page"
href="http://url/of/next/page" />
```

These links, normally invisible to visual browsers like Internet Explorer, can be displayed in alternate browsers and help users navigate through your web site. You probably already provide several ways to navigate: weekly or monthly archives, links to recent posts, a monthly calendar of daily posts. You may even already have visible links on your archive pages pointing to previous and next days or entries. Those are all great; keep them, and add these too.

Who benefits?

1. **Marcus** benefits. His text-only browser [Lynx](#) displays the additional navigation aids at the top of the page, using the title that we specified in the `title` attribute. Reading the daily archive page of June 18, this is what Marcus sees:

```
#Home June 17, 2002 June 19, 2002
```

(The `#` character lets Marcus know that this is metadata, not page content. Lynx does the same thing on the line with your [meaningful page title](#).)

2. **Michael** benefits. His text-only browser [Links](#) (not to be confused with Marcus's browser, [Lynx](#)) also displays these additional navigation aids at the top of the page, like this:

```
Link: home
Link: prev
Link: next
```

3. **Bill** benefits. Mozilla displays the additional navigation links in the Site Navigation toolbar. The type of the link ("home", "prev", "next") is displayed on the toolbar button, and the title of the link is displayed as a tooltip. (Note: the Site Navigation toolbar was in Mozilla 0.99, was removed from version 1.0 at the last minute, and will return in version 1.1. In versions that support it, you can display it under the View menu, Show/Hide, Site Navigation Bar, Show Always.)
4. **iCab** users benefit. iCab displays the navigation links in a drop-down menu in the toolbar, using the title defined in each link.

How to do it

In Movable Type, add these lines to your Date-Based Archive template, immediately after the `<head>` tag:

```
<link rel="home" href="<$MTBlogURL$" title="Home" />
<MTArchivePrevious>
<link rel="prev" href="<$MTArchiveLink$" title="<$MTArchiveTitle$"
/>
```

```

</MTArchivePrevious>
<MTArchiveNext>
<link rel="next" href="<$MTArchiveLink$>" title="<$MTArchiveTitle$>"
/>
</MTArchiveNext>

```

And add this to your Individual Entry Archive template, again immediately after the <head> tag:

```

<link rel="home" href="<$MTBlogURL$>" title="Home" />
<MTEEntryPrevious>
<link rel="prev" href="<$MTEEntryLink$>" title="<$MTEEntryTitle$>" />
</MTEEntryPrevious>
<MTEEntryNext>
<link rel="next" href="<$MTEEntryLink$>" title="<$MTEEntryTitle$>" />
</MTEEntryNext>

```

In Greymatter, add this immediately after the <head> tag in your 4 Entry Page Templates:

```

<link rel="home" title="Home" href="{{pageindexlink}}">
<link rel="prev" title="{{previousentrysubject}}"
href="{{entrieswebpath}}/{{previousentrynumberpadded}}.html">
<link rel="next" title="{{nextentrysubject}}"
href="{{entrieswebpath}}/{{nextentrynumberpadded}}.html">

```

In Radio, the entire matter is simplified by [Sjoerd Visscher's Navigation Links For Radio](#), a set of macros to do exactly this.

1. [Download Navigation Links macros](#).
2. Unzip the download and copy the 4 files (navigationLinks.txt, nextDayLink.txt, prevDayLink.txt, permalinkUrl.txt) into your Macros folder. In the standard installation on Windows, this would be C:\Program Files\Radio UserLand\Macros.
3. Insert this code in your Main Template, immediately after your <head> tag:

```
<%navigationLinks()%>
```

Unfortunately, I do not know how to satisfactorily implement previous and next links in Manila or Blogger, but you can at least add the link to your home page, immediately after your <head> tag:

```
<link rel="home" title="Home" href="http://url/of/your/home/page">
```

Further reading

- [Sjoerd Visscher. Navigation links in your Radio Userland weblog.](#)

Day 10: Presenting your main content first

One of the main advantages of using a purely CSS-based layout is that it is easy to rearrange elements within your HTML source without affecting the visual layout, so that your main content displays while the rest of the page is still loading. However, I am aware that most web sites still use table-based layouts, so this tip is for you.

If you have a table-based layout with a navigation bar along the left, your navigation bar is being presented to blind users like [Marcus](#) and [Jackie](#) before your main content. There is no way to describe how much of a problem this is; you have to see it for yourself:

1. [Sample table-based layout](#).
2. [Modified layout, with content first](#). These two layouts should look essentially the same in visual browsers, but in Lynx, the difference is obvious.
3. [The original layout, rendered by Lynx](#).
4. [The modified layout, rendered by Lynx](#). The main content is displayed first, then the navigation bar.

You do not need to redesign your entire template from scratch to avoid this problem. There is a (relatively) simple technique, affectionately called the "table trick", that can present your main content first, while still keeping your navigation bar on the left side.

Who benefits?

1. [Marcus](#) benefits. As demonstrated by the examples above, Lynx displays content in the order in which it appears in the HTML source. This means Marcus must scroll through your entire navigation bar *every time he visits your page*. Scrolling sucks.
2. [Jackie](#) benefits. [JAWS](#), like Lynx, presents content in the order in it appears in the HTML source code, not the order they appear on screen. With JAWS, the problem is even worse, because Jackie must sit through JAWS *reading your entire navigation bar* before hearing any real content, and there is no sure-fire way to jump straight to the main content. (We'll talk more about this problem on Monday.)
3. [Google](#) benefits. Google gives more weight to content closer to the top of the page. That's the top of your HTML source, not the visual top of the page. In fact, most people who know about this technique are in the search engine optimization industry; to them, the accessibility benefits are secondary.

How to do it

View your own site in the [Lynx Viewer](#) and see if your daily posts are displayed first, before your navigation bar. The Movable Type default template gets it right; if you use the default template or something based on it, you probably do not need to do anything. But view your site in the Lynx Viewer anyway, because it will give you a deeper understanding of the issues involved.

If you are using one of the default Radio templates, you may need to adjust your tables to put your main content first. There is no specific copy-and-paste way to do this; you will have to dig into your own template and look at the table structure. The [sample layout](#) and [modified sample layout](#) show the basic technique.

Instead of the obvious table layout:

```
<table>
```

```
<tr>
  <td valign="top" align="left" width="25%">
    ... navigation bar ...
  </td>
  <td valign="top" align="left">
    ... main content ...
  </td>
</tr>
</table>
```

We do this instead:

```
<table>
<tr>
  <!-- spacer GIF in upper-left cell -->
  <td></td>
  <!-- main content cell first, with rowspan=2 -->
  <td valign="top" align="left" rowspan="2">
    ... main content ...
  </td>
</tr>
<tr>
  <td valign="top" align="left" width="25%">
    ... navigation bar ...
  </td>
</tr>
</table>
```

Further reading

- [Lynx Viewer](#).
- *A Promotion Guide: [The Table Trick](#)*.

Day 11: Skipping over navigation links

If you didn't manage to hack your templates to [present your main content first](#), here's a compromise: provide an link to skip over your navigation links. It's not a perfect solution (presenting your main content first is better), but it's an accepted compromise that many sites use.

This "skip link" is just a regular `<a>` tag, like any other link, but we'll use CSS to hide it from visual browsers like Internet Explorer and Netscape. It won't affect your page layout at all; it will be completely invisible.

Who benefits?

1. [Marcus](#) benefits. When he visits your page, [Lynx](#) will display the link and allow him to skip over your navigation bar and go straight to your main content. See [Day 10: Presenting your main content first](#) for an example of why this is so important.
2. [Jackie](#) benefits. When she visits your page, [JAWS](#) will read the skip link and allow her to skip over your navigation bar and go straight to your main content.

How to do it

First, use [Lynx Viewer](#) on your own home page to determine whether your navigation bar is presented before your main content. If your main content comes first, this tip does not apply to you; enjoy your day off.

Now define a CSS rule for the skip links, to make them invisible to visual browsers. If you have an external stylesheet, put this rule at the end of it. (If you have multiple external stylesheets, put this rule in the Netscape 4-friendly one.) If you just have a `<style>` section at the top of your template, add this rule immediately after the `<style>` tag.

```
.skiplink {display:none}
```

Next, insert the actual skip link immediately after your site name and site description. Can't find them? Search for the appropriate template variables.

- Movable Type: search for `<${MTBlogTitle$}>` and `<${MTBlogDescription$}>`.
- Greymatter: there is no specific template variable; search for the name and tagline of your web site.
- Radio: search for `<%siteName%>` and `<%description%>`.
- Manila: search for `{homePageLink (siteName)}` and `{tagLine}`.
- Blogger: search for `<${BlogTitle$}>`.

Found them? Immediately *after* your site name and site description, insert the skip link:

```
<a class="skiplink" href="#startcontent">Skip over navigation</a>
```

OK, now you need an anchor tag where the skip link should point to; this should be at the start of your main content. Can't find your main content? Don't panic. Template variables save the day again.

- Movable Type: search for `<MTEntries>`.
- Greymatter: on your Main Index Template, search for `{{logbody}}`. On your Entry Page Templates, search for `{{entrymainbody}}`.
- Radio: search for `<%bodytext%>`.
- Manila: search for `{bodytext}`.
- Blogger: search for `<Blogger>`.

Now, the format of your anchor tag depends on what kind of HTML you're using. [Look at your DOCTYPE](#), then do one of the following:

1. If you're using any variant of HTML 4, add this just before your main content:

```
<a name="startcontent"></a>
```

2. If you're using any variant of XHTML 1.0, add this just before your main content:

```
<a name="startcontent" id="startcontent"></a>
```

3. If you're using XHTML 1.1, add this just before your main content:

```
<a id="startcontent"></a>
```

You should include this kind of skip link on every page of your web site, so add it to all your templates.

Day 12: Using color safely

This tip is a general rule that applies to many areas of web design, but I will focus on a specific example that is common among weblogs: link text.

There are two potential problems related to color. First, your link text may not contrast sufficiently with your background color. Any very light color on a white background is trouble; the link text may simply disappear into the background. Similarly, a dark color on a black background is trouble. This actually applies to all text, not just links, but it's fairly common on weblogs for text to be readable and links to be made unintentionally unreadable, which is why I mention links in particular.

The second potential problem is the link decoration. If your CSS redefines a rule to make your links a different color, you need to make sure that the links are also distinguishable in some other way, like bold, italic, or underline. Otherwise, the link text might be perfectly readable, but colorblind people won't be able to tell that it's a link. This is illustrated below.

Who benefits?

1. [Michael](#) benefits. Here is a sample screenshot of three different decoration schemes for links.

1. [This link](#) is underlined, the default behavior.
2. **This second link** is bold and colored instead.
3. [However](#) this link is distinguished only by color.

As shown, the link in the first sentence uses the default scheme, and displays blue/purple and underlined in visual browsers. The second link has two forms of text decoration applied, and displays bold and red (but not underlined). The third link has only one form of text decoration applied, and displays only as red.

Now here are the same three links, as Michael sees them.

1. [This link](#) is underlined, the default behavior.
2. **This second link** is bold and colored instead.
3. [However](#) this link is distinguished only by color.

As shown, the first link is still visible; Michael's colorblindness is not affected by the color blue. In the second sentence, the redness of the link fades, almost to black, but the link still appears bold, so Michael can still distinguish it. The problem occurs in the third link, which was previously only distinguished by its redness; now that the redness has faded to black, it is virtually impossible to tell which word is a link and which words are normal text.

How to do it

To check for "sufficient contrast" between your text color and background color, use [VisCheck](#) to simulate what your web page looks like to a colorblind reader.

To check for link decoration problems, look at your CSS rules for "a" tags. For example, if you have a rule like this in your CSS, then your links are *only* distinguished by their redness, which is no good:

```
a {
  text-decoration: none;
  color: red;
}
```

You can keep your links red, but you need to make sure that the links are *also* bold, or underlined, or italicized. To make them bold as well as red, add one line:

```
a {
  text-decoration: none;
  color: red;
  font-weight: bold; /* add this line */
}
```

Further reading

- [VisCheck](#) simulates colorblindness and allows you to see what colorblind people see. You can check a single image or an entire web page.
- *Cal Henderson: Color Vision*. Shows the color spectrum as seen by people with various types of colorblindness.
- [Ishihara Test for Color Blindness](#) contains a series of images that people with red–green colorblindness see differently, or don't see at all.

Day 13: Using real links

The scourge of web design is the "javascript:" link, a pseudo-link that executes a piece of Javascript code when you click on it. The most common place this problem occurs in weblogs is in the link to display comments in a separate window. Why is it a problem? Because [11% of Internet users don't use Javascript](#) for one reason or another, including many disabled users whose browsers simply don't support it. These pseudo-links won't work for them; use real links instead.

Although it's easy to describe and simple to fix, I can't stress enough how important this tip is. Some problems, like [not having a "skip link" past your navigation bar](#), reduce usability to varying degrees, but at least your page can be read eventually. On the other hand, this problem actually makes entire chunks of important content completely inaccessible. If your comments are hidden behind a "javascript:" link, *they may as well not exist*.

Who benefits?

1. [Marcus](#) benefits. [Lynx](#) does not support Javascript.
2. [Michael](#) benefits. [Links](#) does not support Javascript.
3. [Lillian](#) benefits. Although she uses Internet Explorer, her IT department has implemented a corporate-wide policy to disable Javascript on all but a small list of approved sites. Your web site is not on the list.
4. [Google](#) benefits. Google wants to follow links to find and index more content, but it can't follow "javascript:" links, because it doesn't execute Javascript code as it indexes the web.

How to do it

The default templates in Movable Type and Radio now get this right, so you may not need to do anything. View source on your home page and search for "javascript:". If you don't find it, this tip does not apply to you.

However, if your Movable Type template contains a link like this:

```
<a href="javascript:OpenComments(<$MTEntID$>)">Comments  
(<$MTEntCommentCount$>)</a>
```

Then change it to this:

```
<a href="<$MTCGIPath$>mt-comments.cgi?entry_id=<$MTEntID$>"  
onclick="OpenComments(<$MTEntID$>); return false">Comments  
(<$MTEntCommentCount$>)</a>
```

In Javascript-enabled browsers, it will still work the same way, because the `onclick` attribute takes precedence over the `href` attribute. So the new version still calls the `OpenComments` function, which pops up a new window. However, non-Javascript-enabled browsers (and Google) will ignore the `onclick` attribute entirely and follow the link specified in the `href`, which will display the comments in the same window.

If you're using `javascript:` pseudo-links for any other reason, stop. Just stop. Do not pass go, do not collect \$200, etc. Apply the above technique to your own code so that non-Javascript-enabled browsers always have a chance to follow a real link.

Further reading

- *Jeff Howden*: [Links & JavaScript Living Together in Harmony](#).

Postscript

Don't even get me started on those [dynamic Javascript-based menu systems](#). They make you look cool like smoking makes you look cool. Use real links.

Day 14: Adding titles to links

What with the web being all about links, you would think more people would know about the `title` attribute, but I rarely see it. For those who don't know, all links can have a title, specified by the `title` attribute of the `<a>` tag. This is in addition to whatever link text you specify. The title of a link generally shows up as a tooltip in visual browsers, but it can be presented in non-visual browsers as well.

Not all links should have titles. If the link text is the name of an article, don't add a title; the link text itself is descriptive enough. But if you read the link text by itself, out of context, and can't figure out what it points to, add a title.

Who benefits?

1. **Jackie** benefits. **JAWS** has an option to read the title of a link along with the link text. (This option is not on by default. To activate it, Jackie pressed **INSERT+V** to bring up the JAWS verbosity options window, then changed "Text links verbosity" to "Alt tag or title".)
2. **Michael** benefits. When he moves his cursor over a link in **Opera**, it displays the title of the link in the status bar and as a tooltip. This lets him decide whether he wants to spend precious bandwidth following the link.
3. **Lillian** benefits. When she moves her cursor over the link in Internet Explorer, it displays the title of the link as a tooltip.
4. **Marcus** benefits. When Marcus presses "1", **Lynx** displays a list of links on the current page. The list includes the title of each link, if present.

How to do it

On each link where the link text itself might not be sufficient for the reader to decide whether to click the link, add a `title` attribute. Examples:

1. On my navigation bar, I have a link to my statistics page. The link text is simply **Statistics**, but the `title` attribute gives some further information:

```
<a title="referrers and other visitor statistics"
href="/stats/">Statistics</a>
```

2. On my navigation bar, I have a link to my book, **Dive Into Python**, which looks like this:

```
<a title="Free Python book for experienced programmers"
href="http://diveintopython.org/">Dive Into Python</a>
```

3. When I link to an article using a phrase within a sentence, I try to use a `title` attribute to give identifying information about the link, such as the article title or a citation. For instance, **yesterday's tip** included this sentence:

```
Why is this a problem? Because <a title="TheCounter.com
statistics on Javascript usage in browsers, April 2002"
href="http://www.thecounter.com/stats/2002/April/javas.php">11%
of Internet users don't use Javascript</a> for one reason or
another, including many disabled users whose browsers simply
don't support it.
```

Which renders like this:

Why is this a problem? Because [11% of Internet users don't use Javascript](#) for one reason or another, including many disabled users whose browsers simply don't support it.

Do not go overboard with the `title` attribute. All things in moderation.

Further reading

- *Jakob Nielsen*: [Using Link Titles to Help Users Predict Where They Are Going](#).

Day 15: Defining keyboard shortcuts

One of the least known features of HTML is the `accesskey` attribute for links and forms, which allows the web designer to define keyboard shortcuts for frequently-used links or form fields. On Windows, you can press ALT + an access key; on Macintosh, you can press Control + an access key. If the access key is defined on a link, your browser will follow the link; if defined on a form field, your browser will set focus on that field. Internet Explorer has supported access keys since version 4, Netscape since version 6. Older browsers simply ignore them, with no harmful effect.

While there are no standards for which keys should be assigned to which features, here are some commonly-used keyboard shortcuts:

Access key 1

Home page

Access key 2

Skip to main content (the [navigation bar skip link](#))

Access key 9

Feedback

Who benefits?

1. **Jackie** benefits. When **JAWS** reads a link that defines an `accesskey`, it announces the access key as well. For example, the link `Home page` would be read by JAWS as "link: Home page, ALT + 1". Jackie can focus on the link by pressing **ALT+1**, then follow it by pressing **ENTER**.
2. **Bill** benefits. Since Bill can not use a mouse effectively since his stroke, he relies on keyboard navigation and keyboard shortcuts to move around the page. Access keys are an excellent way for him to jump to common or frequently-used links. Bill can type **ALT+1**, and **Mozilla** immediately follows the link that defines `accesskey="1"`. (Note: Mozilla does not announce access keys, which raises the question of how Bill would discover what they are. We will discuss this in a future tip.)

How to do it: home page link

- **Movable Type**: The default template does not have a link to the home page, so you should make your site name into a link, and give it an `accesskey`. Search your template for `<$MTBlogName$>`, and change it to this:

```
<a href="<$MTBlogURL$>" style="color:black;
text-decoration:none" accesskey="1"><$MTBlogName$></a>
```

- **Radio**: search your template for `{siteName}`. It will probably be in a link, something like this:

```
<a href="<%radio.macros.weblogUrl ()%>" style="color:black;
text-decoration:none"><%siteName%></a>
```

Simply add an `accesskey` attribute to the link, like this:

```
<a href="<%radio.macros.weblogUrl ()%>" style="color:black;
text-decoration:none" accesskey="1"><%siteName%></a>
```

- **Blogger**: search your template for `<$BlogTitle$>`. If it's enclosed in an `<a>` tag, add the `accesskey="1"` attribute to the `<a>` tag, like the above Radio example. If your `<$BlogTitle$>`

is *not* enclosed in an `<a>` tag, enclose it in one like this (insert your own home page address):

```
<a href="http://address/of/your/home/page" style="color:black;
text-decoration:none" accesskey="1"><$BlogTitle$></a>
```

How to do it: skip navigation link

Do you have a [link to skip over your navigation bar](#)? If so, give it an `accesskey="2"`.

```
<a class="skiplink" href="#startcontent" accesskey="2">Skip over
navigation</a>
```

How to do it: feedback link

Do you have a link to a feedback form, or a link to your email address? If so, give it an `accesskey="9"`.

```
<a href="mailto:me@mydomain.com" accesskey="9">Email me</a>
```

Note: Radio weblogs generally have a link to a feedback form (the little envelope icon), but the link is generated by a macro, so you will not be able to add an `accesskey` to it.

Be sure to include each `accesskey` on each page of your web site; make these changes to all your relevant templates.

Further reading

- *Jukka Korpela*: [Improving accessibility with accesskey in HTML forms and links](#). Explains why all my suggested `accesskey` codes are numbers, instead of letters.
- *Paul Bohman*: [Access keys, IE6](#). Part of a discussion of `accesskey` on the [Web Accessibility Forum Mailing List](#).

Day 16: Not opening new windows

The one thing every web user understands is the "Back" button. It's an integral part of browsing the web. Follow a link, go back. Explore a search engine result, go back. Even my father can do this, and he's still excited when he can double-click the "Internet" icon successfully on the first try.

In all dominant browsers, using the `` tag to force a link to open in a new window *breaks the Back button*. The new window does not retain the browser history of the previous window, so the "Back" button is disabled. This is incredibly confusing, even for me, and I've been using the web for 10 years. In 2002, it's amazing that people still do this. Don't do this. Don't force links to open in new windows.

Please note that this tip is about you as a web designer, not you as a web user. If *you* want to open new windows while *you* browse, go right ahead. In Internet Explorer for Windows, hold down the **shift** key while you click a link to open the link in a new window. In Netscape 6 and Mozilla, hold down **Control**. In Internet Explorer for Mac, hold down **Command**. (Some browsers such as Opera support advanced combinations like **Control + Shift** + click to open a link in a new window in the background.) The point is that the choice of whether a link will open in a new window should be the end user's choice, not the web designer's choice.

Who benefits?

1. [Jackie](#) benefits. Although [JAWS](#) does announce "New browser window" when a link opens a new window, this is easy to miss, as it is spoken wedged between the reading of the link text and reading of the new page. [Home Page Reader](#) has a better solution; it plays a distinctive sound every time a new window opens. And [Window Eyes](#), another popular screen reader, gives no indication of new windows at all.

And regardless, the "Back" button is still broken. If Jackie misses the "new browser window" announcement, she can not simply glance at her taskbar and see that two browser windows are open. She will need to read through her entire list of open windows, either using the JAWS-specific shortcut **INSERT+F10** to get a window list, or the standard **ALT+TAB**.

2. [Lillian](#) benefits. Her Internet Explorer window is always maximized (so she can see it), and new windows also open maximized by default. Furthermore, Windows XP groups multiple windows of the same application in the taskbar, so there is virtually no visible indication that a new window has even been opened. Suddenly, the "Back" button is disabled for no apparent reason, and Lillian has no idea why. If you were expecting her to read the rest of your web site after following that link, you can forget it.
3. [Bill](#) benefits. His sister has set Mozilla to use tabbed browsing, so that Bill can look at the tabs and quickly remind himself which windows he has open, and also so he can quickly switch between them (using **CTRL+PAGEUP** and **CTRL+PAGEDOWN** on his handy dandy keyboard extension). Links that are forced to open in a new window will open an entirely new Mozilla window. Not only will this bypass his tabbed browsing preferences, but it will make it appear that all his open windows have disappeared, since the new browser window will not show the tabs that were open in the previous window.

How to do it

1. Don't use `` to force links to open in a new window.
2. If you absolutely must open a link in a new window, explicitly warn the reader. This is a non-optimal, compromise solution, usually brought about by business requirements of "not being associated" with external content. For example, [CNN's "related sites" page](#) does this.

3. If you have a "Links open new windows" checkbox, make sure it is *off* by default.

Further reading

- *Jakob Nielsen: The Top Ten New Mistakes of Web Design*. "#1: Breaking or Slowing Down the Back Button. #2: Opening New Browser Windows."
- *W3C Web Accessibility Initiative: Example for Checkpoint 10.1* gives an example of how to warn users if you have a single link that you absolutely must open in a new window.
- *W3C Validator mailing list: Re: opening a link in a new window*. For those who care about this sort of thing, you should know that the `target` attribute of the `<a>` tag is deprecated, and will prevent your pages from validating in HTML 4.01 Strict, XHTML 1.0 Strict, or any future version.
- *WebAIM mailing list: mailto: links open new windows*. The consensus is that `mailto:` links are not an accessibility problem, even though they generally open your email client in a new window, because this behavior is completely determined on the client side. A web-based mail form (like Radio uses) may be a better overall solution, provided the form is accessible. A web-based form will work for visitors without integrated email clients (by misconfiguration or by circumstance, such as being in a public lab), and it protects your email address from spam harvesters without resorting to inaccessible Javascript tricks. On the other hand, some people *really like* their email clients due to familiarity, functionality (such as built-in spell checking), and the ability to archive outgoing messages. I am not recommending one method over another.

Day 17: Defining acronyms

I used 50 acronyms and abbreviations on my own weblog last month: ADA, ALT, AOL, API, CGI, CMS, CSS, CTRL, DMV, DNS, DTD, EFF, FAQ, FSF, GFDL, GIA, GPL, HTML, IE, IIRC, IIS, IO, KB, KDE, LONGDESC, MB, MSDN, MSN, MT, Mac, NC, OPML, P2P, PGDN, PGUP, PBS, PDF, PONUR, RSS, RU, SOAP, SSN, TDD, US, VNC, W3C, WCAG, WYSIWYG, Win, XHTML, and XML.

If you recognize all 50, congratulations; you have a long and prosperous future as a technical editor. If not, you'll appreciate the fact that I defined each of them with the `<acronym>` tag. Hover your cursor over each acronym to see what it stands for. This works in all modern browsers, and is harmless in Netscape 4.

You should define an acronym whenever you use it, or at least once per post.

Who benefits?

1. [Michael](#) benefits. When Michael hovers his cursor over an acronym, [Opera](#) displays the acronym title as a tooltip.
2. [Bill](#) benefits. [Mozilla](#) goes even further, automatically rendering acronyms with a dotted underline. When Bill hovers his cursor over the acronym, Mozilla changes the cursor to a cursor + question mark, and then displays the acronym title as a tooltip. (You can override this default behavior with cascading style sheets, or use CSS to get a similar effect in other browsers.)
3. [Google](#) benefits. Google indexes the acronym title as well as the acronym itself, so people can find your site whether they search for the acronym or the spelled-out description.
4. I wish I could say that [Jackie](#) benefits, but she doesn't. Neither [JAWS](#) nor any of the other screen readers on the market currently support reading the titles of acronyms. I hope some day they will, and then you'll be ahead of the game.

How to do it

The first time you use an acronym, mark it up with an `<acronym>` tag, like this:

```
<acronym title="cascading style sheets">CSS</acronym>
```

Radio users can automate this markup by using shortcuts. From your Radio home page, click "Shortcuts" in the main navigation menu, then define the acronyms you use frequently. For example:

Name: `CSS`

Value: `<acronym title="cascading style sheets">CSS</acronym>`

(Be sure to change the input type from "WYSIWYG" to "Source" so you can type the HTML directly.)

Then, in your post, simply type "CSS" (with the quotes), and Radio will render it with the acronym tag and the title, just as you defined it.

How to do it: cascading style sheets

As an added bonus, you can change the look of all your acronyms using cascading style sheets. This works in all tools, not just Radio. Here is the rule I use to produce the dotted underline in all browsers (not

just Mozilla):

```
acronym {  
  border-bottom: 1px dotted black;  
}
```

And as an extra bonus, this is the rule I use in my print stylesheet to automatically spell out acronyms when printing my web pages. (This only works when printing from Mozilla and Opera, but it's harmless in other browsers.)

```
acronym:after {  
  content: " (" attr(title) ")";  
}
```

Further reading

Have you been using acronyms without knowing what they mean? Look them up.

- [Acronym Finder](#).
- [Acronym Database](#).
- [Acronym Search](#).

Postscript

Several fellow markup-obsessed gurus have correctly pointed out that there is an `<abbr>` tag for abbreviations. Unfortunately, no version of Internet Explorer for Windows supports it; no tooltips show up at all. Use `<acronym>`.

Day 18: Giving your calendar a real caption

"But," I hear you cry, "my calendar already *has* a caption. Look right there, it has the month and year right at the top. In bold, even."

But if you dig into your HTML source, you'll see that your calendar does not have a *real* caption. It most likely has a single `<td>` table cell defined to span the entire first row, with a CSS rule to make it look bold. This is so much easier with a real `<caption>` tag. It's easier to read in your template, saves a few bytes in your page, looks exactly the same in visual browsers, and is more accessible.

Who benefits?

1. [Marcus](#) benefits. [Lynx](#) displays the caption with the word "CAPTION:" in front of it, making it perfectly clear that this line is the caption and not the column headers or table data.
2. [Jackie](#) benefits indirectly. Using a real `<caption>` tag clears the way for using real table headers, which benefits Jackie in ways we'll discuss tomorrow.

How to do it

You can only do this in publishing tools that support a calendar (which rules out Blogger) and that allow you to customize the HTML generated for calendars (which rules out Manila).

In Movable Type, you probably have a table for your calendar in your Main Index template that starts like this (searching for "calendarhead" is likely to find it):

```
<table border="0" cellspacing="4" cellpadding="0">
<tr>
<td colspan="7" align="center"><span class="calendarhead"><$MTDate
format="%B %Y"$></span></td>
</tr>
<tr>
<td align="center"><span class="calendar">Sun</span></td>
...
```

Leave the table tag alone, but replace that entire first `<tr>` table row with a real `<caption>` tag, like this:

```
<table border="0" cellspacing="4" cellpadding="0">
<caption class="calendarhead"><$MTDate format="%B %Y"$></caption>
<tr>
<td align="center"><span class="calendar">Sun</span></td>
...
```

Leave the rest of it alone; we'll fix it tomorrow.

The `class` attribute on the `<caption>` is optional; I left it in there so this could be a drop-in replacement in the default Movable Type template, which uses a CSS rule to make the month and year bold. Using the `<caption>` tag as shown, your page should look exactly the same as it did before.

In Greymatter, the concept is the same but the template tags are different:

```
<caption>{{monthword}} {{yearyear}}</caption>
```

Again, you could change the visual style of the `caption` using cascading style sheets, if you're into that sort of thing.

In Radio, the process is somewhat more difficult, but not impossible. (I am indebted to [Tony Bowden](#) for these instructions.)

1. In Radio, open the actual Radio application. On Windows, right-click on the little Radio icon in your system tray and select "Open Radio".
2. Under the "Tools" menu, select "Developers", then "Jump..." (**Control+J**). Jump to "system.verbs.builtins.radio.weblog.drawCalendar" (no quotes).
3. Now go to "Edit" menu, "Find and Replace", "Find..." (**Control+F**) and find "hCalendarTable". This should reveal the block of code that draws the initial `<table>` tag and the fake calendar caption.
4. Change the last line of that block (that writes out the `monthYearString` in a `<tr>` tag) to this:

```
add ("<caption>" + monthYearString + "</caption>")
```

5. Close the window. It will ask you if you want to compile the changes, say yes.
6. If you like, you can style the caption. Go to your Home Page Template (on the Prefs page) and add styles for `caption`. This is what I use. Where my `<style>` section used to contain this:

```
body, td, p {  
    font-family: verdana, sans-serif;  
    font-size: 12px;  
}
```

It now contains this:

```
body, td, p, caption {  
    font-family: verdana, sans-serif;  
    font-size: 12px;  
}  
  
caption {  
    text-align: center;  
    font-weight: bold;  
}
```

Further reading

- [Tony Bowden: Changing the Calendar in Radio.](#)

Day 19: Using real table headers

If you have a calendar on your web site, it should be rendered as an HTML table. There have been a few attempts to create pure-CSS calendars. This is misguided; calendars are data tables, and should be marked up as such.

The most important thing about a data table is marking up the headers properly. In the case of a calendar, this means the days of the week along the top. You should definitely include headers for the days of the week; if you don't want them actually visible, you can make them invisible with CSS. (I do this on my own weblog.) But the headers need to be there nonetheless, because screen readers rely on them to help blind users navigate through the table without getting lost.

The thing about a calendar (and any data table, really, but we're mostly talking about calendars today) is that it's very easy to use if you can see it all at once, but very difficult to use if you can only see one day at a time. Imagine that you have a day-by-day calendar on your desk, but each page lists only the day of the month, not the day of the week. Page after page: 1, 2, 3, 4, 5, 6, 7... Today is 4, which I happen to know is a Thursday. Now skip ahead to 11, 12, 13. Quick: which day of the week is 13? The page doesn't tell you; you have to keep track of it yourself.

This is what it's like for a blind user to navigate a calendar without proper headers. You get a bunch of numbers, but no context to keep track of them. Adding proper headers to the calendar allows screen reader software to associate the table header (day of the week) with the table data (day of the month), and it reads them together. "Thursday 4, Thursday 11, Friday 12, Saturday 13." Oh, it's a Saturday.

Note I said *proper* headers. Putting the days of the week in `<td>` tags in the first row is not enough. They need to be `<th>` tags instead. Most weblog templates get this wrong, but it's simple enough to fix, and your calendar will look exactly the same in visual browsers once you're done.

Who benefits?

1. [Jackie](#) benefits. When she encounters your calendar, [JAWS](#) first [reads the caption](#), then announces the headers, then Jackie can hold down `Control + ALT` and move through the table with the arrow keys. As she moves, JAWS announces the header (day of the week) and the cell data (day of the month).

All major screen readers allow this kind of table navigation. [Home Page Reader](#) allows users to switch to "Table Navigation" mode (`ALT+T`), then move through the calendar without holding down additional modifier keys. Home Page Reader actually goes one step beyond JAWS. As we'll see in a minute, you can define a shorter (or longer) title for each table header (sort of like [adding a title to a link](#)), and Home Page Reader will read that instead of the original table header text. This means you can visually display your days of the week as "Sun", "Mon", "Tue", but you can tell Home Page Reader to read them as "Sunday", "Monday", "Tuesday". Cool.

How to do it

If you didn't do it already, make sure your calendar [has a real caption](#). The `<caption>` tag must be the first thing after the `<table>` tag, and the row of `<th>` tags should be the first thing after that.

In Movable Type, find the calendar in your Main Index Template. (Again, searching for "calendarhead" will probably find it.) Immediately after the `<caption>`, you'll see the days of the week defined like this:

```

<tr>
<td align="center"><span class="calendar">Sun</span></td>
<td align="center"><span class="calendar">Mon</span></td>
<td align="center"><span class="calendar">Tue</span></td>
<td align="center"><span class="calendar">Wed</span></td>
<td align="center"><span class="calendar">Thu</span></td>
<td align="center"><span class="calendar">Fri</span></td>
<td align="center"><span class="calendar">Sat</span></td>
</tr>

```

Change it to this:

```

<tr>
<th abbr="Sunday" align="center"><span
class="calendar">Sun</span></th>
<th abbr="Monday" align="center"><span
class="calendar">Mon</span></th>
<th abbr="Tuesday" align="center"><span
class="calendar">Tue</span></th>
<th abbr="Wednesday" align="center"><span
class="calendar">Wed</span></th>
<th abbr="Thursday" align="center"><span
class="calendar">Thu</span></th>
<th abbr="Friday" align="center"><span
class="calendar">Fri</span></th>
<th abbr="Saturday" align="center"><span
class="calendar">Sat</span></th>
</tr>

```

Two things going on here: all the `<td>` tags change to `<th>`, and they each get an `abbr` attribute to specify the full name of the day of the week. (The `abbr` attribute can serve a dual purpose. For very long table headers, it serves as an abbreviation; hence the name. But for very short table headers, it serves as a longer version, which is what we're doing here.)

In Radio, the procedure is similar to what you did yesterday, giving the table a real `caption`.

1. In Radio, open the actual Radio application. On Windows, right-click on the little Radio icon in your system tray and select "Open Radio".
2. Under the "Tools" menu, select "Developers", then "Jump..." (**Control+J**). Jump to "system.verbs.builtins.radio.weblog.drawCalendar" (no quotes).
3. Now go to "Edit" menu, "Find and Replace", "Find..." (**Control+F**) and find "addDayName". This should reveal and highlight the `addDayName` function. Double-click the triangle to reveal the actual function code, which should look like this:

```

on addDayName (name)
  add ("<td width=\"19\" height=\"10\" align=\"center\"
  style=\"font-size:9px\">" + name + "</td>")

```

4. Change it to this:

```

on addDayName (name, fullname)
  add ("<th abbr=\"\" + fullname + "\" width=\"19\"
  height=\"10\" align=\"center\" style=\"font-size:9px\">" +

```

```
name + "</th>")
```

5. Now double-click on the "for i = 1 to 7" line just below it to see this:

```
for i = 1 to 7
  addDayName (radio.string.getLocalizedNameString
    ("dayOfWeekShort." + i))
```

6. And change it to this:

```
for i = 1 to 7
  addDayName (radio.string.getLocalizedNameString
    ("dayOfWeekShort." + i), radio.string.getLocalizedNameString
    ("dayOfWeek." + i))
```

Very important note about layout tables

Tables used exclusively for visual layout have a completely different set of rules. Do *not* use <th> tags on layout tables. Aside from tweaking your tables to [present your main content first](#), there's very little you need to do to make layout tables accessible. We'll discuss one small thing tomorrow.

Further reading

If you need to mark up data tables more complex than a calendar (like tables with multiple levels of headers and subheaders), you're on your own. Here are some starting points:

- *Jim Byrne*: [Table Manners](#).
- *WebAIM*: [Create tables that transform gracefully](#).
- *Kynn Bartlett*: [Understanding Accessible Table Markup](#).

Day 20: Providing a summary for tables

The final piece of marking up tables is providing a summary. The summary of a table is never displayed in visual browsers; it is exclusively designed for screen readers and speech browsers. It is exactly what it sounds like: a summary, a longer description than the caption. It is usually read immediately before the caption.

Every table should have a summary. If you have a calendar, the summary can be as simple as "Monthly calendar with links to each day's posts." If you use tables for layout, you should give each of those tables an empty summary, to indicate that the table is used exclusively for visual layout and not for presenting tabular data. (This is a similar concept to providing an empty ALT attribute on images used exclusively for visual spacing. We'll discuss these "spacer images" on Monday.)

Who benefits?

1. [Jackie](#) benefits. When [JAWS](#) encounters your calendar, Jackie hears "Summary: Monthly calendar with links to each day's posts." Then [she hears the caption](#), then [she hears the table headers](#), then she can navigate through the calendar.
2. [iCab](#) users benefit. iCab can use the built-in text-to-speech capabilities of the Mac OS to read web pages, and it will read the summary of any table that defines one.

How to do it: calendar

In Movable Type, find the calendar in your Main Index Template. (Again, searching for "calendarhead" will probably find it.) You'll see a `<table>` like this:

```
<table border="0" cellspacing="4" cellpadding="0">
```

Change it to this:

```
<table border="0" cellspacing="4" cellpadding="0" summary="Monthly  
calendar with links to each day's posts">
```

In Radio, the procedure is similar to what we've done the past few days.

1. In Radio, open the actual Radio application. On Windows, right-click on the little Radio icon in your system tray and select "Open Radio".
2. Under the "Tools" menu, select "Developers", then "Jump..." (**Control+J**). Jump to "system.verbs.builtins.radio.weblog.drawCalendar" (no quotes).
3. Now go to "Edit" menu, "Find and Replace", "Find..." (**Control+F**) and find "draw the month and year". This should reveal and highlight a line that simply says "bundle // draw the month and year". Double-click the triangle to reveal the actual code, which should look like this:

```
add ("<table cellspacing=\"0\" border=\"0\"  
class=\"hCalendarTable\">"); indentLevel++
```

4. Change it to this:

```
add ("<table summary=\"Monthly calendar with links to each  
day's posts\" cellspacing=\"0\" border=\"0\"  
class=\"hCalendarTable\">"); indentLevel++
```

How to do it: layout tables

If you use tables for layout, add `summary=""` to each table. This is best accomplished with search-and-replace. Search for this:

```
<table
```

And replace it with this:

```
<table summary=""
```

Day 21: Ignoring spacer images

Many designers use transparent spacer images to control the layout of their web site in visual browsers. You may use as many as you like, but you need to explicitly specify an empty `alt` attribute on each spacer image so that non-visual browsers know to ignore them.

Who benefits?

1. **Marcus** benefits. By default, **Lynx** displays the filename of any image that does not contain an `alt` attribute. Many popular weblog templates include several spacer images even before the site name. You don't notice them in your visual browser, of course, but this is what Marcus sees:

```
[shim.gif] [shim.gif]
[shim.gif]
[shim.gif]
Welcome To My Web Site
[ciblueHeader2.gif]
```

```
[ciblueCurve2.gif]
```

2. **Jackie** benefits. By default, **JAWS** reads the filename of any image that does not contain an `alt` attribute. If you thought Marcus was annoyed, imagine how frustrating it is for Jackie to hear this:

```
graphic shim dot gif graphic shim dot gif graphic shim dot gif graphic shim dot gif
welcome to my web site graphic ciblu header two dot gif graphic ciblu curve two
dot gif
```

If you introduced yourself like that in real life, nobody would talk to you.

How to do it

Radio users can take the day off. As of last Monday, Radio automatically generates empty `alt` attributes for all spacer images. (Thanks, **Jake**.) If you view source on your home page and you're not seeing `alt=""` on any of your spacer images, update `Radio.root` and republish your site.

Users of other publishing tools should look through your template for `` tags with filenames like "spacer.gif", "shim.gif", "l.gif", or any image that appears to be repeated several times within your template, possibly with different `width` and `height` attributes each time.

For example, for each spacer image that looks like this:

```

```

Change it to this:

```

```

If you re-use the same spacer image multiple times, it's probably easiest to do this with global search-and-replace.

Things not to do

1. Don't define `alt=" "`. The `alt` attribute should be an empty string, not a space.
2. Don't specify an `alt` attribute on your `<body>` tag, even if it defines a background image. This background image is always ignored by non-visual browsers. It looks like this:

```
<body background="http://url/to/image.gif">
```

3. Don't specify an `alt` attribute on `<td>` tags, even if they define background images. These background images are always ignored by non-visual browsers. They look like this:

```
<td background="http://url/to/image.gif">
```

Further reading

- *WebAIM: [How to Create Accessible Graphics](#)*.

Day 22: Using real lists (or faking them properly)

Suppose you have a blogroll of three links: Slashdot, The Register, and dive into mark. Instead of a boring black bullet, you want a fancy-looking image next to each one. How do you do it? The most common solution is to use `` tags next to each link. This works, and can easily be made more accessible with the addition of proper `alt` text on each image.

However, you can also go further and use real list markup (`` and `` tags), then use CSS to change the boring black bullet into an image. Besides being "the right way to do it" in some academic sense that you may or may not care about, this technique has additional accessibility benefits.

I'll give examples of both techniques in a minute.

Who benefits?

1. **Marcus** benefits. [As we saw yesterday](#), **Lynx** displays the filename of any image without an `alt` attribute.
2. **Michael** benefits. **Links** never displays images, but by default it does *not* display anything for images with no `alt` text. This was acceptable yesterday when we wanted to [ignore spacer images](#), but today we want to make sure that there is some indication that this is a list, so we need that `alt` text.

Also, when Michael browses with images turned off, **Opera** will display the `alt` text instead of a "missing image" block. And if you use the advanced technique, Opera will do even better; it reverts to displaying the normal black bullet instead of the "missing image" block.

3. **Jackie** benefits. [As we saw yesterday](#), **JAWS** reads the filename of any image without an `alt` attribute. The links end up getting lost in a sea of irrelevant filenames. This is what Jackie hears:

fancy dot dot gif link slashdot, fancy dot dot gif link the register, fancy dot dot gif link
dive into mark

Providing an asterisk as the `alt` text helps enormously. JAWS will see that the image is being used as a bullet, and not announce it. However, **Home Page Reader** will still announce the asterisk explicitly, so users will hear this:

asterisk link slashdot, asterisk link the register, asterisk link dive into mark

Using real list markup is best. Since all the visual presentation is in the CSS declarations, none of it clutters up the page, so both JAWS and Home Page Reader simply read your list for what it is: a list. Now this is what it sounds like:

link slashdot, link the register, link dive into mark

How to do it

If you have a blogroll that looks like this:

```
 <a  
href="http://www.slashdot.org/">Slashdot</a> <br>  
 <a  
href="http://www.theregister.co.uk/">The Register</a> <br>  
 <a
```

```
href="http://diveintomark.org/">dive into mark</a> <br>
```

You should provide `alt` attributes for all your bullet images. Use an asterisk as the `alt` text, to simulate what the list would look like if you were using real list markup. (To prevent this from displaying as a tooltip in visual browsers, you should also provide an empty `title` attribute.)

```
 <a href="http://www.slashdot.org/">Slashdot</a> <br>
 <a href="http://www.theregister.co.uk/">The Register</a>
<br>
 <a href="http://diveintomark.org/">dive into mark</a>
<br>
```

How to do it: advanced

The advanced (and preferred) technique uses CSS to define the image to use as a list bullet.

```
<style type="text/css">
ul.blogroll {
  list-style: url(/images/fancydot.gif) disc;
}
</style>
```

Then in your template, you can write your list using real list markup:

```
<ul class="blogroll">
<li><a href="http://www.slashdot.org/">Slashdot</a></li>
<li><a href="http://www.theregister.co.uk/">The Register</a></li>
<li><a href="http://diveintomark.org/">dive into mark</a></li>
</ul>
```

Results:

- Modern browsers will display the image as the list bullet.
- Browsers with images turned off will display the boring black bullet.
- Netscape 4 will always display the boring black bullet.
- Text-only browsers always ignore CSS, so they will display the list however they normally display lists (usually rendering the list bullet as an asterisk).

Postscript: un-bulleted lists

Another common way to create a list of links is with no images whatsoever, just a heap of links, possibly right-aligned, like this:

```
<div align="right">
<a href="http://www.slashdot.org/">Slashdot</a><br>
<a href="http://www.theregister.co.uk/">The Register</a><br>
<a href="http://diveintomark.org/">dive into mark</a><br>
</div>
```

This can also be accomplished with CSS and real list markup:

```
<style type="text/css">
ul.blogroll {
  list-style: none;
  text-align: right;
}
</style>
```

Or, if you want left-aligned links, you can do this instead:

```
<style type="text/css">
ul.blogroll {
  list-style: none;
  margin-left: 0;
  padding-left: 0;
}
</style>
```

Either way, the list markup hasn't changed from the previous example:

```
<ul class="blogroll">
<li><a href="http://www.slashdot.org/">Slashdot</a></li>
<li><a href="http://www.theregister.co.uk/">The Register</a></li>
<li><a href="http://diveintomark.org/">dive into mark</a></li>
</ul>
```

The "list-style: none" line suppresses the usual black bullet in visual browsers. This works in all browsers, even Netscape 4. Thanks to [Tobias Schmidt](#) for reminding me of this technique.

Further reading

- *Tobias Schmidt*: [Styling lists with CSS](#).
- *W3Schools*: [CSS List Properties](#).
- *Eric Meyer*: [Lists and Indentation](#).
- *Eric B. Bednarz*: [Manipulating Margin and Padding of Lists With CSS](#).

Day 23: Providing text equivalents for images

This is the most important day of the series, so pay attention:

Every single image on every single page of your site should have a text equivalent, so-called "alt text", specified in the `alt` attribute of the `` tag.

Screen readers read it, text-only browsers display it, Google indexes it, and visual browsers can display it as a tooltip or on the status line (although you as the designer can override this). We've already seen how to specify [empty alt text for spacer images](#), and several ways to [create accessible lists with image bullets](#). What's left?

- Permalink icons
- "Powered by" icons
- Mail-to icons
- XML icons
- Small graphics floated within your item posts
- Any other images you've added to your template

They all need appropriate `alt` text.

Who benefits?

1. [Jackie](#) benefits. [JAWS](#) reads the `alt` text. Without valid `alt` text, Jackie hears the filename instead, [which sounds horrible](#).
2. [Marcus](#) benefits. [Lynx](#), as a text-only browser, doesn't display any images, only `alt` text. Without `alt` text, Lynx displays the filename, which looks as bad as JAWS sounds.
3. [Michael](#) benefits. [Links](#), as a text-only browser, doesn't display any images, only `alt` text. Without `alt` text, Links does not display anything for an image (unless the image is a link, in which case Links simply displays "[IMG]"). When browsing with [Opera](#) with images turned off, Michael sees the `alt` text in place of the image.
4. [Lillian](#) benefits. Internet Explorer displays the `alt` text as a tooltip (although you as a designer can suppress this).
5. [Google](#) benefits. The Googlebot indexes `alt` text, which is used not only in matching keywords in normal searches, but also in [image searches](#). (How did you think that worked?)

How to do it

The default Movable Type template doesn't include any `` tags. If you are using a graphical "Powered by Movable Type" image, you should make sure the `` tag includes the attribute `alt="Powered by Movable Type"`.

The default Greymatter template includes only one image, generated by the `{{gmicon}}` template variable. This generates an `` tag which includes appropriate `alt` text, "Powered by Greymatter".

Radio auto-generates appropriate `alt` text for the following standard icons:

- XML coffee mug: `alt="Subscribe to <site name> in Radio UserLand."`
- XML icon: `alt="Click to see the XML version of this web page."`
- Mailto icon: `alt="Click here to send an email to the editor of this weblog."`

However, Radio users will need to manually specify `alt` text for customized images. Go to `Prefs`, then `Customized Images` (under `Templates`), and add these `alt` attributes:

- Day-level permalink: `alt="Permanent link: <%longDate%>"`.
- Item-level permalink: `alt="Permanent link"`.
- Source link icon: `alt="source"`.
- Enclosure link icon: `alt="enclosure"`.

You could also add `title=""` to suppress tooltips in visual browsers.

Of course, regardless of your publishing tool, if you've added your own images to your template, or if you have small graphics floated in your item posts, they each need appropriate `alt` text. Since I learn best by example, here are some examples. More general principles and examples are listed in the "further reading" section.

Examples of bad `alt` text

- Any HTML markup. `alt` text can only contain plain text and entities, no tags.
- `alt="filename.jpg"`. This doesn't get us anywhere. What is the graphic's function? We don't care what it's called.
- `alt="alt text"`. Inserted by some HTML editors as a reminder, and left there by clueless designers.
- `alt="Click here!"` Serves no useful purpose (unless of course it's on a graphic that says "Click here!").
- `alt="Turn images on!"` This is like being asked by a blind person what time the clock says, and responding, "Just open your eyes!" Images may be off for a reason (Michael's low bandwidth), they may be unavailable (Marcus's text-only browser), or they may not even be off at all (Jackie's screen reader, which displays images but reads `alt` text aloud).
- [More examples of bad `alt` text](#).

Examples of good `alt` text

- [Jonathon Delacour](#) has a graphic in his page banner of a Chinese symbol. `alt="Site logo: xin, the Chinese character for heart"`.
- [Leslie Harpold](#) has a graphical page banner that includes the site name, "The Historical Present", and a tagline, "Hypermodernism has a posse". `alt="the historical present: hypermodernism has a posse"`.
- [Simon Willison](#) has a "W3C XHTML 1.0" sticker. `alt="Valid XHTML 1.0!"`
- [Jeffrey Zeldman](#) has a navigation bar of text-as-graphics; on rollover, each graphic puts a little tagline in the status bar with Javascript. Of course, blind users will miss this, so Zeldman also puts the same text in the `alt` text of each graphic. Slick.
- [Dean Allen](#) has a graphic in his page banner that includes the name of the site and a tagline. His `alt` text is on the long side and includes a different tagline (a bit confusing), but Dean is cool enough to get away with it. `alt="Textism is an ephemera focused on the composition, design, and reading of text. In addition, there will be pie"`. Note: you are probably not cool enough to get away with this. Keep it simple.

Further reading

- *A. J. Flavell*: [ALT texts in IMG](#).
- *Jukka Korpela*: [Simple guidelines on using ALT texts in IMG elements](#).

- *Ian Hickson*: [Mini-FAQ about the alternate text of images.](#)
- *Watchfire.com*: [Provide alternative text for all images.](#)
- *All My FAQs Wiki*: [ALT attribute.](#)
- *WebAIM*: [How to Create Accessible Graphics.](#)
- *Martin Schrode*: [On accessible advertising.](#)
- *Section 508 Federal Accessibility Guidelines*: [What is meant by a text equivalent?](#)

Day 24: Providing text equivalents for image maps

I was surprised to find how many high-profile web sites use client-side image maps. I don't use them myself, and I don't think they're included in any default weblog templates, but apparently lots of people have figured them out. If you don't know what an image map is, [Leslie Harpold](#) uses one for the navigation links at the bottom of her home page. That's all one image, but clicking on the word "archives" takes you to one page, "by category" to another, and so forth.

Image maps sound like an accessibility nightmare, but they're not. In the same way that [every image needs a text equivalent](#), every image map and every clickable area of the image map needs a text equivalent. You can provide `alt` text for the image itself (in the `` tag), and for each clickable area in the image map (in the `<area>` tags of the associated `<map>`, that defines where the clickable areas are and what they link to).

Who benefits?

1. [Marcus](#) benefits. [Lynx](#) displays the `alt` text of the image as a link. When Marcus hits **ENTER**, Lynx displays a separate page listing the links in the image map. Each link is labeled by `alt` text of the area in the image map. Without `alt` text, Lynx displays the link address of each area, which may be incomprehensible.

If [Leslie](#) didn't have `alt` text on her image map, this is the link Marcus would see at the bottom of her home page:

```
[USEMAP:hpfooter.gif]
```

Following this link would take Marcus to a page that lists all the links in the image map. Without `alt` texts, Lynx could only display each URL, which would look like this:

```
[USEMAP:hpfooter.gif]
```

```
MAP: http://leslie.harpold.com/#Map
```

1. <http://leslie.harpold.com/archives.html>
2. <http://leslie.harpold.com/category/>
3. <http://leslie.harpold.com/links.html>
4. <http://leslie.harpold.com/leslie.html>
5. <http://www.moveabletype.org>

However, in reality, Leslie *does* have proper `alt` text for her image and every area of her image map. So this is the link Marcus *really* sees at the bottom of her home page:

```
Site navigation links
```

Following this link takes Marcus to a page that looks like this:

```
Site navigation links
```

```
MAP: http://leslie.harpold.com/#Map
```

1. [previously...](#)
2. [by category](#)
3. [about the site](#)
4. [about leslie](#)

2. **Michael** benefits. **Links** displays the `alt` text of image as a link. When Michael hits **ENTER**, Links pops up a menu of all the links defined in the map. Each link is labeled by the `alt` text of the area. Without the `alt` text, Links displays the link address of each area, which may be incomprehensible.
3. **Jackie** benefits. **JAWS** will read the `alt` text of each area of the image map, in the order in which they are defined in your HTML source. Jackie can hit **ENTER** to follow the link. Without the `alt` text, JAWS reads the link address of each area, which is almost certainly incomprehensible. (Have you ever tried reading a long web address to someone over the phone?)
4. **Lillian** benefits. Internet Explorer displays a tooltip when hovering over each linked area in the image map.
5. **Google** benefits. The Googlebot indexes the `alt` text of the main image and each area within the image map. The `alt` text is a factor in determining both your page's relevance to keywords, and also each link's relevance to keywords contained in the `alt` text of that area.

How to do it

If you have an image map like this:

```


<map name="Map">
<area shape="rect" coords="203,114,258,129" href="/archives.html">
<area shape="rect" coords="277,113,348,129" href="/category/">
<area shape="rect" coords="364,113,401,128" href="links.html">
<area shape="rect" coords="418,114,488,130" href="leslie.html">
<area shape="rect" coords="-4,190,131,210"
href="http://www.moveabletype.org">
</map>
```

Add `alt` text to both the main image, and to each linked area within the image map, like this:

```


<map name="Map">
<area alt="previously..." shape="rect" coords="203,114,258,129"
href="/archives.html">
<area alt="by category" shape="rect" coords="277,113,348,129"
href="/category/">
<area alt="about the site" shape="rect" coords="364,113,401,128"
href="links.html">
<area alt="about leslie" shape="rect" coords="418,114,488,130"
href="leslie.html">
<area alt="Powered by Movable Type" shape="rect"
coords="-4,190,131,210" href="http://www.moveabletype.org">
</map>
```

All the rules about [writing good alt text for images](#) also apply to image maps. You could also add `title=""` to the main `` and each `<area>` to suppress the tooltip in visual browsers.

Things not to do

Don't create server-side image maps, images that pass your exact click coordinates back to the server for further processing. These are *completely inaccessible* to JAWS users like Jackie, users of text-only browsers like Michael and Marcus, keyboard-only users like Bill, and search engines like Google. If you must use server-side image maps, add a text-only navigation bar below it that includes [real text links](#) to every page you could get to by clicking on the image map.

Further reading

- *Leslie Harpold: [The Historical Present](#)*. Leslie kindly allowed me to use her weblog as the basis of today's example.

Day 25: Using real horizontal rules (or faking them properly)

Suppose you want a divider between your posts. Regular horizontal rules (`<hr>` tags) are boring, so you use an image instead. This works, and can easily be made more accessible with the addition of proper `alt` text.

However, you can also go further and use a real horizontal rule, then use a little CSS trick to display it as an image in modern browsers. Older browsers and text-only browsers will ignore the CSS and just render a horizontal rule in their native style. (Text-only browsers generally use a row of underscores or dashes, expanded to fit the current screen width.)

I'll give examples of both techniques in a minute.

Who benefits?

1. **Jackie** benefits. [As we've already seen](#), **JAWS** reads the filename of any image without an `alt` attribute.
2. **Marcus** benefits. **Lynx** displays the filename of any image without an `alt` attribute. If you use a real horizontal rule, Lynx will render it as a series of underscores as wide as the current screen.
3. **Michael** benefits. **Links** does *not* display anything for images without `alt` text, so Michael does not have any indication that there is a divider. We need that `alt` text, or better yet, a real `<hr>` tag, which Links will render as a series of dashes as wide as the current screen.

How to do it

If you use images as horizontal rules, the easiest way to make them accessible is add an `alt` attribute to your `` tag. You should add an empty `title` attribute too, to suppress the tooltip in visual browsers. So if you have this:

```

```

Change it to this:

```

```

Do not go crazy and specify 80 dashes for the `alt` text. Two or three will suffice.

How to do it: advanced

The advanced (and preferred) technique uses an actual `<hr>` tag. However, since browser support for styling `<hr>` tags directly is flaky at best, we'll use a dummy `<div>` tag to display the image. Put the following CSS in your `<style>` section at the top of your template. (If you're using an external stylesheet like `style-sites.css`, put it anywhere in there. If you're using multiple stylesheets, put it in the Netscape 4-friendly one.)

```
div.hr {display: none}
/**/a{
div.hr {
```

```
display: block;
height: 25px;
background-image: url(/images/fancyrule.gif);
background-repeat: no-repeat;
background-position: center center;
margin: 1em 0 1em 0;
}
hr {display:none}
/* */
```

(For the `height`, substitute the height of your image. For the `background-image`, substitute the address of your image.)

Then in your template, where you want your decorative rule, do this:

```
<div class="hr"></div><hr />
```

Results:

- All modern browsers will display the image instead of the normal plain horizontal rule.
- Netscape 4 will display a plain horizontal rule.
- Text-only browsers always ignore CSS, so they will display a plain horizontal rule (usually rendered as a series of underscores or dashes).

Further reading

- [Hiding CSS From Netscape 4](#) without using additional stylesheets. This is the technique we used above, in the advanced example.
- [CSS1 and the Decorative HR](#), if you're feeling particularly brave and want to style decorative horizontal rules with CSS in Netscape 4. Lots of luck with that.

Day 26: Using relative font sizes

Web sites, with few exceptions, center around words. News, opinions, thoughts, ideas, stories, original writing, e-commerce: all words. Visual design and images are important, to be sure, but if people can't read your words, what's the point?

In the fall of 2000, Jeffrey Zeldman famously said that [relative font sizing was impossible](#) ("pixels, baby... or nothing") because of an overwhelming variety of browser bugs, starting with Netscape 4 and ending in the most modern browsers. Since then, Netscape 4 still hasn't gotten any better, and it still hasn't gone away, but at least we've all learned a thing or two about taming the browsers and making relative font sizing a reality. (Zeldman too; his recently reincarnated [Web Standards Project](#) uses the technique described below.)

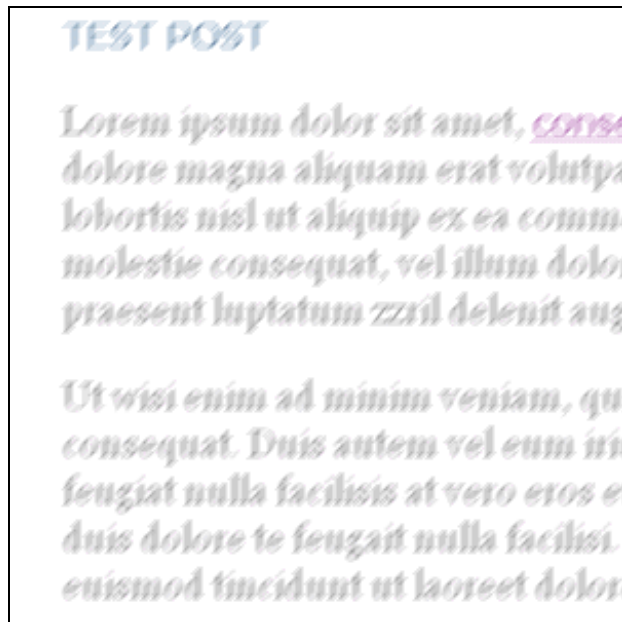
Use relative font sizes in browsers that can handle them, and absolute font sizes in Netscape 4, which does not reliably support relative font sizes. You can do this even if you don't use multiple stylesheets. In a minute, I'll give copy-and-paste solutions for the default Movable Type template and *all* default Radio themes. And a lengthy explanation of the technique itself to help you implement it in other templates.

Who benefits?

1. [Lillian](#) benefits. Lillian has difficulty seeing web pages clearly, due to nothing more than old age. Like 80% of the Internet population, she uses Internet Explorer for Windows, which does not support resizing text unless the web designer exclusively specifies relative font sizes. Lillian has changed the default text size in her browser (under the "View" menu, "Text Size"), but it doesn't do any good on sites that use absolute font sizes. This includes virtually every weblog template in existence. For example, this is what the default Movable Type template looks like to Lillian:



If the template used relative font sizes, it would look **exactly the same** to the majority of readers who don't need (or care) to change their text size. But this is what it would look like to Lillian:



Again: if people can't read your words, *what's the point?*

How to do it: Radio

In your Home Page Template, look in your `<style>` section near the top for a CSS rule that looks like this:

```
body, td, th, p {
  font-family: verdana, sans-serif;
  font-size: 12px;
}
```

Keep that just the way it is, but add this immediately after it:

```
/* */a{}
body,
body td,
body th,
body p {
  font-size: x-small;
  voice-family: "\"}\"";
  voice-family: inherit;
  font-size: small;
}
html>body,
html>body td,
html>body th
html>body p {
  font-size: small;
}
/* */
```

Make sure to include the comments at the beginning and the end. They're the key to the whole thing, as explained below.

How to do it: Movable Type

The default Movable Type template is more complex than the Radio templates, but we're going to do the same thing, just more of it. In your Stylesheet template (`styles-site.css`), add this at the end:

```
/**/a{}
body,
body a,
body .calendar,
body .calendarhead,
body .title,
body .sidetitle,
body .syndicate,
body .powered,
body .comments-post,
body .posted {
    font-size: xx-small;
    voice-family: "\"}\"\"";
    voice-family: inherit;
    font-size: x-small;
}
html>body,
html>body a,
html>body .calendar,
html>body .calendarhead,
html>body .title,
html>body .sidetitle,
html>body .syndicate,
html>body .powered,
html>body .comments-post,
html>body .posted {
    font-size: x-small;
}

body .date {
    font-size: x-small;
    voice-family: "\"}\"\"";
    voice-family: inherit;
    font-size: small;
}
html>body .date {
    font-size: small;
}

body #banner {
    font-size: 200%;
}

body .description {
    font-size: 60%;
}

body .blogbody {
    font-size: 110%;
}

body .blogbody,
body .calendar,
body .calendarhead,
body .side,
body .title,
body .sidetitle,
```

```

body .syndicate,
body .powered,
body .comments-body {
  line-height: 128%;
}
/* */

```

Again, make sure to include the comments at the beginning and the end.

How to do it: detailed explanation

The general idea is that we're going to use font-size keywords. These are little-used (due to bugs in older browsers), but they have three interesting properties:

1. They don't compound. If you have a "main" section sized at 90%, and within that you have a "post" section sized as 90%, some browsers will display the post at 81% (90% x 90%), but some will display it at 90%. With more than one level of nesting (common in templates that use tables for layout), text quickly becomes unreadably small as the percentages compound. However, if your "main" section is sized as `small`, and the "post" section within it is sized as `small`, all browsers will display the "post" section as `small`.
2. They resize properly in Internet Explorer for Windows. This is odd, since `small` sounds like an absolute size (especially in light of the fact that `small` nested within `small` is still `small`, see above), but it works. What can I tell you? IE/Win resizes text sized with font size keywords. I swear.
3. They never get *too* small. That "Text Size" setting that Lillian uses in Internet Explorer can be used to make text smaller as well as larger. Many people with good eyesight prefer everything one, even two, sizes smaller than normal. Text sized with percentages tends to become microscopic and fuzzy when combined with the smallest default text size setting. However, text sized with font size keywords always stays at least 9px, which is readable in any font (assuming good eyesight).

So we're going to use font size keywords to specify our basic sizes. And if we need finer control than that, we're going to use percentages, but only on leaf classes that contain text (so on "post", but not "main") to avoid compounding percentages, and not too small, to avoid becoming microscopic when combined with users' smaller default text sizes.

Here's the general idea of font size keywords:

```

p {
  font-size: 12px;
}

/**/a{}
body p {
  font-size: x-small;
  voice-family: "\"}\"";
  voice-family: inherit;
  font-size: small;
}
html>body p {
  font-size: small;
}
/* */

```

There's a lot going on here, and it's all important, so pay attention.

1. First, we're defining an absolute size (12px) for every `<p>`. All browsers apply this style, including

Netscape 4.

2. Then we include the odd-looking comment `/*/*`. Due to bugs in Netscape 4, everything between this comment and the following one will be ignored. That's right, all the following styles will only be applied in non-Netscape-4 browsers.
3. Immediately after the odd-looking comment, we include an empty rule `a {}`. Opera 5 for Mac is buggy and ignores this rule (and only this rule). It applies everything else.
4. We have now carved out a realm where we can define rules that are applied in all browsers except Netscape 4. Now we can start defining relative font sizes (which Netscape 4 can't handle). The first thing we do is use a `body p` selector to redefine the behavior of the `p` tag. Due to the way CSS works, this will override our previous `p` selector. (In technical terms, `body p` is a *more specific selector* than `p`.)
5. We redefine the font size of all `<p>` tags to be `x-small`. This is a font size keyword which, at default settings, Internet Explorer 5 for Windows will translate into 12px. However, if the user changes their "Text Size" (under the View menu), this text will scale larger or smaller, depending on the user's setting. This is what we want. (Note: we've now defined font-size *twice* for IE5/Win, but that's okay, because the more specific selector always wins, and the previous selector is simply ignored.)
6. Unfortunately, IE5/Win has an off-by-1 bug with its font size keywords; every other browser in the world (IE5/Mac, Netscape 6, Mozilla, IE6/Win) will translate `x-small` to 10px, not 12px. Luckily for us, IE5/Win has its own parsing bug that we can exploit: it looks at that odd-looking voice-family and mistakenly thinks that this entire `body p` selector is over, so it ignores all the lines until the `}`.
7. Now we have carved out a smaller realm where we can define rules that are applied in all browsers except IE5/Win (and Netscape 4, which is still blissfully ignoring all of this). So we redefine font-size to `small`, which modern non-IE5/Win browsers (the only ones still listening) correctly interpret as 12px (at default settings). Again, if the user sets their "text size" to larger, this text will scale larger, which is what we want.
8. But wait! Opera 5 has the same parsing bug that IE5/Win has, so it was also confused by the voice-family hack, but it correctly interprets font size keywords, so now our text will look too small in Opera 5. Luckily, Opera 5 supports a *third* type of selector, `html>body p`. (Again, this is "more specific" than `body p`, so it takes precedence over the previous selector.) IE5/Win does not support this type of selector, so it will just ignore it, which is what we want (since we've already compensated for its off-by-1 bug and don't want to go mucking that up now). IE6/Win also does not support it, but that's OK, because we caught it with the `font-size: small` after the "voice-family: inherit" hack in the `body p` selector. All other browsers support `html>body` selectors, so for them we end up defining font-size *four* times. Again, that's not a problem, because the most specific selector always wins, and the rest are simply ignored.
9. Finally, we have a set of empty comments: `/* */`. This triggers Netscape 4's parser to start listening again. If we defined any further rules after these empty comments, all browsers (including Netscape 4) would apply them.

To recap:

1. Netscape 4 displays `<p>` text at 12px, regardless of user setting.
2. Internet Explorer 5 for Windows displays `<p>` text at `x-small`, which works out to be 12px at the default setting, but would scale larger if the user set their "Text Size" setting larger in the View menu.
3. Internet Explorer 6 for Windows displays `<p>` text at `small`, because of the `font-size: small` rule in the `body p` selector. This works out to 12px at the default setting, but would scale larger if the user set their "Text Size" setting larger.
4. Internet Explorer 5 for Mac, Opera, Netscape 6, Mozilla, and (hopefully) all future browsers will display `<p>` text at `small`, because of the `font-size: small` rule in the `html>body p` selector. This works out to 12px at the default setting, but would scale larger if the user used the "Text Zoom"

feature.

Further reading

- *Mark Pilgrim*: [Relative Font Sizing HOWTO](#). Gives essentially this same explanation, but the page itself is an example of the technique, so you can see it in action.
- *Todd Fahrner*: [Size Matters: Making Font Size Keywords Work](#).
- *Caio Chassot*: [Hiding CSS from Netscape 4](#) without using multiple stylesheets.
- *Tantek Çelik*: [Box Model Hack](#). How to hide CSS from Internet Explorer 5 for Windows.
- [The Web Standards Project](#) also uses font size keywords with the IE5/Win hack, although they use a Javascript-based solution (instead of the inline comment hack) to deal with Netscape 4.
- *Owen Briggs*: [Text Sizing](#). Screenshots of various relative font sizing techniques across browsers, platforms, and default text size settings.

Day 27: Using real headers

Think of your web site as an outline. The top level is labeled by your site name. On your home page, you list entries from several days. So the second level is labeled by your date descriptions: "Tuesday, July 16, 2002", or something similar. On each day, you make multiple posts, which may each have their own title. If so, then you have a third level, labeled by your individual post titles.

Now mark up your web site as an outline, using real `<h1>`, `<h2>`, `<h3>` tags. Screen readers rely on these tags to interpret the structure of your pages. Your pages *do have* a structure, but without proper header tags, screen readers can't find it. In a minute, I'll show you how to use CSS to make your headers look the same in visual browsers as whatever ``-based monstrosity you're currently using.

Who benefits?

1. [Jackie](#) benefits. As soon as Jackie hits your page, [JAWS](#) announces that the page has a certain number of links and a certain number of headers. Jackie can type **INSERT+F6** to hear all the headers on your page, or **CTRL+INSERT+ENTER** to quickly navigate through your page by skipping to the next header.
2. [Michael](#) benefits. In [Opera](#), he can type **s** to skip to the next header, or **w** to skip to the previous one.
3. [Google](#) benefits. Google appreciates a well-structured page, and ranks keywords higher when they appear in real header tags. (Yet another reason to write well-crafted post titles.)

How to do it: Movable Type

1. Define the styles for your site logo. In your Stylesheet template (`styles-sites.css`), add the following lines:

```
h1, h2, h3 {
  margin: 0;
  padding: 0;
}

h1 {
  font-size: 20px;
  font-weight: normal;
}

/**/a{}
h1 {font-size: 100%}
/** */
```

2. Define your site logo using an `<h1>` tag. In your 4 major templates (Main Index, Category Archive, Date-Based Archive, Individual Entry Archive), search for this:

```
<div id="banner">
<MTBlogName$><br />
```

And replace it with this:

```
<div id="banner">
<h1><MTBlogName$></h1>
```

3. Define your date headers using `<h1>` tags. We already have a class defined for these, so we shouldn't need to make any stylesheet changes; we're just changing the tag. In your 4 major templates, search for this:

```
<div class="date">
<$MTEnterDate format="%B %d, %Y"$>
</div>
```

And replace it with this:

```
<h1 class="date">
<$MTEnterDate format="%B %d, %Y"$>
</h1>
```

4. Define your post titles using `<h2>` tags. Again, this only involves changing the tag, not the stylesheet. In your 4 major templates, search for this:

```
<span class="title"><$MTEnterTitle$></span>
```

And replace it with this:

```
<h2 class="title"><$MTEnterTitle$></h2>
```

How to do it: Radio

1. Define your header styles. The default Radio themes don't use any real header tags, so we'll need to define these styles ourselves. (Tailor to suit, but these examples should make your page look the same as it used to in visual browsers.)

Actually, before we start, search your Home Page Template for "h1 {}". If you find a rule like this, remove it; it's not actually used anywhere, and it'll get in our way:

```
h1 {
font-family: Verdana, Arial, Helvetica, sans-serif; font-size:
24px; font-weight: bold
}
```

2. OK, now add these styles, anywhere in the `<style>` section of your Home Page Template:

```
h1, h1 {
margin: 0;
padding: 0;
}

h1 {font-size: 24px}
h1 {font-size: 13px}

/**/a{}
h1 {
font-size: large;
voice-family: "\"}\"\"";
voice-family: inherit;
font-size: x-large;
}
html>body h1 {
font-size: x-large;
}
h1 {
font-size: x-small;
voice-family: "\"}\"\"";
voice-family: inherit;
font-size: small;
}
```

```
html>body h1 {
  font-size: small;
}
/* */
```

Note that we're using relative font sizes for our headers in browsers that can support relative font sizes, and absolute font sizes in Netscape 4. This technique should look familiar; [we did the same thing yesterday](#).

3. Define the header for your site name. In your Home Page Template, search for "<%siteName%" and find a line that looks like this:

```
<font size="+2"><b><a href="<%radio.macros.weblogUrl ()%"
style="color:Black;
text-decoration:none"><%siteName%></a></b></font>
```

And change it to this:

```
<h1><a href="<%radio.macros.weblogUrl ()%"
style="color:Black;
text-decoration:none"><%siteName%></a></h1>
```

4. Define the header for your date headers. In your Day Template, search for "<%longDate%" and find a line like this:

```
<b><%longDate%></b>
```

And change it to this:

```
<h1><%longDate%></h1>
```

Further reading

- [Shirley Kaiser. Don't Fake Your Markup](#): Accessibility Issues for CSS.

Day 28: Labeling form elements

Has it ever bothered you that web forms are so hard to use? For instance, in regular GUI applications, you can click anywhere on a checkbox label to check or uncheck the box, but in web-based applications, you can only click on the little checkbox square itself. This is annoying but not fatal. But for blind users, the situation is even worse. Even simple forms, like comment posting forms, can be difficult to use if you can't see them all at once. (We noted [a similar problem with tables](#); a weblog calendar is easy to use if you can see it all at once, but difficult if you can only see it one day at a time.)

There are HTML tags which can help make forms easier to use. I'll talk about one, the `<label>` tag; you can read about the others in the "Further reading" section.

The `<label>` tag allows you to associate a form label with any kind of form input element: text box, multi-line text area, checkbox, radio button, whatever. This allows screen readers to intelligently announce what a particular input element is, by reading the label. Furthermore, if you use a `<label>` tag to associate a checkbox (`<input type="checkbox">`) with the text next to it, your web-based form will work like a GUI application: clicking anywhere on the text label will toggle the checkbox.

Who benefits?

1. **Jackie** benefits. When Jackie tabs through a form, **JAWS** announces the name of each element (by its `name` property), which may or may not be intelligible. But if the form element is associated with a label, JAWS will read the label text instead. "Text: name." (**TAB**) "Text: email address." (**TAB**) "Text: URL." (**TAB**) "Text area: comments." And so forth.
2. **Lillian** benefits. Once form elements are associated with labels, Lillian can click anywhere on the text next to a checkbox, and it will toggle the checkbox. This gives a much wider margin of error for toggling the checkbox with a mouse, and with her limited vision, the wider the better.
3. **Bill** should benefit, but he doesn't yet. He navigates mostly with the keyboard, which mostly means the **TAB** key. When he tabs to a checkbox in a form, Mozilla should set a focus rectangle around the entire label, but it doesn't; it just puts a focus rectangle around the checkbox itself. (Internet Explorer gets this right, though. Even Netscape 4 gets this right. Bad Mozilla.)

How to do it: Movable Type

In Movable Type, edit your Comment Listing Template. Near the bottom, you should see a form that contains elements like this:

```
Name:<br />
<input name="author" /><br /><br />

Email Address:<br />
<input name="email" /><br /><br />

URL:<br />
<input name="url" /><br /><br />

Comments:<br />
<textarea name="text" rows="10" cols="50"></textarea><br /><br />

<input type="checkbox" name="bakecookie" />Remember info?<br /><br />
/>
```

Each of those naked labels needs to be wrapped in a `<label>` tag. Also, since the `<label>` tag points to a form element by ID (not name), each `<input>` tag will need an ID attribute. All in all, it will end up looking to this:

```
<label for="author">Name:</label><br />
<input id="author" name="author" /><br /><br />

<label for="email">Email Address:</label><br />
<input id="email" name="email" /><br /><br />

<label for="url">URL:</label><br />
<input id="url" name="url" /><br /><br />

<label for="text">Comments:</label><br />
<textarea id="text" name="text" rows="10" cols="50"></textarea><br />
<br />

<input type="checkbox" id="bakecookie" name="bakecookie" /><label
for="bakecookie">Remember info?</label><br /><br />
```

Be sure to make the same changes to your Comment Preview template, your Comment Error template, and your Individual Entry Archive.

How to do it: Greymatter

Under "Edit Karma & Comments–Related Templates", you should see a template called "`{{entrycommentsform}}` Posting form" that includes this:

```
Name
<BR>
<INPUT TYPE=TEXT NAME="newcommentauthor" SIZE=40>
<P>
E-Mail (optional)
<BR>
<INPUT TYPE=TEXT NAME="newcommentemail" SIZE=40>
<P>
Homepage (optional)
<BR>
<INPUT TYPE=TEXT NAME="newcommenthomepage" SIZE=40>
<P>
Comments
<BR>
<TEXTAREA NAME="newcommentbody" COLS=35 ROWS=10
WRAP=VIRTUAL></TEXTAREA>
```

Change it to this:

```
<label for="newcommentauthor">Name</label>
<BR>
<INPUT TYPE=TEXT id="newcommentauthor" NAME="newcommentauthor"
SIZE=40>
<P>
```

```
<label for="newcommentemail">E-Mail (optional)</label>
<BR>
<INPUT TYPE=TEXT id="newcommentemail" NAME="newcommentemail"
SIZE=40>
<P>
<label for="newcommenthomepage">Homepage (optional)</label>
<BR>
<INPUT TYPE=TEXT id="newcommenthomepage" NAME="newcommenthomepage"
SIZE=40>
<P>
<label for="newcommentbody">Comments</label>
<BR>
<TEXTAREA id="newcommentbody" NAME="newcommentbody" COLS=35 ROWS=10
WRAP=VIRTUAL></TEXTAREA>
```

Further reading

- [WebAIM: How to Create Accessible Forms](#). For more complex forms, additional accessibility-related tags like `<legend>` and `<fieldset>` may be required. This tutorial shows you what they are and how to use them.
- [W3C: Forms in HTML Documents: The LABEL element](#).

Day 29: Making everything searchable

Every web site needs a site search. Period.

Rules for a good site search:

1. Put the search box on every page, preferably "above the fold" (visible without scrolling).
2. Search everything by default. If you have an option to search entries, comments, or both, make "both" the default.
3. Don't clutter your search box with advanced options, like boolean logic, case sensitivity, or regular expressions. Choose defaults that mimic Google's behavior (match all words, don't match partial words, don't match case, don't use regular expressions) and make all the options visible on a separate "advanced search" page.
4. Give your search box [a proper label](#) and [an access key](#). I recommend `accesskey="4"`. (Note: [On day 15](#), I gave an example of how to assign an `accesskey` to your search form, and I got it wrong. For maximum cross-browser compatibility, you need to define the `accesskey` on the `<label>`, not on the `<input>`. See the examples below for the correct syntax.)

Who benefits?

[Jackie](#), [Michael](#), [Bill](#), [Lillian](#), [Marcus](#), and pretty much everyone else in the world benefit from a well-implemented site search. Especially on a weblog or news-oriented site, where content is primarily organized chronologically, it's very frustrating to try to find a specific post that's scrolled off the main page. Very few people know about the "site:domainname.com" syntax on Google (to restrict search results to a particular domain), and Google only reindexes once a month anyway. Provide your own site search.

How to do it

Greymatter has built-in search functionality, but you will need to customize the template slightly to provide an `accesskey` for the search box. Go to "Edit Templates", "Miscellaneous Templates", "Search Form Template", and find a form like this:

```
<FORM ACTION="{ {cgiwebpath} }/gm-comments.cgi" METHOD=POST>
<INPUT TYPE=TEXT NAME="gmsearch" SIZE=20>
<INPUT TYPE=SUBMIT VALUE="Search">
</FORM>
```

And change it to this:

```
<FORM ACTION="{ {cgiwebpath} }/gm-comments.cgi" METHOD=POST>
<label for="gmsearch" accesskey="4">Search for:</label>
<INPUT TYPE=TEXT id="gmsearch" NAME="gmsearch" SIZE=20>
<INPUT TYPE=SUBMIT VALUE="Search">
</FORM>
```

Movable Type users can try the [MT-Search](#) plug-in. I am using this on an upcoming MT-powered site. I tried it on my main weblog (900+ entries) and it was fairly slow, but it seems to work well with smaller sites. It is no longer being actively developed, but it works well, even with the latest version of Movable Type. (Note: if you're using the MySQL version of Movable Type 2.2, [you need to tweak mt-search.cgi slightly](#).)

If you have your weblog on your own domain name, you can use a third-party search service that indexes your content and provides search results on demand. Popular alternatives on this front include [Atomz Express Search](#), which Blogger.com recommends, and [Google Free Web Search](#), which I use on my own weblog. Both are customizable to some degree, and quite fast, although their indexes of your content are not up-to-the-minute fresh. Both allow you to customize the look of your search box; my Google-powered search form looks like this (note the use of `<label>` and `accesskey`):

```
<form id="searchform" method="get"
action="http://www.google.com/custom">
<p id="searchlabel"><label for="q" accesskey="4">Search this
site:</label></p>
<p id="searchinput"><input type="text" id="q" name="q" size="18"
maxlength="255" value=" " /></p>
<p id="searchsubmit"><input type="submit" value="Search" />
<input type="hidden" name="cof"
value="LW:116;L:http://diveintomark.org/images/eyes.jpg;LH:68;AH:left;GL:0;S:ht
/>
<input type="hidden" name="domains" value="diveintomark.org" />
<input type="hidden" name="sitesearch" value="diveintomark.org" />
</p>
</form>
```

Note: you can't cut and paste this onto your own weblog. If you want to use Google Free Web Search, you'll need to sign up and get your own code that goes into that hidden `cof` field.

Further reading

- *Jay Allen*: [MT-Search](#), a search plug-in for Movable Type.
- *Phil Ringnalda*: [mt-search.cgi and MySQL](#). Important information getting MT-Search to work with Movable Type 2.2 and MySQL.
- [Atomz Express Search](#).
- [Google Free Web Search](#).
- *Blogger.com*: [How do I add a search engine to my blog?](#) Recommends Atomz, but links to others not listed here.
- *Jukka Korpela*: [Improving accessibility with accesskey in HTML forms and links](#). Near the end, it explains why to use "4" as the `accesskey` for your site search.
- *Jakob Nielsen*: [Is Navigation Useful?](#) Users often rely on search as their main hunting strategy.
- *Jakob Nielsen*: [Search Usability](#). Five years old and still incredibly relevant.
- *PHP.net*: [URL HOWTO](#). PHP.net has the most amazing site search I've ever seen. Beyond the standard search box, they use custom 404 ErrorDocuments (which would usually just throw a "page not found" error) to intelligently search the site based on the URL. So you can type an address like [php.net/phpinfo](#) into your browser's location bar, and php.net will redirect you to [the reference page on the phpinfo function](#).

Day 30: Creating an accessibility statement

If you've implemented any of the tips in this series, create an accessibility statement that lists the accessibility features of your site.

Who benefits?

1. **Bill** benefits. He makes extensive use of the keyboard for navigation, and he would use your [accesskey keyboard shortcuts](#) if he knew what they were. Unfortunately, Mozilla does not announce them or make them visible in any way, so you'll need to list them yourself.
2. Many other users benefit, by learning how to use their own software better. For instance, **Lillian** did not know that she could set her default text size in her browser until someone told her how to do it; now she can more easily read your site (that is, if you [use relative font sizes](#)). Netscape and Mozilla users may not know that they can turn on the Site Navigation Bar to take advantage of your [LINK-based navigation aids](#). ("View" menu, "Show/hide", "Site Navigation Bar", "Show Only AS Needed". Unfortunately, this feature was pulled from Mozilla 1.0 final at the last minute, but it's back in 1.1.)
3. Fellow web designers benefit, by learning more about accessibility and seeing that you've taken the time to implement these tips.

How to do it

In Radio or Manila, you can create your accessibility statement as a story ("Stories" in the Radio or Manila toolbar) so it has its own URL. Otherwise, just create a new post. Make a note of the URL after saving; you'll need it later.

The first thing you should do is list the [accesskey keyboard shortcuts](#) you've defined, since most browsers do not announce them or make them visible in any way. Also list the `accesskey` for your accessibility statement itself (which will be `accesskey="0"`, as we'll see in a minute).

After that, you should list all of the accessibility features that you've implemented throughout this series. If you implemented some tips but not others, just mention the ones you actually implemented. This will likely be an enlightening experience, like updating your résumé for the first time since taking a new job and realizing just how much stuff you've actually done. You've done a lot.

If you like, you can also mention briefly why you implemented each feature ("easier to use with screen readers", "for users of text-only browsers", "for users who can not distinguish colors", and so forth). Remember, an accessibility statement will be read by more than disabled users; it should be both informative (for users who directly benefit) and educative (for fellow web designers who choose to follow your example). Or, instead of explaining them yourself, you can link to individual days in this book, or just [link to the book's home page](#).

Finally, link to your accessibility statement from every page of your site, either in your site-wide navigation bar or in your page footer. Give the link a title and an `accesskey="0"`, like this:

```
<a href="/accessibility_statement.html" title="accessibility
features of this site" accesskey="0">Accessibility statement</a>
```

Further reading

- [Accessibility statement for diveintomark.org](#).
- [Accessibility statement for diveintoaccessibility.org](#). Feel free to use either of these as a template, including structure, wording, and links to further reading.

Conclusion

This has been "Dive Into Accessibility: 30 days to a more accessible web site". On behalf of [Jackie](#), [Michael](#), [Bill](#), [Lillian](#), and [Marcus](#), thank you for your attention.

Further reading: books that I recommend

1. *Joe Clark*: [Building Accessible Websites](#). I tech–edited this book; it's excellent. Comprehensive but not overwhelming.
2. *Jim Thatcher and others*: [Constructing Accessible Web Sites](#). Less comprehensive than Joe's book, but goes into greater depth in the topics it covers. Gives screenshots of how various screen readers and alternative browsers interpret various tags and markup. Also has a chapter on the current state of legal accessibility requirements.
3. *Steve Krug*: [Don't Make Me Think: A Common Sense Approach to Web Usability](#). Usability and accessibility overlap in many ways. Steve has lots of good advice about usable (and accessible) navigation.
4. *Owen Briggs, Steve Champeon, Eric Costello, Matthew Patterson*: [Cascading Style Sheets: Separating Content from Presentation](#). As we've seen throughout this series, CSS is an integral part of accessibility, because it allows you to "do the right thing" in your HTML markup (which assistive technologies care about) and still present your page the way you like in visual browsers.
5. *Eric Meyer*: [Eric Meyer on CSS: Mastering the Language of Web Design](#). There's also some free material on the book's [companion web site](#).

Accessibility statement

This is the official accessibility statement for [Dive Into Accessibility](#). If you have any questions or comments, feel free to email me at feedback@diveintoaccessibility.org.

Access keys

Most browsers support jumping to specific links by typing keys defined on the web site. On Windows, you can press **ALT** + an access key; on Macintosh, you can press **Control** + an access key.

All pages on this site define the following access keys:

- Access key 1 – Home page
- Access key 3 – Table of contents
- Access key 4 – Search
- Access key 8 – Terms of use
- Access key 9 – Feedback
- Access key 0 – Accessibility statement

Standards compliance

1. All pages on this site are [Bobby AAA approved](#), complying with [all the Bobby guidelines](#). This is always a judgement call; many accessibility features can be measured, but many can not. I have reviewed all the guidelines and believe that all these pages are in compliance.
2. All pages on this site is WCAG AAA approved, complying with [all priority 1, 2, and 3 guidelines](#) of the [W3C Web Content Accessibility Guidelines](#). Again, this is a judgement call; many guidelines are intentionally vague and can not be tested automatically. I have reviewed all the guidelines and believe that all these pages are in compliance.
3. All pages on this site are [Section 508 approved](#), complying with all of the U.S. Federal Government [Section 508 Guidelines](#). Again, a judgement call. I have reviewed all the guidelines and believe that all these pages are in compliance.
4. All pages on this site [validate as XHTML 1.0 Strict](#). This is not a judgement call; a program can determine with 100% accuracy whether a page is valid XHTML. For example, [check the home page for XHTML validity](#).
5. All pages on this site use structured semantic markup. H2 tags are used for main titles, H3 tags for subtitles. For example, on this page, JAWS users can skip to the next section within the accessibility statement by pressing **ALT+INSERT+3**.

Navigation aids

1. All pages have `rel=previous`, `next`, `up`, and `home` links to aid navigation in text-only browsers. Netscape 6 and Mozilla users can also take advantage of this feature by selecting the View menu, Show/Hide, Site Navigation Bar, Show Only As Needed (or Show Always).
2. The tips are cross-referenced in several ways. You can browse the tips [by person](#), [by disability](#), [by design principle](#), [by assistive technology](#), and [by publishing tool](#).
3. The home page and all archive pages include a search box (access key 4). Advanced search options are available at the [advanced search page](#).

Links

1. Many links have title attributes which describe the link in greater detail, unless the text of the link already fully describes the target (such as the headline of an article).
2. Links are written to make sense out of context.

Images

1. All content images used in this site include descriptive ALT attributes. Purely decorative graphics include null ALT attributes.
2. Complex images include LONGDESC attributes or inline descriptions to explain the significance of each image to non-visual readers.

Visual design

1. This site uses cascading style sheets for visual layout.
2. This site uses only relative font sizes, compatible with the user-specified "text size" option in visual browsers.
3. If your browser or browsing device does not support stylesheets at all, the content of each page is still readable.

Accessibility references

1. [W3 accessibility guidelines](#), which explains the reasons behind each guideline.
2. [W3 accessibility techniques](#), which explains how to implement each guideline.
3. [W3 accessibility checklist](#), a busy developer's guide to accessibility.
4. [U.S. Federal Government Section 508 accessibility guidelines](#).

Accessibility software

1. [JAWS](#), a screen reader for Windows. A time-limited, downloadable demo is available.
2. [Home Page Reader](#), a screen reader for Windows. A downloadable demo is available.
3. [Lynx](#), a free text-only web browser for blind users with refreshable Braille displays.
4. [Links](#), a free text-only web browser for visual users with low bandwidth.
5. [Opera](#), a visual browser with many accessibility-related features, including text zooming, user stylesheets, image toggle. A free downloadable version is available. Compatible with Windows, Macintosh, Linux, and several other operating systems.

Accessibility services

1. [Bobby](#), a free service to analyze web pages for compliance to accessibility guidelines. A full-featured commercial version is also available.
2. [HTML Validator](#), a free service for checking that web pages conform to published HTML standards.
3. [Web Page Backward Compatibility Viewer](#), a tool for viewing your web pages *without* a variety of modern browser features.
4. [Lynx Viewer](#), a free service for viewing what your web pages would look like in Lynx.

Related resources

1. [WebAIM](#), a non-profit organization dedicated to improving accessibility to online learning materials.
2. [Designing More Usable Web Sites](#), a large list of additional resources.

Accessibility books I recommend

1. *Joe Clark*: [Building Accessible Websites](#). I tech-edited this book; it's excellent. Comprehensive but not overwhelming.
2. *Jim Thatcher and others*: [Constructing Accessible Web Sites](#). Less comprehensive than Joe's book, but goes into greater depth in the topics it covers. Gives screenshots of how various screen readers and alternative browsers interpret various tags and markup. Also has an amazing chapter on the current state of legal accessibility requirements.

Terms of use

This book is free. You may use the information in this book for any purpose, commercial or otherwise, without fee or obligation of any kind.

Furthermore, permission is granted to copy, distribute, and/or modify this book under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included below.

GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA
02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time

you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front–Cover Text, and a passage of up to 25 words as a Back–Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front–Cover Text and one of Back–Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self–contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c) YEAR YOUR NAME. Permission is granted to copy,
distribute and/or modify this document under the terms of the GNU
Free Documentation License, Version 1.1 or any later version
published by the Free Software Foundation; with the Invariant
Sections being LIST THEIR TITLES, with the Front-Cover Texts being
LIST, and with the Back-Cover Texts being LIST. A copy of the
license is included in the section entitled "GNU Free Documentation
License".
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts".

being LIST"; likewise for Back–Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Translations

As specified in the [terms of use](#), translations of this book are permitted as long as they are published under the same free license as the original. You are encouraged to publish your translation on your own site. I also plan to host translations on the main *Dive Into Accessibility* site eventually.

If you translate *Dive Into Accessibility* into another language and wish to be listed here, contact me at translate@diveintoaccessibility.org. Note that you do not need to contact me asking permission to begin translating; you already have my permission.

One point which Karl Dubost brought up is that the character sketches used in this book are fairly American-centric. You are free to take some liberties with the characters, changing where they live, how much they pay for rent, even their names if you like. Just keep their disabilities intact, since they are referenced throughout the book.

Chinese

Both a [Traditional Chinese translation](#) and [Simplified Chinese translation](#) are available.

Estonian

Priit Laes (amd@tt.ee) is working on an Estonian translation.

Finnish

Visa Kopu (visa@visakopu.net) and Marjut Mutanen (marjut@iki.fi) are working on a Finnish translation.

French

Karl Dubost has published his [French translation](#).

German

Haiko Hebig (sfs@hebig.org) and several others are working on a German translation.

Italian

Roberto Scano and [Webaccessibile.org](#) team (info@webaccessibile.org) have completed [an Italian translation](#).

Norwegian

Jesro Christoffer Cena (chris.cena@blankspot.org) is working on a Norwegian translation.

Polish

Micha witkiewicz (michal@mimas.ceti.pl) is working on a Polish translation.

Romanian

Petru Paler (petru {at} paler . net) is working on a Romanian translation.

Spanish

Iván de Cincomonos (webmaster {at} cincomonos . com) and others are working on a Spanish translation.

Swedish

Marko Injac (marko . injac {at} mai – consulting . si) is working on a Swedish translation.