

Web Standards Solutions

The Markup and Style Handbook Special Edition

Dan Cederholm



Web Standards Solutions: The Markup and Style Handbook, Special Edition

Copyright © 2009 by Dan Cederholm

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-4302-1920-0

ISBN-13 (electronic): 978-1-4302-1921-7

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at <http://www.apress.com/info/bulksales>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editor Ben Renow-Clarke
Associate Production Director Kari Brooks-Copony

Technical Reviewer Matt Heerema
Production Editor Laura Cheu

Editorial Board Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Composer Lynn L’Heureux

Proofreader Lisa Hamilton

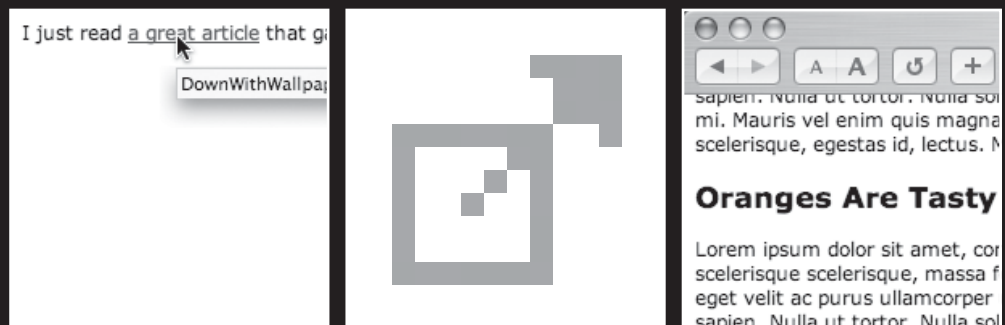
Indexer Broccoli Information Management

Project Manager Richard Dal Porto
Interior and Cover Designer Kurt Krames

Copy Editor Liz Welch
Manufacturing Director Tom Debolski

CHAPTER 7

ANCHORS



HTML links, or anchors as they are properly called, enable us to point not only to files, but also to specific sections of a page, and can be a handy way of “linking with precision,” narrowing the focus of a destination. In this chapter, we’ll take a look at the differences between four methods of anchoring, noting what benefits either method may gain. We’ll also look at the `title` attribute, and how it can improve a link’s accessibility, as well as the styling of links using CSS.

When pointing to a specific portion of a page, what is the best way to mark up an anchor?

It’s a common web design action—you’d like to link to a specific section of a web page, either within the current page that the user is on or within a separate page. You may choose to do this one of the four ways discussed in the following sections.

Let’s set up the examples by saying that our intention is to link to a particular heading within the same page.

Method A: An empty name

```
<p><a href=#oranges">About Oranges</a></p>
```

```
... some text here ...
```

```
<a name="oranges"></a>
<h2>Oranges Are Tasty</h2>
```

```
... more text here ...
```

Using an empty anchor element along with the name attribute to mark a specific point will probably look familiar to you. Placing the empty `<a>` element and closing `` just above the heading element and linking to it (using the `#` character, followed by the value matching the name attribute) will allow us to link to that specific portion of the page, which is especially helpful if the page consists of a long, scrolling list of items that we’d like to point to individually.

Figure 7-1 shows the results of clicking the “About Oranges” link, anchoring the page to where we’ve marked the ``, just above the heading.

It works great, although it’s a bit unsemantic to just waste an *empty* element as the marker. We can improve on that by taking a look at Method B.

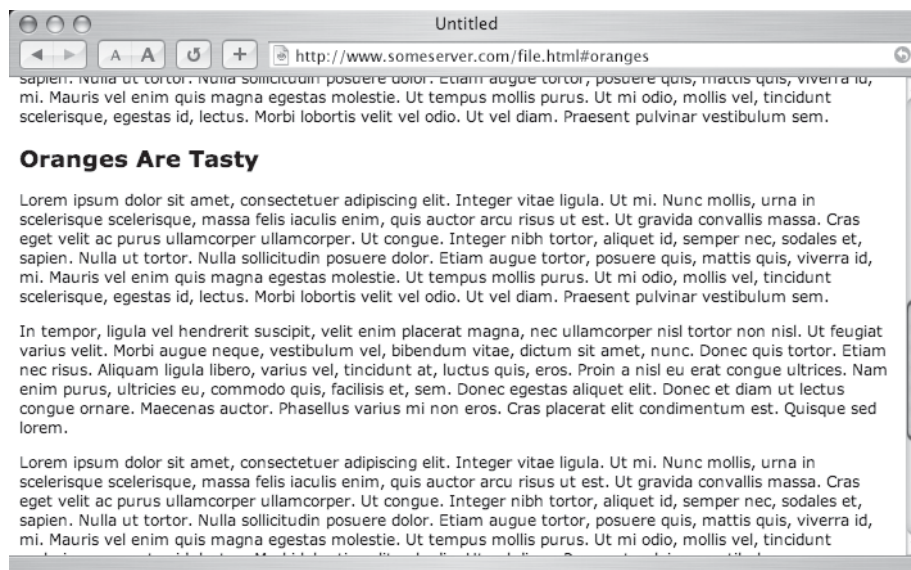


Figure 7-1. Demonstration of clicking a link to a named anchor

Method B: It's all in a name

```
<p><a href="#oranges">About Oranges</a></p>
```

... some text here ...

```
<h2><a name="oranges">Oranges Are Tasty</a></h2>
```

... more text here ...

As with Method A, we're still using the `<a>` element with the name attribute, yet this time we're wrapping it around the heading that we're intending to target. It does make a little more semantic sense this way; in Method A, we're giving meaning to... well, *nothing*, whereas in Method B, we're saying that not only is this a heading element, but it's also an anchored section of the page.

Beware of global `<a>` styling

One thing to watch out for if using Method B is that if you're setting a global CSS style for all `<a>` elements (color, size, decoration, etc.), that styling would override any styles you have for `<h2>` elements. This would occur because the `<a>` element, in this example, is a child element and sits inside the `<h2>` that surrounds it all.

For instance, if in your CSS you had a declaration that went something like this:

```
a {
  color: green;
  font-weight: bold;
  text-decoration: underline;
}
```

using Method B along with the preceding CSS would result in the heading being green, bold, and underlined along with any other <a> elements on the page—perhaps different from the way you’d like <h2> elements to be styled.

We can avoid this (and gain some other benefits) by using the :link pseudo-class for <a> elements, which we’ll go over in detail in the “Extra credit” section, later in this chapter.

Richer name attribute

One upside to using Method B, or Method A for that matter, is that the name attribute has the ability to handle richer anchor names—more specifically, the ability to use character entities in the names.

For instance, if using Method B, you could do this (where the entity é represents the “e”):

```
<p><a href="#resum&#233;">My Resum&#233;</a></p>
... some text here ...

<h2><a name="resum&#233;">Dan's Resum&#233;</a></h2>
... more text here ...
```

This becomes more important when dealing with foreign languages and the characters that stray from the English alphabet.

But there are a few more methods to investigate—the first of which eliminates completely the need for using the <a> element to set the anchor point. Let’s take a close look at Method C.

Method C: Lose the name

```
<p><a href="#oranges">About Oranges</a></p>
... some text here ...

<h2 id="oranges">Oranges Are Tasty</h2>
... more text here ...
```

Aha! The `id` attribute acts just like the `name` attribute, in that it also sets an anchor point on the page. In addition, by using Method C, we can eliminate the need for the extra `<a>` element that's necessary when going with the `name` attribute in Methods A and B. We're cutting down on code, and that of course is always a good thing.

Because the `id` attribute can be added to any element, we can easily anchor anything we'd like on the page. In this case, we're choosing to anchor the heading, but we could also just as easily anchor a `<div>`, `<form>`, `<p>`, ``—and the list goes on.

Two birds with one stone

Another benefit to using Method C lies in the fact that, many times, we can utilize a pre-existing `id` attribute that we've added for the purposes of style or scripting. Because of this, we'll eliminate the need to include additional code in order to set the anchor point.

For instance, let's say you had a comments form at the bottom of a long page that you'd like to link to nearer the top. This form had an `id="comments"` already in place, particularly for purposes of styling it uniquely. We could link to that `id` as an anchor—without having to insert an `<a>` element with the `name` attribute.

The code would look something like this:

```
<p><a href="#comments">Add a Comment!</a></p>

... a lot of text here ...

<form id="comments" action="/path/to/script">
... form elements here ...
</form>
```

Also, if your page requires a long scroll, you could make it easy for users to get “back to top” by adding a link at the bottom that refers to a top-level element's `id` (for instance, a logo or header).

It's a good idea to point out that, while it's the most obvious choice, it's best to avoid using the name “top” when anchoring. There are some browsers that have that particular name reserved and using it can cause mixed results. It's best to choose something similar, knowing that it won't cause problems. #genesis perhaps? #utmost? You get the idea.

Older browsers and the `id` attribute

An important downside to mention when using only the `id` attribute for anchors is that some *older* browsers don't recognize them. Ouch. This is certainly something to consider when marking up your own anchors, and an unfortunate case for backward compatibility. Let's take a look at the final example, Method D.

Method D: The all-in-one

```
<p><a href="#oranges">About Oranges</a></p>
```

... some text here ...

```
<h2><a id="oranges" name="oranges">Oranges Are Tasty</a></h2>
```

... more text here ...

If both forward and backward compatibility are the most important points for you when building anchors, then this method should please everyone. Older and newer browsers alike will recognize the named anchor element—but because the `name` attribute is deprecated by the W3C in the XHTML 1.0 recommendation (http://www.w3.org/TR/xhtml11/#C_8), you're covered for the future using the `id` attribute as well.

As with Method B, we'll have to beware of any global styling that may be done on the `<a>` element itself.

Sharing names

If choosing to go with Method D, it's perfectly acceptable (and probably convenient) to use the same value for both the `id` and `name` attributes—but only when they are contained in *a single element*. Furthermore, this is allowed only within a few *certain* elements: `<a>`, `<applet>`, `<form>`, `<frame>`, `<iframe>`, ``, and `<map>`, to be exact. For this reason, we've moved the `id="oranges"` from the `<h2>` element to the anchor within.

Now that we've looked at four different methods to create anchors, let's summarize what each has to offer.

Summary

For this chapter, there may not be a real clear-cut winner here, although two methods rise above the others (C and D), each having its own pros and cons. Let's recap the facts regarding each:

Method A:

- This method should work across *most* browsers.
- As an empty element, it provides no structure or meaning to the markup.
- This method requires extra markup.
- With the `name` attribute deprecated in XHTML 1.0, forward compatibility should be a concern.

Method B:

- This method should work across all browsers.
- You must be conscious of any global `<a>` element styling that could override outer element styles.
- This method requires extra markup.
- With the `name` attribute being deprecated in XHTML 1.0, forward compatibility can be a concern.

Method C:

- This method entails less markup.
- You have the option of using an existing ID.
- This method ensures forward compatibility.
- This method requires a reasonably modern browser.

Method D:

- This method is both forward and backward compatible.
- You must be conscious of any global `<a>` element styling that could override outer element styles.
- This method requires extra markup.

It appears that Methods C and D are the better choices, where forward compatibility and less markup are pitched against more markup and full compatibility. My suggestion is to take into account the target audience and make an informed decision based on this.

For instance, if you're building a web-based application or intranet that you know will require a recent browser version, then going with Method C would most likely be best. Less markup is required—but this is known not to work in some version 4.x browsers. Check your site statistics to see if there is a substantial audience still using outdated ancient browsers.

Alternatively, if you're building a site that could be viewed by anyone, anytime, you may opt to go with Method D, which will ensure backward *and* forward compatibility—with the extra baggage of the anchor element added.

It's your choice, and hopefully by looking at each, you can make the right decision at the right time out in the real world.

Extra credit

For this extra credit session, we'll take a look at more things anchor related—specifically the benefits of using the `title` attribute as well as styling anchor links with CSS.

The title attribute

While earlier we were talking explicitly about anchors for creating page sections, let's shift gears slightly and talk about anchor links in general—when pointing to other destinations.

As an added accessibility feature, adding the title attribute to anchor links can provide a richer and more specific description of the destination that you're pointing the user to. With this added information, it becomes clearer to users as to where they are going—and they need not base their decision for clicking a link solely on just the text or image that is being anchored.

How does this added information become available to the user? We'll find that out next.

Title in action

Let's take a look at the title attribute in action. We'll mark up an ordinary hyperlink like this:

```
I just read <a href=http://www.downwithwallpaper.com/tips.html
title="How to Take Down Wallpaper">a great article</a> that gave me a
few home improvement tips.
```

Although the text is intentionally a bit vague in this example, the title attribute gives us an additional nugget of information about the link—in this case, the actual title of the article that's being pointed to.

Another common practice when inserting title attributes is to simply use the page's <title> (which is usually displayed in the browser's title bar). This, of course, should only be used if the title bar's text makes sense—ideally that includes both the site's title as well as the page-specific title.

For instance, let's say that for the preceding example, the page's title was "DownWithWallpaper.com | How to Take Down Wallpaper." Besides potentially being the *only* article necessary for the site, it could be used in the title attribute of our example as follows:

```
I just read <a href=http://www.downwithwallpaper.com/tips.html
title="DownWithWallpaper.com | How to Take Down Wallpaper">a great
article</a> that gave me a few home improvement tips.
```

We now have a richer description of what's being linked to. But how do users receive the information contained within the title attribute?

Tooltip titles

Most modern browsers support the title attribute by turning the value into a "tooltip"—a small colored box that pops up when the mouse is hovered over the link. Visually, it'll give users that extra useful bit of information just before they click the link. This has obvious benefits in letting users know *exactly where they're going*.

Figure 7-2 shows our example in a browser, with the tooltip exposed by the hovering mouse.



Figure 7-2. An example with the title attribute revealed by the mouse hover

Titles are spoken

Another benefit to adding title attributes to links is that screen readers will read out the value along with the linked text. Sighted and nonsighted users alike will gain better understanding of the destinations you're taking them to, and that is certainly a good thing.

Styling links

Earlier in the chapter, I'd mentioned "beware of global link styling"—that there was a way to avoid unintentional styling of named anchor elements and instead narrow our focus to hyperlinks that use the href attribute only.

Gone are the days of defining link colors in the HTML of a document. We can separate those design details from the markup by using the pseudo-classes `:link`, `:visited`, `:active`, and `:hover` to uniquely style *hyperlinks* in a variety of ways.

Let's take a look at a few different CSS styles that we can apply to normal, everyday links:

```
a:link {
  color: green;
  text-decoration: none;
  font-weight: bold;
}
```

Quite simply, the preceding declaration will make all anchor elements that use the href attribute green, bold, and *not* underlined.

Instead of `text-decoration: none;`, we could've said `underline` (the default), `overline` (for the rebels out there), or a combination of the two, as shown here:

```
a:link {
  color: green;
  text-decoration: underline overline;
  font-weight: bold;
}
```

Figure 7-3 shows how the `underline overline` combination would appear in a typical browser. Sort of unconventional—but possible!

I just read [a great article](#) that gave me a few home improvement tips.

Figure 7-3. A link example with underline-overline text decoration

Backgrounds

The possibilities for uniquely styling links are just about endless. Most CSS rules that we've applied to other elements are available for anchors as well. For instance, we can apply background colors to links and/or even background images as well—perhaps a small icon, aligned to the left or right of the link text, as shown in Figure 7-4.


I just read [a great article](#)  that gave me a few home improvement tips.

Figure 7-4. A link with a right-aligned icon as a background image

The CSS needed for achieving Figure 7-4 goes something like this:

```
a:link {
  padding-right: 15px;
  background: url(link_icon.gif) no-repeat center right;
}
```

We're setting the icon to align center (vertically) and to the right of the link text. Extra padding is added to the right side as well, to allow the icon to show through without any overlapping of text.

Dotted borders

Tired of the plain, solid underlines of links that we've been seeing for years now? By using the dotted or dashed value of the border property, we can create... you guessed it, dotted or dashed link borders.

First, we'll need to turn off default underlining with the text-decoration property to get it out of the way, and then we'll add a 1-pixel border-bottom that's both dotted and green.

```
a:link {
  color: green;
  text-decoration: none;
  border-bottom: 1px dotted green;
}
```

It's important to note that, if you'd like the dotted border to be the same color as your link text, you'll need to declare that color in the border-bottom property as well. The results can be seen in Figure 7-5.

I just read [a great article](#) that gave me a few home improvement tips.

Figure 7-5. A link with a dotted border

Using the preceding method, you could also mix and match border colors, giving your link text one (with the `color` property) and your border another (with the `border-bottom` property). Furthermore, you could use the `solid` or `dashed` value for the `border-bottom` property.

Internet Explorer for Windows gets the dotted property a bit wrong when using a 1-pixel width. Using 1px as the value for a dotted border ends up looking exactly like the “dashed” style border. Fear not; it’s just a small glitch.

Where you been?

Don’t forget to add an `a:visited` declaration to help users see where they’ve been before. All the usual CSS rules can be applied to this pseudo-class as well, giving visited links their own unique style, with a different color, border, background, and so forth.

The CSS goes like this:

```
a:visited {
  color: purple;
}
```

At the very least, the preceding declaration will alert users that they’ve visited a link by changing its color to purple. It’s very important to make even just a slight change, as the preceding one.

Hovering

Similarly, we can use the `:hover` pseudo-class to add powerful effects to links when they’re hovered over by the mouse. This could be a color change or an addition of a border, background, or icon. The possibilities are endless.

```
a:link {
  color: green;
  text-decoration: none;
  border-bottom: 1px dotted green;
}

a:hover {
  color: blue;
  border-bottom: 1px solid blue;
}
```

The two preceding declarations give us links that are green with a dotted border, but then on hovering, the link turns blue, with a solid bottom border (also blue).

This is just one example of a simple hover effect. You can imagine that by trying different combinations of CSS rules on both links and hovered links, you can start to design sophisticated mouseover effects without the need for JavaScript or extra markup.

Active state

The `:active` pseudo-class handles the state of a link when the mouse button is clicked. The same rules can be applied here—changing the color, text decoration, background, and so on. For instance, if you had the link turn red when the mouse button is clicked, this could be an extra visual cue to users that they've chosen to head to that particular destination, and have indeed clicked.

The following declaration does just that:

```
a:active {
  color: red;
}
```

LoVe/HAtE your links

Ordering the four pseudo-classes mentioned becomes important in order for all of them to behave properly—without one overriding the other.

LoVe/HAtE is a handy way to remember the correct order to place your declarations (www.mezzoblue.com/css/cribsheet/):

- a:link (L)
- a:visited (V)
- a:hover (H)
- a:active (A)

You could make up your own abbreviation for this—whatever it takes to help you remember. Love Vegetables? Have Asparagus!

To demonstrate, here are four of the preceding examples, assembled in the right order, as a complete package:

```
a:link {
  color: green;
  text-decoration: none;
  border-bottom: 1px dotted green;
}
```

```
a:visited {
  color: purple;
}

a:hover {
  color: blue;
  border-bottom: 1px solid blue;
}

a:active {
  color: red;
}
```

Fitts' Law

We can keep Fitts' Law in mind as it relates to hyperlinks and increased usability. Fitts, an American psychologist whose work is often cited by interaction design experts, says

The time to acquire a target is a function of the distance to and size of the target.

—Paul Fitts (<http://www.asktog.com/basics/firstPrinciples.html#fitts's%20law>)

In other words, the larger the link's target is, the quicker and easier it is to use. One easy way we can apply Fitts' Law is applying `display: block;` to links where appropriate. This way, not only is the link text clickable, but the space around the link is as well.

Take a simple unordered list of links, for example, as Figure 7-6 illustrates. By applying `display: block;` and a smidgen of padding to the `<a>` elements, we can increase the target area of the link, making it easier to scan and click the list by selecting the surrounding area (the entire row) as well as the text (see Figure 7-7). Adding a `background-color` to the hover state is helpful in providing feedback to the user as to the increased hotspot available.

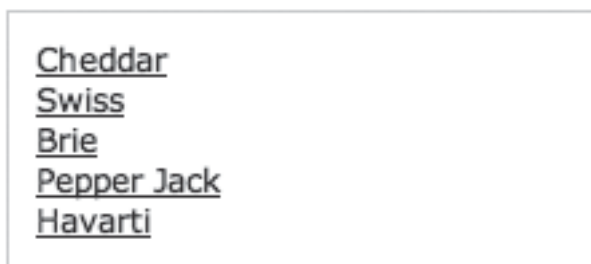


Figure 7-6. An unordered list of links

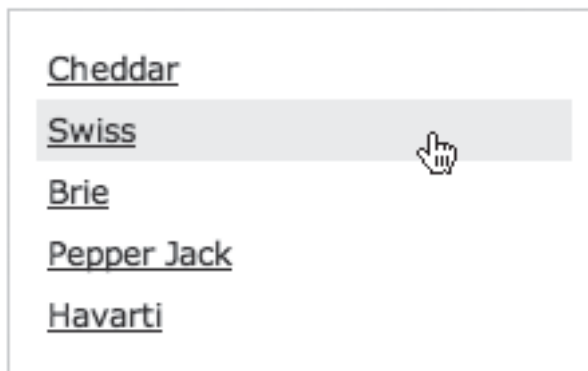


Figure 7-7. A block-level link; we've added a background-color to the hover state.

Here's the CSS that would enable that to happen:

```
ul li a {
  display: block;
  padding: 4px;
}
ul li a:hover {
  background-color: #eee;
}
```

A hack for IE6

Often, Internet Explorer version 6 adds extra vertical space to hyperlinks that are set as block-level elements. Boo. But there is a little fix for that, should you run into any rendering issues. We'll target IE6 specifically by prefacing the declaration with the `* html` hack. Only IE6 will read this declaration, and it'll be ignored by other browsers:

```
* html ul li a {
  height: 1%;
}
```

That magic hack will make block-level links look correct in IE6. Dubbed the “Holly Hack” after its author, Holly Bergevin, `height: 1%` also fixes a plethora of related IE bugs. For more info on why it works, check out “*On having layout*” (<http://www.satzansatz.de/cssd/onhavinglayout.html>). Warning: it's not exactly light reading!

See also Dunstan Orchard's "Link Presentation and Fitts' Law" (<http://www.1976design.com/blog/archive/2004/09/07/link-presentation-fitts-law/>) or Dave Shea's article (http://www.mezzoblue.com/archives/2004/08/19/fitts_law/) for more information on block-level link styling.

Anchors aweigh

Before we set sail to the next chapter, let's review what we've discussed. We looked at four different ways to create anchors on a page—the last two of which we thought were more optimal. Depending on your audience, you now have the knowledge you need to make a decision on your next project.

We then talked about the `title` attribute and how it can improve usability by giving the user extra information about a link's destination. Visual readers and nonsighted listeners will both be able to take advantage of the `title` attribute's additional info.

Lastly, we looked at the styling of links using CSS's pseudo-classes. With an imagination, and a few declarations, rich, interactive effects can be achieved using zero JavaScript and no extra markup by targeting different CSS to the four different link states.

Now it's time to furl our sails and raise the anchor to the gunwale! For it's... sorry, got carried away there.