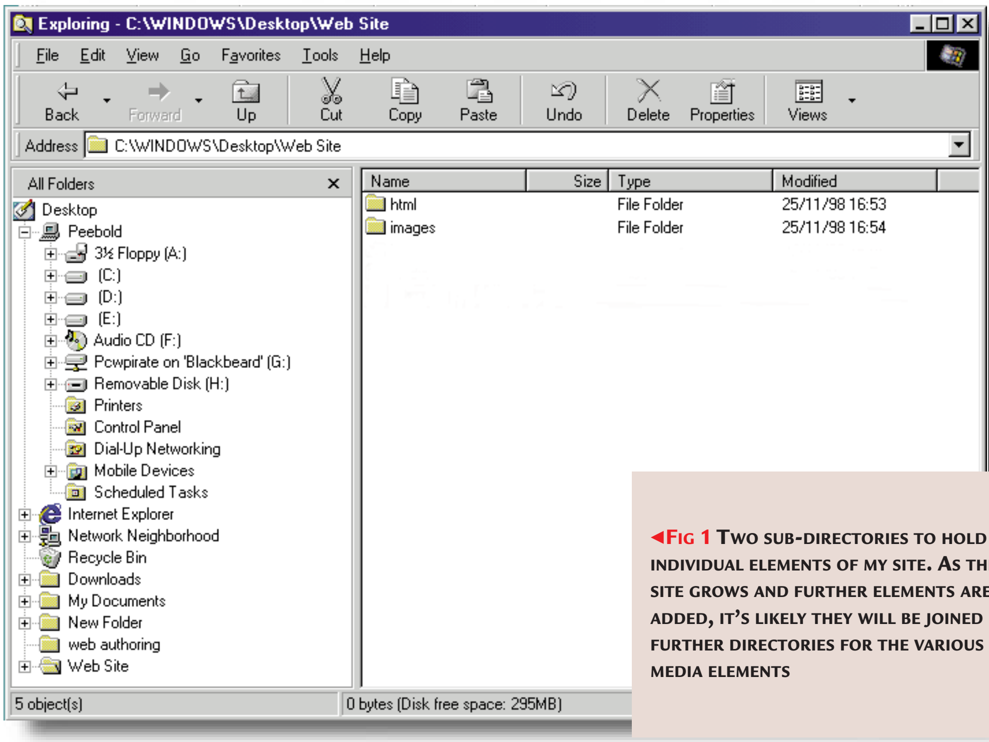
Weaving a web

Design and build your own web site. Nik Rawlinson shows you how to start the project.


Exploring - C:\WINDOWS\Desktop\Web Site

Address C:\WINDOWS\Desktop\Web Site

```
Name      Size  Type          Modified
html            File Folder   25/11/98 16:53
images          File Folder   25/11/98 16:54
```

5 object(s)   0 bytes (Disk free space: 295MB)

◀FIG 1 TWO SUB-DIRECTORIES TO HOLD INDIVIDUAL ELEMENTS OF MY SITE. AS THE SITE GROWS AND FURTHER ELEMENTS ARE ADDED, IT'S LIKELY THEY WILL BE JOINED BY FURTHER DIRECTORIES FOR THE VARIOUS MEDIA ELEMENTS

Designing your own web site from scratch, coding it by hand and seeing successful results can be a rewarding experience. It's useful, too, as it's cheaper than investing in a dedicated authoring package and means you'll be able to keep an eye on what's happening. You'll build streamlined pages which, theoretically, will look exactly the way you want. This month, we'll look at designing your layout and run through the basic HTML code words you'll need to get started.

Your chosen subject

You'll probably have a clear idea of the subject around which you'll be basing your project. Whether it's a personal home site, a guide to your home town or a business site showing your products to the world, a few common principles apply. For the sake of illustration, this series will be centred on creating a personal web site, although the tags and methodology used here will apply to almost any other site you may have in mind. All files created are on our cover CD, and you can see the site as it grows online by visiting http://i.am/pcw/. To keep things tidy, we'll put our images and pages in separate directories. Other media, such as sound files, Java applets or proprietary animations generated with Macromedia Flash and similar packages, will be segregated to make them easy to find as the site grows, and allow us to expand the site at a later date without having to search for files, especially as names may by then have been forgotten [Fig 1].

Firstly, we need to decide if our site will use frames. These are a useful method of subdividing the screen and keeping elements such as the title and menu visible while the content of the pages changes in the primary frame. The main benefit is that although setting up the frames may initially be a little tricky, it will save time in the long run. You'll no longer have to place navigation buttons and replicate your menu on each page. Instead, the site's main menu can remain visible at all times, dramatically reducing your workload.

Many older browsers cannot support frames and so designers rightly argue that to cater for the widest possible audience it's best to avoid them altogether. A more sensible alternative is to offer a non-frames option that will lead users to a version of the site along the lines of a traditional flat-form layout.
With this in mind, we'll create a frame-based site. If your browser doesn't support frames, the latest versions of Netscape's and Microsoft's browsers can be found on our cover-mounted CD.
The frame structure of the site is set up with the code in Fig 2, allowing for a screen-wide title bar, 50 pixels deep, straddling two columns. The leftmost vertical column, 130 pixels wide, will host our site menu. To the right will be the main frame, occupying the remainder of the screen, which will be the area where our pages are displayed, and it is thus wider than the menu frame.
You'll notice that each is named. These names will be used to direct menu options to the appropriate destination.

Index linked

This page will be the first thing to confront visitors to the site, so we'll call it index.htm, saving it in the site's root directory. The reason for this is that many ISPs direct browsers to automatically search for a file called index.htm or index.html in any directory to which a browser is pointed if visitors don't specify a destination file.
This is the only page we shall not be saving in our site sub-directories.
You will notice this initial page has no actual content *per se*. It just defines a frame-bound structure into which the

[FIG 2] Setting the frame structure

```
<HTML>
<HEAD>
  <TITLE>Nik's HomePage</TITLE>
</HEAD>
<FRAMESET FRAMEBORDER=0 BORDER=0 FRAMESPACING=0 ROWS="70,*">
  <FRAME SRC="html/banner.htm" NAME="bannerframe" SCROLLING=NO>
<FRAMESET FRAMEBORDER=0 BORDER=0 FRAMESPACING=0 COLS="130,*">
  <FRAME SRC="html/menu.htm" NAME="menuframe" SCROLLING=NO>
  <FRAME SRC="html/welcome.htm" NAME="contentframe">
</FRAMESET>
<NOFRAMES>
<BODY>
This page uses frames and requires a compatible browser which
yours, I'm afraid, is not.
</BODY>
</NOFRAMES>
</FRAMESET>
</HTML>
```

browser will load the pages of the site. The code in Fig 2 will load three further pages: banner.htm will be loaded into the frame we've called bannerframe, menu.htm will be loaded into menuframe, and welcome.htm will load into content frame.

On the right lines

Let's look at a couple of the key lines of this structure in detail.

First, the line

```
<FRAMESET FRAMEBORDER=0
BORDER=0 FRAMESPACING=0
ROWS="50,*">
```

generates the first frame. It removes any borders (BORDER=0) so the edges are effectively invisible to any user with a modern browser. Similarly, FRAMEBORDER=0 and FRAMESPACING=0 remove any padding, allowing page elements within the frame to stray up to the edges.

The ROWS tag tells the browser we want to split the screen horizontally. Had we used COLS, as in Fig 2, two lines below, the screen would have been split vertically instead. The '50,*' defines the frame size. It refers to a depth of 50 pixels, and the asterisk tells the browser that the frame below should fill what remains of the screen. Had we stated '50,*,70' we would have had a frame of 50 pixels depth at the top of the screen, another of 70 pixels depth at the foot, and the remaining centre portion, whatever size that would be, using any space remaining between the two. So, our frames will always exactly fit within the browser windows no matter how large or small the site visitor makes it.

Second, the line

```
<FRAME SRC="html/banner.htm"
NAME="bannerframe"
SCROLLING=NO>
```

defines the frames' content. The source (SRC=) is the file banner.htm, found in the html directory of our site, while the frame's name is bannerframe. Each frame in a set must be given a unique name. When we start to build our menu we'll use these frame names to direct the hyper-linked pages to specific locations on our screen, rather than opening in the frame where the link was clicked.

Finally, on that line, we've specified that the user shouldn't be allowed to scroll the contents of the frame. If its contents are too big to display, they'll be cropped at the border.

Now to put some content into our frames. We'll start with the banner. The banner runs across the width of our site and won't change as users move around it. So, we'll use it to display a site logo. Although there are a number of specialised web graphics creation packages such as Adobe's ImageStyler or Macromedia's Fireworks, you can use regular imaging software. Our cover CD includes a copy of PaintShop Pro 5 which will prove to be more than adequate.

Screen resolution is generally around 72dpi so it's advisable to create any images at this resolution. Upping the resolution will improve the quality but it will be a disproportionately longer download for site users. It's also a good tip to reduce the number of colours used in the image — images with only 256 colours will load a lot faster than those with 16.7 million.

Many graphics packages have a 'web safe' palette of colours which a browser should be able to display without trouble on either a Mac or PC platform. If your graphics package incorporates this facility, it's worth putting it to use.

Create a new image of this resolution and make a note of the size. If the height is larger than that of your bannerframe, you'll need to either make the image smaller or increase the frame height.

Many graphics packages have 'web-safe' colours

As the frame definition for bannerframe specifically states the frame should not scroll, an oversized image will be cropped at the borders. We now need to put the banner image onto our site.

Once it's saved in the images directory as banner.jpg, we'll write a one-line web page that points to it [Fig 3]. The height and width attributes define the size of our image. If we didn't

[FIG 3] Pointing out the banner

```
<img src="images/banner.jpg" align=right height=50 width=300 alt="Nik's HomePage">
```

[FIG 4] Virtual rectangles

```
<Html>
<Head>
<Title>
The Imagemap
</Title>
</Head>
<Body BGcolor="white">
<Img Src="images/imagemap.jpg" Border="0" Height="300" Width="130" Name="imagemap_menu" UseMap="#imagemap_menu" ↙
Alt="This is an imagemap - please enable images on your browser">
<Map Name="imagemap_menu">
<Area Shape=Rect Coords="13,232,103,262" Href="html/contact.htm" target=contentframe>
<Area Shape=Rect Coords="55,185,120,215" Href="html/noise.htm" target=contentframe>
<Area Shape=Rect Coords="6,93,82,123" Href="html/words.htm" target=contentframe>
<Area Shape=Rect Coords="44,41,116,71" Href="html/about.htm" target=contentframe>
</Map>
</Body>
</Html>
```

(Key: ↙ code continued on line below)

include them, a browser would simply leave the image as it was after loading it. But, if the image were not loaded, it would not be able to produce a blank box of the same size. Should this happen, our layout would be lost. The 'alt' tag, meanwhile, is used by text-only browsers or those with speech synthesis for the sight-impaired. It too will either display while the image is loading, or if the image fails to load, leave the page in a usable condition even if it has been only partially downloaded. This line of code is saved in the html directory as banner.htm. Reloading our original index.htm, you'll see it now simultaneously loads banner.htm and our image, and places it into the top frame.

Seeing successful results can be a rewarding experience

On the menu

The menu will run vertically down the left-hand side of the screen. Like the banner, this will be a permanent resident on the screen. It won't scroll, as we've disabled the scrolling attribute in that frame. We'll set each menu option so that rather than opening in the default location (which would be menuframe, as that's where we're clicking the hyper-link) it will open any associated pages in contentframe.

Two options are available to us in generating our menu. The easiest is to create a series of text-based links. In these days of fast modems and unlimited (well, 20Mb at least) web space, that's a little boring. A far more interesting option would be to generate an image map — a picture with hot spots which link the pages of your site. In the frame structure we've defined, the menu frame is fairly slim; just 130 pixels wide. To avoid changing this we'll have to design our menu to fit within these constraints. As with the banner image, we'll stick to 72dpi and reduce the colours used to a web-safe palette.

The site will incorporate four main sections: 'about', 'words', 'noise' and 'contact'. Each is accessed from the image map. Before the hot spots can be defined though, the map needs to be drawn. It's then time to mark the co-ordinates of your hot spots. Using the sample image accompanying this workshop (*on our cover CD*) we'll define four virtual rectangles to encapsulate each of the words in the image.

On location

To see how this works, load the image into any graphics package displaying the co-ordinates of your cursor location and check that the vectors, in Fig 4, define the opposite corners of boxes around each word. In Fig 4, the browser loads the image imagemap.jpg from the images directory. We name this imagemap_menu so that the browser can locate it as you run your mouse across its surface. Next, it's defining each hyperlink. This is best explained by looking at one link in detail:

```
<Area Shape=Rect
Coords="55,185,120,215"
Href="html/noise.htm"
target=contentframe>
```

In this line, we tell the browser that the hyperlink is a rectangular section of the image. The upper left-hand corner of this link is 55 pixels in and 185 pixels down the graphic we've called imagemap_menu.

The lower right-hand corner of the rectangle, meanwhile, is at the co-ordinate 120, 215. Therefore, if the user clicks anywhere within this virtual rectangle, the browser will open the file noise.htm, found in the html directory, in the frame we've called contentframe.

Expecting visitors

So now we've defined the three permanent parts of our site: the banner and menu, both of which will remain visible at all times, and the frame structure that binds the content of our site into a set layout. Next month, we'll deal with attracting visitors to your site.

PCW CONTACTS

Nik Rawlinson can be contacted via the PCW editorial office (address, p10) or by emailing nik_rawlinson@vnu.co.uk