

Programming Tools | Digital Media Programming | Ben Dunkle, Professor

Programming and coding are what we'll be doing in the class this semester. It's important to understand the difference; programs are text files that are actually executed by a computer. They can run as independent applications, rather than within another application. Code (like html) is actually PART OF text files that contain descriptions of pieces of information. Code must be interpreted by a browser or other application that will then display the information in an organized way.

Programming and coding can be done within text editors and WYSIWYG editors.

What are text editors?

Text editors are applications that are well suited to the types of text manipulation that programming requires. They are make it easier to than, for instance, a word processor like MS Word to open all kinds of text files, organize the content in a way that is intuitive for programmers, and save the files as compact and in as universal a format as possible.

BBEdit and other text editors

This class will use BBEdition Lite as our basic text editor. BBEdition Lite is a robust, full-featured text editor that has the added advantage of being free. One drawback to using BBEdition Lite is that the code is not colorized. Creatext, on the other hand, displays tags, comments, and scripts in different colors, making things a bit easier on the eye-and it's free. There are dozens of editors for PC-the most popular being Notepad and Homesite.

What are WYSIWYG editors?

WYSIWYG is an acronym for "what you see is what you get". A WYSIWYG (pronounced "wiz-ee-wig") editor or program is one that allows a designer to create a variety of documents – graphical user interfaces (GUI), web pages, and even print documents are actually lines and lines of code describing how information should be displayed. This way, the designer can see what the end result will look like while the document is being created. A WYSIWYG editor can be contrasted with text editors, which require the developer to enter descriptive codes (or markup) and do not permit an immediate way to see the results of the markup.

WYSIWYG programs, such as DreamWeaver, RealBasic, and Quark XPress, conceal the markup and allow the developer to think entirely in terms of how the content should appear. One of the trade-offs, however, is that a WYSIWYG editor does not always make it easy to fine-tune its results.

In a nutshell, WYSIWYG editors are easy to master, make it easy to control the look and feel of the file you're working on, but have several flaws that make understanding the code they produce crucial.

Proprietary languages such as Actionscript and Lingo are written within the applications that utilize them...

...which are Flash and Director, respectively. Unfortunately, text editors like BBEdition are currently unable to work effectively with Flash and Director, but fortunately, Flash and Director, like DW, have powerful scripters built in. By proprietary, I mean languages which only apply to individual applications (you can't put ActionScript on a web page)..

Working with text

When programming, the most common thing you'll do is NOT TO TYPE TEXT, but to move text around from other sources. Cut, Copy, Paste—these three things are your bread, butter, and butter knife.

To cut, copy and paste on a Mac:

⌘ X (or F2) — Cut ⌘ C (or F3) — Copy ⌘ V (or F4) — Paste

To cut, copy and paste on Windows:

Control X - Cut Control C - Copy Control V - Paste

You'll be cutting, copying, and pasting text from various sources, including:

- Your own files - Most text files we use have tons of repeated and repeatable information. Rather than type it over and over, it makes sense to type once, copy, and paste as needed.
- Other people's files - No, it's not stealing (don't quote me, you may know otherwise after taking the Digital Media Law and Ethics class) and absolutely moral to use other people's code (it's a nice gesture to mention the author in your code, however).

To view other people's code (fig. 1.1):

You can highlight, copy and paste any text you want into your own files, and it will work just the same way it did on the page you took it from.

Downloaded files

Many of the support sites we will use offer files that we can download, open on our own machines and take the code out of. These are usually in .zip, .sit, or .tar format. They must be decompressed by an application like Stuffit or WinZip.

Naming files

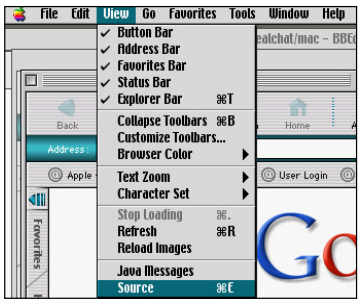
We'll be saving, naming, and renaming tons of files in this class. Keep the following in mind:

Don't do the following when you name files:

- use spaces in the file names. If you must have a separator between words in a file name, use "-" (option dash, or underscore), for instance "summer_photos.html". Alternatively, you may separate words by capitalizing all but the first word in the file name (like summerPhotos.html)
- use capitals, slashes, dashes, or special characters (!@#%&*[{}]) etc.).

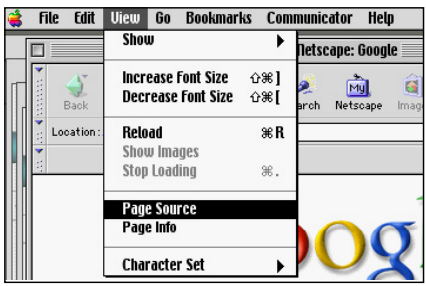
Do the following when you name files:

- try to make the name something intuitive and descriptive, like "dma207_projects.html", rather than something obscure like


fig. 1.1


On Internet Explorer

Click on View>Source, or hit ⌘ E (control E on windows)



On Netscape

Click on View>Page Source



and you'll be able to look at the code that controls the page, as well as highlight and copy it.

"projects.html". Finding the file a year or two down the road will be much easier.

- keep filenames under 30 characters or so.

Testing files

After copy and pasting, the next most common thing you'll do is test your files to see if they work. How we test our files will depend on the type of coding we're doing, but the majority of what we do is testable on a browser, like IE or NN.

After we've worked on a file, we save it and then open it from within a browser. To do this, make sure the file is in an easy-to-find place, like a well-named folder on your zip, or the desktop of your computer, etc. Then, from your browser, go to File>Open, locate the file, and see if it works. If it does, great; if it doesn't, DON'T CLOSE THE BROWSER WINDOW-go back to your text-editor, make any changes, save the file, return to the browser, and hit Refresh on IE or Reload on NN.

Testing files directly

There will also be times when we want to test our files that are located on an web page server. To do this, we will use our GORT accounts and Fetch, a Macintosh-based ftp (file transfer protocol) program (fig. 1.3).

Host name is where your site is located. For example, if your isp (internet service provider) is aol, the host name is probably aol.com.

User ID is the name you sign on to your account with.

Password is, uh, your password.

Directory You can usually leave this field blank, and then navigate through the server's window to find where to put your files.

After you connect, you'll get a window that looks like this (fig. 1.4). This is the server where my website is hosted. All of the files that are viewable over the internet are in the WWW folder*. If you double-click this folder, you'll see a list of the files on your Canisius site, just like if you double-click a folder in your hard-drive.

If you were to go to the address bar in IE or NN, and type in the address <http://www.canisius.edu/~dunkleb>,

it takes you to the index.html file (<http://www.canisius.edu/~dunkleb/index.html> is the same page). That's because index.html is the DEFAULT, in other words, your browser assumes that's the file you want it to open, because you haven't specified any other.

Fetch is "Drag and Drop", you can drag files you want to test right into the main window (PUT), or drag files you want to edit from fetch to your local harddrive (GET).

To test a file after you've uploaded it to your server, type this address in the address bar of the browser:

<http://www.canisius.edu/~username/pathtofile>

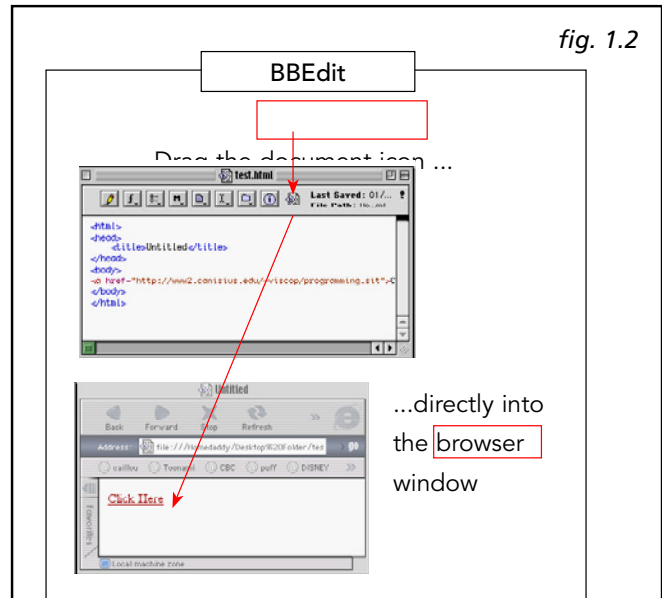


fig. 1.2

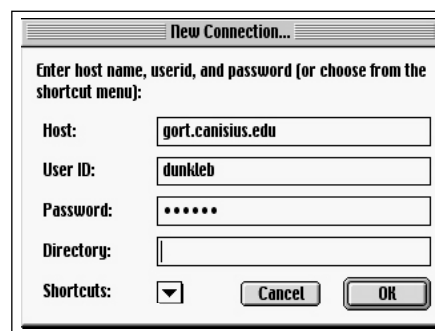


fig. 1.3

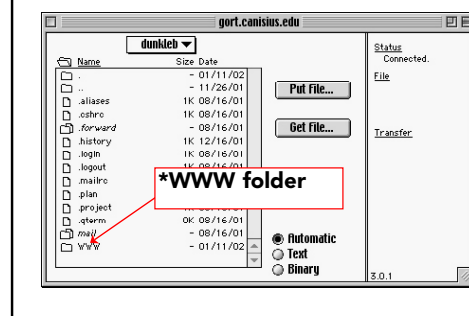


fig. 1.4

where "~" or "tilde" is the character you get when you hold down the shift key and press the key next to the 1 key, "username" is your user name, and "pathtofile" is the path, in slash-syntax, to the file you want to view.

Find and Replace

Text is the backbone of every html file. We have text inside tags, text outside tags, and text as the dominant medium for displaying information on pages.

Because of this, text is everywhere; there's tons of it, and it can get very sloppy. If we have hundreds or even thousands of text files making up a site or application, de-bugging or updating the files can be daunting to say the least.

Fortunately, there's Find and Replace. Because our files are made up of text, it is very easy to modify them. BBedit gives us the ability to edit common pieces of text in a file or directory of files using Find and Replace, and even more power using grep ("Globally search for the Regular Expression and Print the lines containing matches to it").

This window (fig. 1.5), brought up by `⌘ F`, lets us type an "expression", or particular string of letters, numbers, and/or characters, and replace them with another string. We can look in the current file only, all open files, or a particular directory of files.

Using this window is extremely easy, straightforward and useful; you will usually only need it at it's most basic functionality. It's also very similar to find and replace windows in many other programs including Dreamweaver and Golive.

However, this button (fig. 1.5.1) opens up new possibilities. Often the strings we want to find are similar but different. We may want to find all `` tags, along with any attributes assigned to them, and eliminate them. In this case, we'd need to use grep, because there's no absolute way to describe all `` tags.

As a real world example, if I have a file with code that says `` and another file that says ``. How can I find and replace both lines of text with `` in one shot?

Using Grep, I'd put ``

(which will find any `` image) in the find box, and `` in the replace box.

I could also do this with `` in the find box.

"The Find dialog box's 'Match Case' and 'Entire Word' options turn on special searching patterns. Suppose that you're looking for 'corn'. With the 'Case Sensitive' option turned off,

fig. 1.5

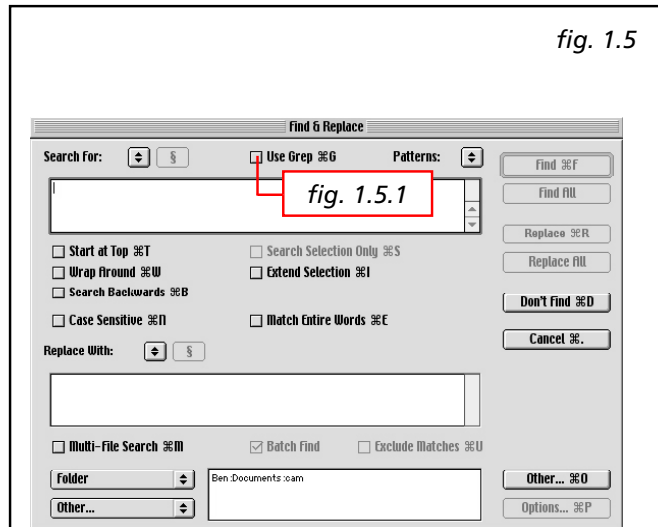


fig. 1.6

Decoding Regular Expressions			
Symbol	What It Means	Example	What It Might Find
+	find 1 or more of preceding item	ta+	ta, taa, taaa
*	find 0 or more of preceding item	ba*	b, ba, baa, baaa
?	preceding item is optional	ab?c	abc, ac
.	match any character (except return)	1.2	1 2, 122, 1d2, 1\$2
[^x]	match any character except x	us[^a]	usb, usc, usd, use
[0-9]	match any digit	9021[0-9]	90210 through 90219
^	match start of line	^hello	matches <i>hello</i> only at beginning of line
\$	match end of line	end\$	matches <i>end</i> only at end of line
()	group items together	(ab)+	ab, abab, ababab

you're actually looking for a pattern that says: look for a C or c, O or o, R or r, and N or n. With the 'Entire Word' option on, you're looking for the string 'corn' only if it's surrounded by white space or punctuation characters; special search characters, called metacharacters, are added to the search string you specified to indicate this.*

(*adapted from <http://logidac.com/abuseEmail/grep/aboutgrep.html>)

Fig. 1.6 shows a list of regular expressions and how they work.