Hacking in Hot Potatoes:
A little knowledge brings a lot of power.

Stewart Arneil and Martin Holmes
University of Victoria Humanities Computing and Media Centre

Contents

Introduction

The Hot Potatoes authoring suite has now reached version 6, and has over 150,000 registered users all over the world. Over the years, as we've developed the programs, our main intention has been to make it very simple for ordinary users to create very complex interactive Web pages — exercises that present material, evaluate user responses, and provide feedback and scoring. The idea is that you can produce fairly sophisticated Web-based teaching materials without understanding anything about the code (XHTML, Cascading Stylesheets and JavaScript) behind the scenes. We provide interfaces into which users can type data (questions, answers, etc.), and make basic decisions about presentation (colours, font choices and so on); then the programs do the hard work of turning all this into working Web pages.

However, from the earliest versions, we have recognized that for many users this is simply not enough. The configurable items in the Hot Potatoes programs are necessarily limited to the set of options that are of the most use to the majority; there is no way that we could enable every single configuration any user might require, because the programs would be huge, and hopelessly complex. In our own work using Hot Potatoes, we have always needed to create exercises that do not function or appear in quite the default Hot Potatoes manner. Therefore we have built the architecture of Hot Potatoes in such a way that anyone with a little knowledge about the underlying code can easily make significant customizations without risk. In this presentation, we'll show three examples, each showing how a little coding knowledge in one of HTML, CSS, or JavaScript can give you a lot of power over how Hot Potatoes exercises appear and behave.

Why we call it hacking

As software developers, our job is to know the coding languages we use very well indeed. However, neither of us has any formal training in programming; everything we know was learned through reading, examining other people's code, and trial-and-error. Few working teachers have the time to do formal programming courses, and most would think that any kind of programming is therefore beyond them. In an effort to avoid what they perceive to be a difficult subject, many instructors fall back on WYSIWYG tools such as FrontPage and DreamWeaver to build their Websites, and become so heavily dependent on these tools that they're unable to manipulate any of their Web content unless they can figure out a way to do it using their editor. This dependency can be very counter-productive; it's possible to invest hundreds of hours in learning a complex program such as FrontPage, and still be frustrated by the inability to find a way to do a task that is very simple to someone who understands basic HTML. For these reasons, we have always tried to encourage users of Hot Potatoes to overcome their fear of code, and invest a little time in learning how HTML and CSS work. Even a very limited knowledge of the basics can give you a remarkable amount of power when it comes to customizing both ordinary Web pages and Hot Potatoes exercises.

The HostingWorks dictionary definition of hack (http://hostingworks.com/support/dict.phtml?foldoc=hack) includes these senses:

- A quick job that produces what is needed, but not well.
- An incredibly good, and perhaps very time-consuming, piece of work that produces exactly what is needed.
- To interact with a computer in a playful and exploratory rather than goal-directed way.

Our suggestions incorporate all of these ideas. You can achieve a small goal quickly and simply with a little knowledge; you can achieve the ultimate customization if you really get stuck in and learn the nitty-gritty; and you can just have a lot of fun playing around to see what works. We'll show you how to use a little bit of knowledge of HTML, CSS and JavaScript to get more control over your Hot Potatoes exercises.

Example 1: Typing HTML code into Hot Potatoes

Let's say that you're creating a multiple-choice exercise in JQuiz, and you would like to make a piece of text bold. Your question is:

What is the capital of Canada?

and you'd like to make the word Canada bold. If you rely on a WYSIWYG editor to do this, you'll have to follow these steps:

1. Make the exercise in JQuiz, and export it to create a Web page.
2. Open the Web page in your WYSIWYG editor.
3. Find the word Canada, and bold it.

Not too onerous, but what happens if you need to make a change to your exercise in JQuiz later, to add a new question? You'll have to do the same process over again. If you're bolding some text in every question, you'll end up making dozens of changes over and over again, each time you need to edit the exercise.

If you're prepared to dip your little toe into the world of HTML code, there's no need. All you need to do is to type your question in JQuiz like this:

What is the capital of Canada?

That's it. The b tag makes text bold in a Web page. When you create the exercise, Hot Potatoes will pass the tag through to the Web page, where the browser will read it and display the word Canada as bold text, like this:

What is the capital of **Canada**?

You can use a similar approach to perform a wide range of formatting tasks in your Hot Potatoes exercises, changing the style of text, fonts, and colours, adding superscript or subscript text, or inserting bulleted or numbered lists. Learning the code you need is very simple -- just work your way through an introductory tutorial such as the one at W3Schools (http://www.w3schools.com/html/default.asp), and start hacking. Hot Potatoes itself includes a set of keystroke shortcuts you can use to enter common HTML tags; for a list of these, check out the topic Keystroke shortcuts for HTML tags in the Hot Potatoes help file.

Example 2: Hacking the Cascading Stylesheet file

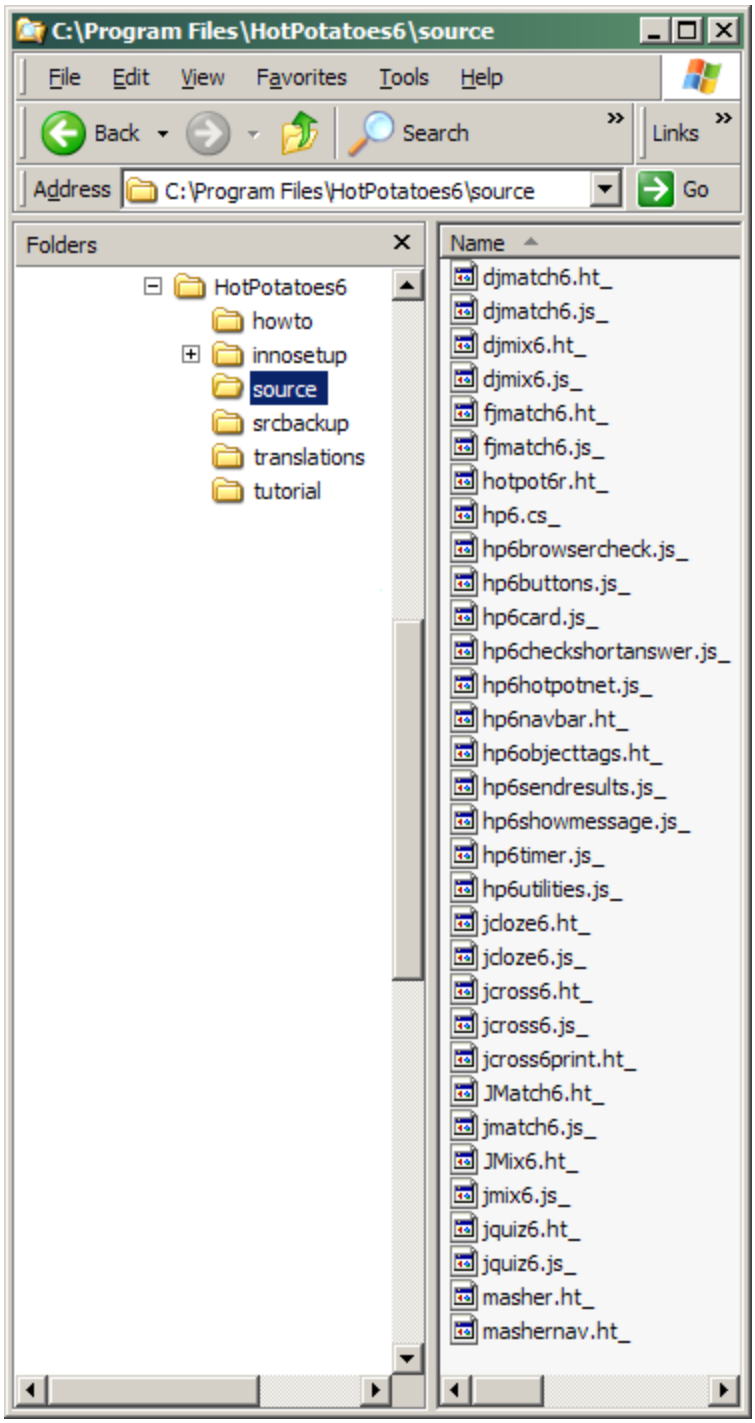
The second stage in learning to hack Hot Potatoes involves understanding what happens when you export a Hot Potatoes exercise. When a program such as JQuiz builds an exercise page, these are the steps it takes:

1. It loads into memory a set of templates containing the basic XHTML, CSS and JavaScript files. These are called source files in Hot Potatoes.

2. It combines these files to create a single template for the exercise.
3. It builds all the exercise data and configuration information into the template, to make a complete exercise page.
4. It saves the page to disk.

The source files are stored (by default) in a folder called source inside your Hot Potatoes program folder. Find this folder in Windows Explorer. If you installed Hot Potatoes with the default settings, it will be here:

c:\Program Files\HotPotatoes6\source



When you find the folder, you'll find a long list of files, each of which has an extension ending with an underscore, like this:

- **hp6.cs_**
- **jmatch6.ht_**
- **jmatch6.js_**

Each file contains a specific type of code that is used to create the exercise. The first file, **hp6.cs_**, is the one we're interested in. The meaning of the extension is roughly this: the **cs_** file contains CSS (Cascading Stylesheet) code, which is used to control the style and appearance of the finished Web page. CSS files usually have a **.css** extension, but because this file is only a template, not a well-formed CSS file, it has an underscore at the end.

The **hp6.cs_** file is used by all the Hot Potatoes programs, and it contains all the CSS code used for the display of exercises. Before we edit the file, we will take one step that ensures that the changes we make will not be destructive; we're going to copy the file to a new location, so that the original file remains unchanged, and then we're going to tell JQuiz to look in the new location for our changed file. If we screw up, or want to abandon the changes, we can just tell JQuiz to go back to using the original file in the source folder, and no harm will be done. This is how we do it.

1. Create a new folder inside the Hot Potatoes folder called (for example) **my_source**.
2. Copy and paste the **hp6.cs_** file into that folder.
3. Start JQuiz.
4. Press **Control + Alt + Shift + S** on your keyboard. (Told you this was hacking!)
5. Find and select your new folder.

This tells JQuiz to look for source files in the new location first; if it finds a source file there, it will use it, but if it doesn't find it there, it will look in the default folder. Therefore, when we create our exercises, JQuiz will now use the version of **hp6.cs_** in the new folder, but it will find the other source files it needs in the default location because they don't exist in the new folder.

Note: When you've finished hacking around, and you want to tell JQuiz to stop looking in the new folder and use all the original source files, start JQuiz and press **Control + Alt + Shift + T** on your keyboard.

So now we've made a copy of the CSS file, and told JQuiz to use the copy. It's time to start modifying the file. The modification we're going to make is one that a user recently asked about on our Hot Potatoes bulletin board. By default, when a student chooses an answer in JQuiz, he or she sees a little popup box with the feedback for that answer. The feedback is centred -- the convention for popup boxes in most computer systems is to have centred text, so that's what JQuiz does. However, if your feedback is long or complicated, it would be more appropriate to make the feedback text left-aligned like a regular paragraph. That's what we're going to do.

First, you'll need to open your copy of **hp6.cs_** in a text editor (NOT a word-processor). If you don't have a favourite text editor, use Windows Notepad (NOT WordPad). Make sure you open the copy you made in your new folder, not the original. When you open the file, you should see a lot of code like this:

```
/* This is the CSS stylesheet used in the exercise. */
/* Elements in square brackets are replaced by data based on configuration
settings when the exercise is built. */


body{
        font-family: [strFontFace];
[inclPageBGColor]       background-color: [strPageBGColor];[/inclPageBGColor]
        color: [strTextColor];
[inclGraphicURL]        background-image:
url([strGraphicURL]);[/inclGraphicURL]
        margin-right: 5%;
        margin-left: 5%;
        font-size: small;
}

p{
        text-align: left;
        margin: 0px;
        font-size: small;
}

div,span,td{
        font-size: small;
        color: [strTextColor];
}
```

...and so on.

If you don't know CSS code, this will look a bit mysterious. This is where you need to do a little learning. Take a few minutes to go through the W3Schools tutorial on CSS:

http://www.w3schools.com/css/default.asp

If you do know CSS, this will be mostly familiar to you. The only unusual elements are the items in square brackets (such as [strFontFace]). These are placeholders, and they are replaced with appropriate values by the Hot Potatoes programs when they create their exercises. For example, if you specify Arial as your font in the JQuiz configuration screen, [strFontFace] will be replaced by Arial.

You'll find that the basic ideas are simple; each block of code in the CSS file defines the way a particular HTML element or elements will be displayed. For example, the above code specifies that text inside a <p> (paragraph) tag will appear left-aligned, with a small font size, and no margin (zero pixels).

In this exercise, we want to change the text alignment of the JQuiz feedback box. How do we know what to look for? Well, this is hacking; you have to look around and figure out what might work. Let's search for the word Feedback. The first item you find should be this one:

```
.FeedbackText{
        color: [strTitleColor];
}
```

This might be a good place to start. Try adding this:

```
.FeedbackText{
        text-align: left;
        color: [strTitleColor];
}
```

Now try making a JQuiz exercise with a multiple-choice question, and click on one of the answers. The feedback should now be left-aligned in the popup window. Success!

As you read through the **hp6.cs_** file, you'll find that it's easy to guess what most of the settings refer to; we've tried to use descriptive identifiers such as FeedbackText or TimerText to make it easier for hackers to customize the settings. Nevertheless, you're still going to have to make some guesses and play around a little. That's how hacking works.

Example 3: Changing the JavaScript scoring system

Our final example is more challenging. What we're going to do is change the JMatch scoring algorithm, so that instead of giving a percentage score, it gives a PASS or FAIL result. To do this, we'll have to learn a little JavaScript. JavaScript is a programming language that's used to make Web pages dynamic (in other words, to change elements of the page in response to user actions). As usual, you can get started over at W3Schools:

http://www.w3schools.com/js/default.asp

Now you've learned a little JavaScript (don't worry about learning too much at this stage), we can get going with our hack. The file we need to modify in this case is **jmatch6.js_**. This file contains all the JavaScript code that is specific to a standard JMatch exercise (the format with drop-down lists, not the drag-drop format). As you did before, copy your **jmatch6.js_** file from the Hot Potatoes **\source** folder to your new **\my_source** folder, then start JMatch, press **Control + Alt + Shift + S**, and tell the program where to look for the new source file.

Now open the source file in your text editor. You should see something that looks like this:

```
var CorrectIndicator = '[strCorrectIndicator]';
var IncorrectIndicator = '[strIncorrectIndicator]';
var YourScoreIs = '[strYourScoreIs]';
var CorrectResponse = '[strGuessCorrect]';
var IncorrectResponse = '[strGuessIncorrect]';
var TotalUnfixedLeftItems = 0;
var TotCorrectChoices = 0;
var Penalties = 0;
var Finished = false;
var TimeOver = false;
```

...and so on.

These items at the beginning of the file are variables used by the page. As in the CSS file, items in square brackets are placeholders that will be replaced with proper values by JMatch.

Now we're looking for the bit of code that calculates the score. Do a search for the word score. The second hit should find a block of code which looks like this:

```
//Calculate the score
        Score = Math.floor(((TotCorrectChoices-
Penalties)/TotalUnfixedLeftItems)*100);
        if (Score<0){Score = 0;}
        var Feedback = '';

//Build the feedback
        if (AllDone == true){
                Feedback = CorrectResponse + '<br />' + YourScoreIs + Score +
'%.';
        }
        else{
                Feedback = IncorrectResponse + '<br />' + YourScoreIs + Score
+ '%.';
```

```
//Penalty for incorrect check
                Penalties++;
        }

        if (AllDone == true){
                WriteToInstructions(Feedback);
        }
```

Here, two things are happening; the score is being calculated, and it's being displayed. The second operation has two variants, one of which is used when the answers are all correct (i.e. the exercise is finished), and the other one of which is used when the answer is only partially correct.

Now, let's imagine that 70% or above is a PASS, and anything below that is a FAIL. We want to tell the student You have failed or You have passed, instead of giving a percentage score. (Incidentally, we're not advocating this as a pedagogical approach; this is just an example of hacking the code.)

Now, we only want to give the result at the end of the exercise, so the only change we need to make is in the first variant:

```
        Feedback = CorrectResponse + '<br />' + YourScoreIs + Score + '%.';
```

Let's start by creating a string of text dependent on the value of the score. We can do this *after* the score is calculated, but *before* the if clause, like this:

```
        Score = Math.floor(((TotCorrectChoices-
Penalties)/TotalUnfixedLeftItems)*100);
        if (Score<0){Score = 0;}
        var Feedback = '';

//New bit:
        var FinalResult = ''; //this is an empty string
        if (Score>69){
                FinalResult = 'You have passed!';
        }
        else{
                FinalResult = 'Unfortunately, you have failed. Try the
exercise again.';
        }
```

Now we need to display this string instead of the percentage score, by changing this part:

```
        if (AllDone == true){
                Feedback = CorrectResponse + '<br />' + YourScoreIs + Score +
'%.';
        }
```

to this:

```
        if (AllDone == true){
                Feedback = CorrectResponse + '<br />' + FinalResult;
        }
```

That should do it. Now make a JMatch exercise and test it out!

Final thoughts

Clearly, the more you know, the more you can do. Even a little knowledge of basic HTML enables you to add superscripts, subscripts and other text formatting, but if you gain a thorough knowledge of JavaScript, you'll be able to understand and change any of the functions of the Hot Potatoes exercises; you can make them do anything you like. Customizations like this can never be achieved using WYSIWYG editors such as DreamWeaver or FrontPage. But the real power of this kind of approach is that it enables you to escape from the predefined style and functionality we have given you, and implement your own approaches based on your own beliefs about what constitutes good pedagogy for the Web. If you believe that presenting students with a score at the end of each exercise is detrimental to the learning process, you can just hack out the scoring code; if you have ideas for very sophisticated answer-checking based on wildcards, you can write the code and integrate it right into the system. Knowledge is power. That's why we're working in education.

This kind of hacking is a great way to learn to code, too. To hack code written by someone else, you need to read and understand it first, then experiment with changing it. Every time you hack, you see how something is done, and eventually you'll build up a good grounding in whatever coding language you're using. However, you don't need to understand everything; you just need to understand the little bit of code that you're trying to change, and you can experiment to your heart's content without risk.

For another short tutorial on how to edit Hot Potatoes source files, with another simple example to get you started, go here:

http://web.uvic.ca/hrd/hotpot/howto/editsource.htm

If you really want to get your hands dirty and learn all about how Hot Potatoes source files work, there is some detailed documentation here:

http://web.uvic.ca/hrd/hotpot/howto/sourcefiles.xml

(The above page uses XML/XSLT, so you'll need Internet Explorer 6, Mozilla or Firebird to view it.)

If you do start hacking into the Hot Potatoes source files and you'd like some help or suggestions, post your questions, along with URLs of example exercises so that we can see what you're doing, on the Hot Potatoes bulletin board.

Happy hacking!
