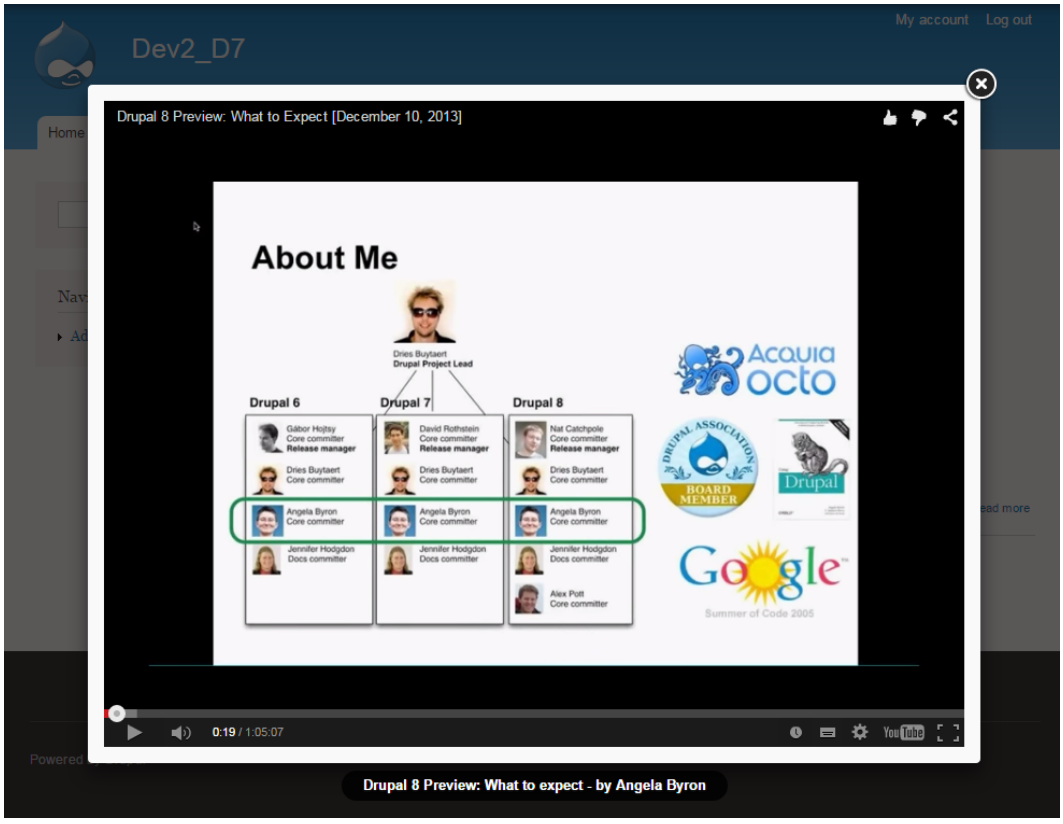
How to build a module for adding your own JavaScript libraries to Drupal 7

Written by Jesús Heredia Reboira

www.jesusheredia.info

The Goal

One of your clients needs to use a certain JavaScript library in Drupal 7, but, unfortunately, there is no module providing support for that library. Say he needs to use the [fancyBox 2] library, which offers a nice and elegant way to add zooming functionality for images, HTML content and multimedia on your web pages.

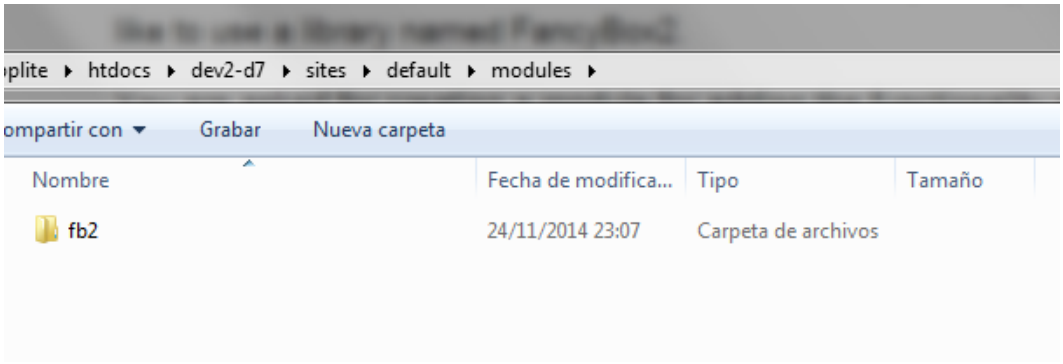


Getting Started

Let's get started by creating the file used by every module to provide Drupal with basic information about itself (the *.info* file).

1. Create a folder to store your module. Although you could put your module in *sites/all/modules*, there is a best place to put custom modules in: *sites/default/modules*. This follows Drupal best practices and allows you to find your own modules easily.

> If your server is running more than one Drupal site (multi-site configuration), then the *sites/default* folder is only used by the default site. If this is your case, you'd better move your custom module into *sites/all/modules/custom*.



First, go to *sites/default* and create the *modules* folder (which does not exist by default). Then, create another folder inside *sites/default* to be named *fb2*. This is the folder you'll be using for your custom module.

> The folder must be named with the machine-readable name of the module and the same applies to the *.info* and *.module* files (fb2.info and fb2.module).

> Make sure that your web server is allowed to read the .info and .module files. In addition, it should not be allowed to write either files.

2. Write the .info file. As said above, the .info file is intended to provide Drupal with basic information about the module. Open your favourite IDE or text editor and create a file to be named *fb2.info*.

The first *directive* that you're going to add to the .info file is the *name* directive. A directive is composed of a name, an equal sign, and a value, like this:

```
name = fancyBox 2
```

> By convention, there should be one space on each side of the equal sign. In addition, if a value spans more than one line, it must be enclosed in quotation marks. If there is a semicolon at the beginning of a line, Drupal will treat that line as a comment, meaning that it'll be ignore by the Drupal INI parser.

Let's have a look at the complete .info file:

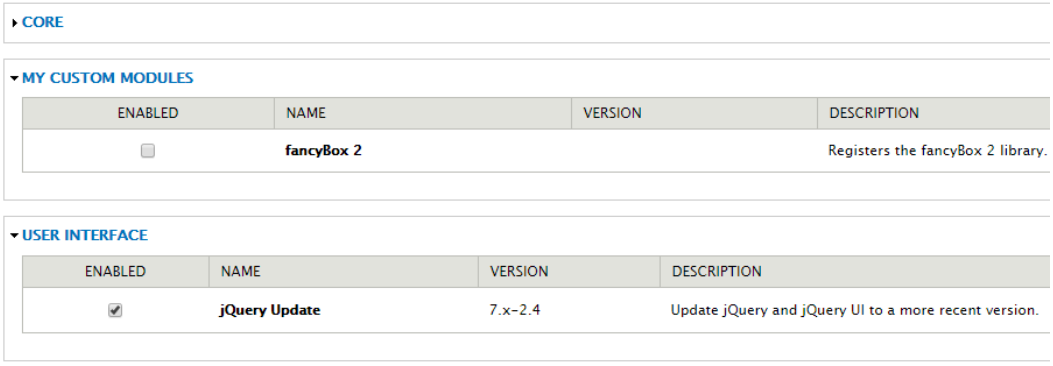
```
;$Id$

name = fancyBox 2

description = Registers the fancyBox 2 library.

package = My custom modules

core = 7.x

files[] = fb2.module
```

While the name, description, package, and core directives are self-explanatory, both the first and last line of the previous piece of code may be a little tricky for you.
Every .info file should begin with *;Id*, which is a placeholder used by the version control system to store information about the file. If you're using Git, then you can omit it.

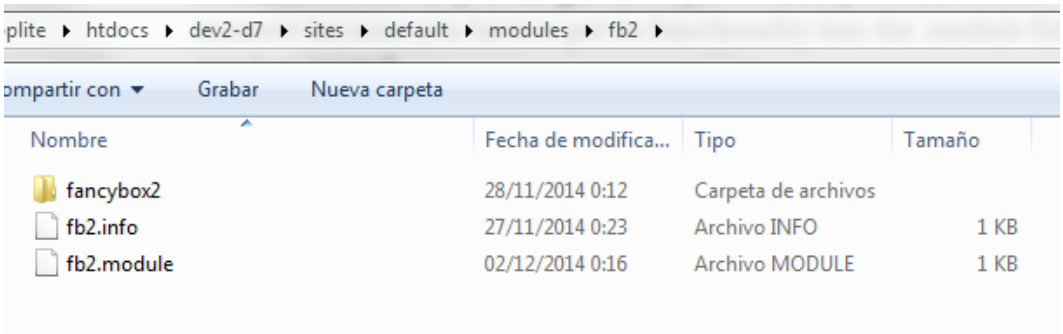
In addition, some directives use an array-like syntax because multiple values can be assigned to them. That's why the square brackets are used by the *files* directive.

3. The .module file. Save all the changes to the .info file and then create and save a new file to be named *fb2.module*.

If you go to your Drupal site now, you'll see that the module is available for use. As you can guess, it's just a dummy module at the moment because you have to put its functionality into the .module file you just created.



Before you start coding the .module file, download the fancyBox 2 library from **fancyapps.com/fancybox**; put it in *sites/default/modules/fb2*; and rename its folder to *fancybox2*. At the time of this writing, version 2.1.5 is available for download.



The .module file

Now that the .info file and the fancyBox 2 library are ready, you can start getting your hands dirty with code.

1. Type the <?php processor instruction, marker for CVS, and Doxygen-style (/ */) block for commenting what the module is intended for.**

```
<?php
// $Id$
/**
 * @file
 * Registers the fancyBox 2 library in Drupal 7.
 *
 * This module is a case study on how to put the
 * functionality provided by a certain library in
 * Drupal 7.
 */
```

```php
<?php
// $Id$

/**
 * @file
 * Registers the fancyBox 2 library.
 *
 * This module is a case study on how to put the
 * functionality provided by a certain library in
 * Drupal 7.
 */
```

In order to parse a file, PHP looks for its opening and closing tags, which are <?php and ?>. That's why every PHP file should start with <?php and end with ?>. However, when it comes to .module files, you must omit the PHP closing tag ?> to prevent the inclusion of white space from breaking HTTP headers.

As to comments, you should use Doxygen-style comments for functions, classes, interfaces, constants, files, and globals. All other comments should use the double-slash comment. As you can see in the previous image, this is the one used to comment the marker for CVS.