Developers Webinar #3: Developing Custom Modules For Drupal

This webinar is the third in a series of three webinars about Drupal in legal services. The first webinar provided an introduction to the Drupal website content management system and how to use it to build legal services websites. The second webinar coverred how to plan, architect, and build a Drupal website for legal services This third webinar covered building custom modules in Drupal.

The webinar was held on Wednesday, November 30 , 2016 - 11:00am Pacific.

The webinar was presented by Abhijeet Chavan and Ki Kim.

Webinar Transcript

What’s required to build a module?

Drupal’s functionality can be extended by adding modules. In fact, we could say that true power of Drupal comes from thousands of community developed modules. Drupal itself comes with built-in core modules. There are about 40 core modules. In addition to core modules, the Drupal community has developed contributed modules.

There is nothing special about core modules that separate them from the rest contributed modules. Drupal’s maintainers happened to pick most important and popular modules and bundled them with Drupal package so you don’t have to download them separately. However, only essential modules are included in Drupal package, so it is very common to add functionalities by downloading contributed modules. It is common to install more contributed modules than existing core modules.

It is very possible to build a Drupal site using contributed modules and not write a single line of code. However, Drupal is a good framework as well as a content management system(CMS) . It is best practice to search as hard as possible, for contributed modules that perform the function you want. After searching through available contributed modules, if you cannot find a module that fills your specific need, then it is time to build your own custom module.

Whether a module is Drupal’s core, contributed, or custom built, they all follow same rules and there is no special treatment on certain groups of modules. The same coding pattern used for core or contributed modules is used to develop a custom module.

Let’s see where the modules are located in the code base. Core modules are in `/modules` folder. You see about 40 of them. If you open sites folder, there is folder called `all`, and that’s where all contributed or custom developed modules and themes are located.

What if you placed downloaded or custom developed modules inside `/modules` folder at the top. Will it work? Yes, it would. However, one of the rule is /"never hack the core"/. It means you should not alter Drupal core code to change its behavior. You should do your best to extend Drupal with developing custom modules or themes. There could be some exception, where there is no other way but to change Drupal's code. But it's the general rule not to touch Drupal core.

You will find yourself upgrading Drupal code throughout the lifetime of your site, sometimes to add functionality to Drupal core, and more often, you want to apply upgrades when security patch version is released. Which is very important to keep your site in good condition and guard against various attacks from the internet.

In order to make upgrade easy, it is a best practice to leave Drupal code alone and not touch it. It also means it is best not to place your custom code among Drupal core. Any contributed or custom developed code should be placed under designated place, the `sites` folder.

Under `sites` folder, there are `all` and `default` folders. Most of the time, all modules and themes, and 3rd party libraries should go under `sites/all` folder as you can see.

Inside `sites/all/modules` folder, we typically separate contributed modules and custom modules in each folder. It is by convention that most Drupal developers follow. You could mix contrib modules and your custom modules without separation and it will work just fine. However, it is considered best practice for organization purpose.

Inside contrib folder, you can see some contrib modules that were downloaded from last webinar.

Now, let’s create our first custom module. Decide a module name for the folder. The module name to use in the code should be lower case staring with an alphabet optional using underscore if it consists of multiple words. So names like "views" or "admin_menu" are valid module name.

For this demonstration, we will develop a module that creates a custom block that displays related content of a given article. Let's ccall it "related"

I'd like to check if the module name is not taken already. You notice all contrib modules post to drupal.org can be found at this URL.

- Check http://drupal.org/project/xxx for conflict

I feel more comfortable if the name is not taken. Why does it matter? It may or may not. If your custom turns out to be very useful and you decide to share your hard work with Drupal community, then you want your module's name to be unique.

So is it okay if you name your module with an existing name? In fact, it will work, *as long as* you don't download and use the contrib module with same name. Otherwise using two modules with same name will confuse and break your Drupal site. But you never know what will happen in the future. It is best to reduce any possibility of confusion from start.

When I develop a custom that I anticipate will end up as a contrib module, I may register an empty module to drupal.org even before I build it, in order to reserver the module name so nobody can take it.

So back our module, we've decide to name it "related".

I am going create a folder called "related".

Inside it, I am going to create a file related.info

It is text file with a list of key/value pairs.

```
name = Related
description = My first custom module
core = 7.x
version = 7.x-0.1
package = Block
```

This is almost enough to install your module. We just need to add one more file related.module. It can actually be empty for a start.

When we check list of module, towards the bottom of the modules list is our custom module under OTHER category.

We can successfully enable the module, but it doesn't do anything for now.

Let's make it do something.

Developing a Drupal module mostly revolves around building php function with certain naming conventions. Here come a thing called "hook"

Drupal's module system is based on the concept of "hooks". A hook is a PHP function that is named `foo_bar()`, where "foo" is the name of the module (in our case, "related") and "bar" is the name of the hook. Each hook has a defined set of parameters and a specified result type.

To extend Drupal, a module can implement a hook. When Drupal wishes to allow intervention from modules, it determines which modules implement a hook and calls that hook in all enabled modules that implement it.

Available hooks to implement are explained here (first link on 7.x API page) in the Hooks section of the developer documentation. The string "hook" is used as a placeholder for the module name in the hook definitions. For example, if the module file is called related.module, then hook_help() as implemented by that module would be defined as example_help().

Links:
Webinar #3 Presentation
