Video 1: Why build custom modules?

OK, let's talk about building your own modules. First off, let's quickly define what we're talking about here.

When you're building HubSpot templates, you'll use a combination of modules and groups to simultaneously create the layout of the page and the editing experience for your content editor. There are a couple dozen built-in modules to handle common content such as images, headers, and menus.

These default modules are fundamental components of the CMS and you'll probably use them regularly. That being said, most developers will eventually want to build their own modules as well. There are three main reasons to build your own module:

The first reason folks build custom modules is to create more complex design solutions. Some common examples would be accordions, sliders, and tabbers.

The second reason to build your own module is to create a customized implementation of a default module type. You may want to craft your own specialized markup or add extra options for your content editors.

The third reason to build your own module is because it's actually a best practice. Technically speaking, you can build fairly complex and customized components using only the template editor. That being said, in most situations where you're deciding between building something directly into a template or building a module, you should strongly consider the module option, even if it doesn't appear necessary at first glance. Let's take a moment to explore the reasoning here.

We can essentially boil this topic down to one statement: Modules are consistent and portable. This means that once you figure out how to implement a design solution, you can apply that same solution every time it's needed. This is great for both content editors and developers. On the editing side: Consistency is solid gold for the content editor's user experience. On the developer's side: When you inevitably need to make adjustments to a component, you can quickly modify one canonical source instead of digging through multiple files to make changes throughout a site.

Video 2: How to build a module in the HubSpot CMS

OK, let's walk through the process of creating a module in the HubSpot CMS.

The most common way to build a module in HubSpot is to use the same design manager you use to build templates. The module editor has a very similar interface to the template editor, with an editing section and an inspector section. The editor section for templates is all code based divided by HTML+HubL, CSS, and Javascript. You'll use the inspector section to generate new fields to use in your module, to link stylesheets, scripts, and images, and to assign tags to your module for organization. This section also gives you a code snippet for using your module in coded templates.

The heart and soul of the module editor is the Fields section and the HTML+HubL window. With these two sections, you can create fields and weave them into your HTML. The fields you create here are the fields that

your content editor will populate to build a page. There are also options to group fields and make fields and groups of fields repeatable.

The available fields are sorted into Content, Logic, and Selector categories. Content fields are for things like text and dates. You can use Logic fields to create switches and dropdown menus for your content editors. These can be especially handy for building more complex modules. Selectors allow editors to grab content from other parts of the CMS such as images, forms, and CTAs.

Each field has a number of options that you can set such as default content and validation. These options give you fairly granular control over how content editors interact with your module. It's safe to say that module fields are probably the most powerful tool for CMS developers to build intuitive and bulletproof solutions for content editors.

Fields are added to the module markup with HubL. This is the HubSpot templating language. Check out the Resources tab for links to more educational content on HubL.

In the inspector each field features a Copy snippet link that will load up your clipboard with the necessary HubL for using that field in the HTML + HubL window. This snippet contains suggested code for typical usage. As you work on more complex modules, you can modify this code to fit your needs.

After you've created your fields and markup, you'll probably want to style your module and perhaps add some functionality with JavaScript. The module builder is not overly prescriptive about how you approach your styles and script. You can make your module as specific or as abstract as you'd like. That being said, keep in mind that modules are designed to be used in multiple templates and often more than once in a template. Make sure you plan ahead and test thoroughly.

Also, there’s one major technical thing you should be aware of when working with scripts and styles in modules:

HubL cannot be added to the CSS or JS windows.

HubL is not available in either of these windows with the exception of one special function named module_asset_url that allows you to link to assets. See the Reference section of this video for more information on this function and suggestions for passing data to CSS and Javascript.

Once you're finished creating your module, click the **Publish changes** button to publish its current state. At this point it will be available in templates. If you'd like to work on a module without it being available for templates, you can disable this with the **Make available for templates** switch.

Video 3: Creating a module in the HubSpot CMS

Ok, let's take a look at building a module in the HubSpot CMS. First off, here's a glance at the finished module. This is a fairly simple stylized two column content area with an image. It allows content editors to add an image and some text. It also allows them to pick a background color and to switch the image position.

This module allows us to explore some fundamental aspects of building a module, but it's also worth mentioning that it's a good example of balancing content editor flexibility with some guardrails. [delete?:]There are lots of

ways for a CMS developer to create a two column layout like this. In this case, editors get control of the color and orientation, but they don't have access to things like font formatting or the ability to add links or CTAs. This keeps design consistency in the hands of the cms developer.

Ok, let's get started.

To build a new module, click file and then select New file from the drop down menu. In the new file dialog click Module. In the setup dialog, select where this module will be used, in this case we'll create a module for page templates. Then name your module. Let's use the name two column feature for this example.

Now we have a blank module ready for some fields.

Let's add the text field first. Click Add field and scroll down the content column and select Text. Give this field an intuitive name. In this case, we'll use "Feature Text" to correspond with the name of the module. Next add some default content to help folks use this in the editor. "Add feature text. Make it exciting."

Ok, now to add this field to our module markup, click copy snippet. Paste this snippet into the HTML/HubL window with command + v on a mac or control + v on a PC.

Now let's use the preview button to see where we're at. So far so good. On the left we can see the editing experience with our field name and default content. On the right we see a preview of how this module will appear. So far we're just printing out some text so there isn't much to look at.

Ok, let's add the image field. In the breadcrumb menu at the top of the inspector, click the module icon to return to the module's home state. Click Add field and scroll down the Selectors column. Click Image to add an image field. Let's name this field Feature image in keeping with our example. Then in the Content options section of the inspector, select a default image.

Now to add this image to our markup, we'll again use Copy snippet. Click the Copy snippet link above the variable name field and paste the snippet into the HTML/HubL window.

We won't dive too far into the details of this HubL code, but it's worth mentioning two things. One, this default snippet contains a conditional statement. This is a good idea with an image field like this. If no image is selected, this will ensure that we don't end up with a broken image tag cluttering up our page.

The second thing to point out is that this default HubL adds width and height attributes to this image tag. This is another option that CMS developers should consider carefully. These width and height attributes allow content editors to specify the size of this image. Sometimes this is desirable, sometimes it isn't. In our case, we're building a responsive module so a fixed size image isn't really what we're looking for. Let's select these attributes and remove them.

Switch browser tabs to preview what we've got so far. We have some text on top and an image below. Let's add a little markup and some CSS to achieve our two column layout. First we'll switch tabs again to bring up the module editor.

There are lots of ways to do this, but we're going to use flex box. You may want to include some fallback CSS if you're supporting older browsers, but this is a demo so we won't worry about that right now.

Let's take care of the markup first. Wrap the entire module in a div and give it a class. We're going to use "demo-two-col" here for the sake of example. These markup decisions are mostly up to you, but you will want to make your CSS specific enough make this module portable. The CSS you add here will be included on any page where your module is used so generic selectors like tags aren't a good idea.

Next we'll wrap each column in a div. We'll give each of these columns a class of "col" and we'll also add "text" and "image" classes so we can target these separately.

Ok, let's add our CSS. First let's set the wrapper div to flex, then set the columns to 50%. Finally, add a max width to the image to constrain it to our column. If you're using any boilerplate CSS on your site, you'll likely already have this rule but we're working from a blank slate here so we'll add it in.

If you tab over to the preview, you'll see that we now have a two column module. There are still some tasks remaining: We need to style this text and we also need to add a background color selector and a switch to reverse the columns. But this is a solid start.

Video 4: Adding custom HubL to a module

Ok, now we're looking at a module with two columns and very little else. Here's a look at the module we're working towards. We still need some text styles, the background color selector, and the image position selector.

Let's start with the background color selector.

Tab back over to the module editor, and use the breadcrumbs at the top of the inspector to get back to the module's home state. Click Add field and choose Color from the Selectors column.

Rename the field Background color. For the sake of example, we'll also set a default of ccc.

Now, we need to plug this background color into our CSS, but we can't actually put HubL into the CSS window. Check out the resources section of this video for more details on this. What we'll do in this case is use a bit of inline style. Click Copy snippet to get the HubL for this field. Now paste this snippet below the rest of our markup. As you can see, we get two expressions here. This is the reason we're using the bottom of our markup as a sketchpad right now. We aren't going to use the opacity value in this example, so we'll select it and delete it. We do want this color value, so let's select that and use command or control X to cut it.

Add a style attribute to the wrapper div and a rule for background color, then paste in that HubL.

Now, tab over to the preview. The background gray fills in behind the module. We can also use the editor preview on the left to set a new value.

Ok, let's add the last field. Tab back over to the module editor. Use the breadcrumbs in the inspector to get back to the module's home state and then click Add field. Select the Choice field in the Logic column.

Rename this field Image position. Now let's populate the choices.

In the Content options section of the inspector, enter Image Right as the first label and Image Left as the second label. Enter img underscore right as the first value and img underscore left as the second value. Click on the default drop down menu and choose Image Right as the default value. Ok. We've got our field data set up. The default value is the current default state of the columns with the image on the right.

What we need to do is to override this state when a content editor selects Image Left. Let's use the bottom of the HTML/HubL window as a sketchpad again to talk through this. Click Copy snippet above the HubL variable name and paste the HubL into the bottom of the HTML/HubL window.

This is a surprisingly simple HubL snippet for what might seem like a complicated field. If we tab over to the preview, we see that this will simply spit out the selected value. Right now that's the default of img underscore right. Let's tab back over to the module editor and fold this into our markup.

All we really need here is a hook for our CSS so that we can write a new rule to switch the direction of the columns. We could simply paste this HubL expression into the class stack and style off of that.

For the sake of example though, let's use a conditional. In the class stack we'll add an if statement to simply check if the image position is left since that's the exception to the default. If it is, then we'll add a class of reverse.

Now we'll add a new CSS rule to put this into effect. This is where flexbox comes in really handy. Instead of messing around with swapping floats, we can just reverse the rows if our wrapper has the reverse class.

Tab over to the preview and give it a try. The text still needs some style, but the module functionality is all set. Tab back over to the module editor to finish up the CSS.

For the sake of brevity, I've pasted in the final bits of CSS here, but lets just briefly walk through it. First we've added flex and align items to the columns to get that nice vertical alignment of the text.

Then we changed the alignment of the text for the reversed rows.

And finally, we added some style to the text itself. I'm personally a sucker for letterspacing and uppercase text, but I'm sure you have your own favorites to add here.

Once you've finished creating your module, click Publish changes.

To use your custom built module in a drag and drop template, use the add section of the inspector just like you would with any other module.