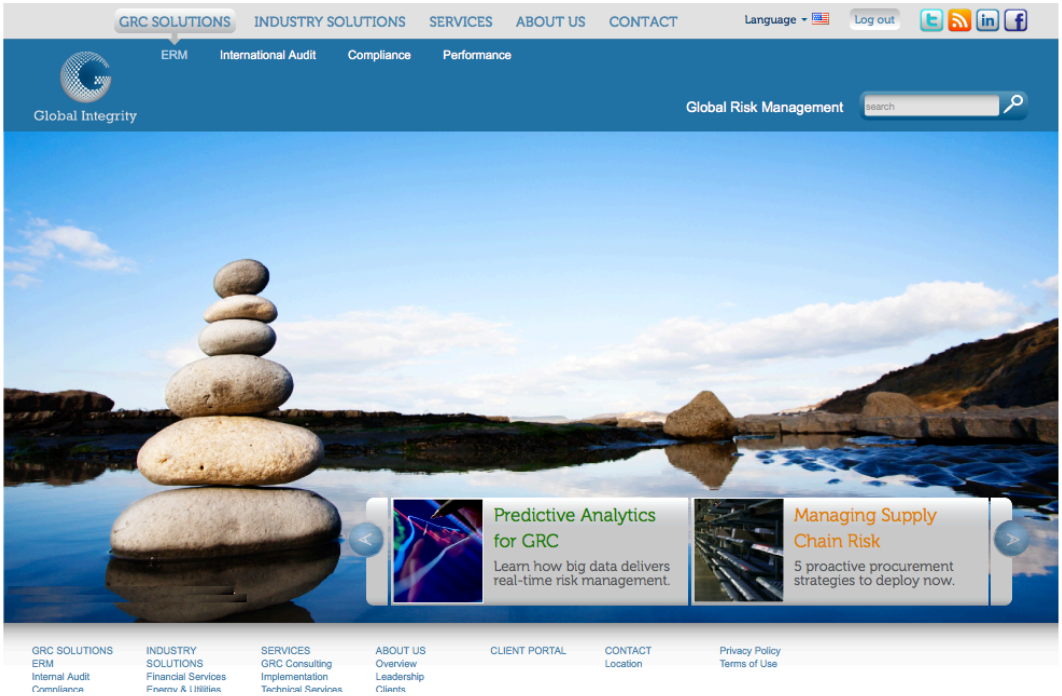
Create a Dynamic Website

Today's internet is noisy. Our customers are bombarded with content and messaging from all angles. As a result they quickly learn to filter out anything that is not directly and immediately useful to them. Thus there is ever growing importance for our websites and services to be up to date with fresh content and for that content to speak as directly to them as is possible.

In this article we'll explore a number of ways to leverage the Crafter CMS platform to create a dynamic website that can be updated quickly and effortlessly to keep your content fresh and timely. We'll go beyond the basics to show how you can support complex requirements and content production processes and finally we'll demonstrate how to target your content directly to the visitors on your site.

Author Selected Content

In many cases our authors want to make a specific editorial choice about what content to display on their site. For example, they may want to decide which promotions to show on the home page.



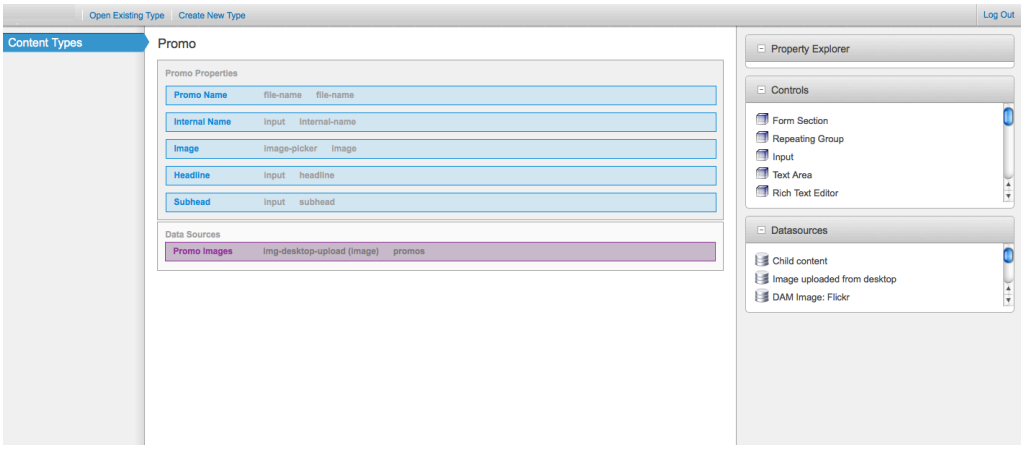
To begin let's keep things simple and start with a use case that has a very basic set of requirements:

- Authors manage a collection of promotional objects which can be assigned to the home page for display in a carrousel.
- Authors make a daily decision about which objects to show on the home page from the collection or gallery of objects from within the CMS.
- All site visitors see the same promotional objects.

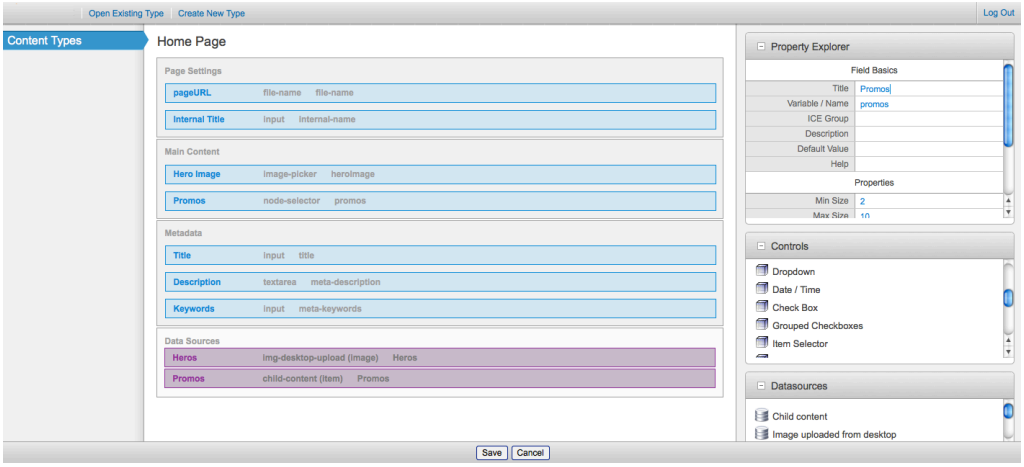
To allow authors to select content to be associated with a page, region, or component in Crafter CMS we must create a content association between the target content and the object that will point to them. In the case of our example above there is a parent child association between the home page and the promotions.

To create this association we use the Content Type Editor in Crafter Studio, which is the authoring environment for Crafter CMS.

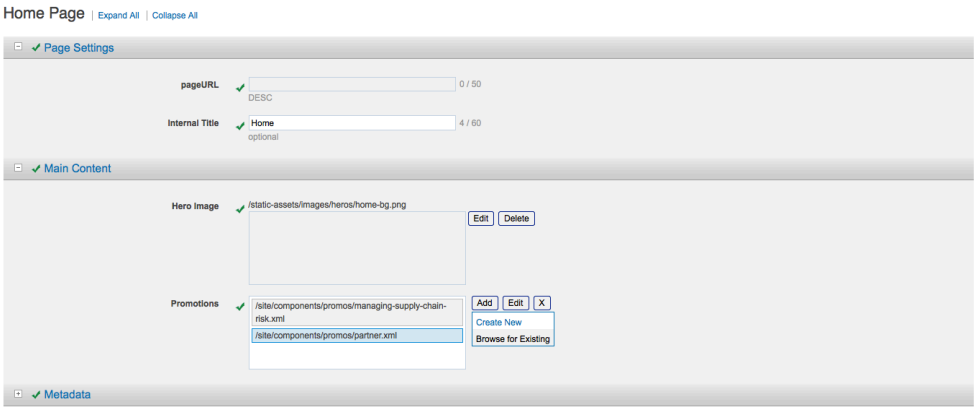
First we create the promotion type. You can see an example of the fields in the image below.



Next we update our home page type and add a item selector type field and child content data source.



Once we've updated our models you can create and link promotion objects to your home page by editing the home page.



Finally we edit our home page template to include code that will walk the list of the associated child content (the promotion objects) and render them.

```
<#assign promos = model.promos.item />
```

```
<#list promos as promo>
 <#assign promoItem = siteItemService.getSiteItem(promo.key) />
 <div>
 <img src="${promoItem.image!''}" />
 <div>
 <h1>${promoItem.headline!""}</h1>
 <p>${promoItem.subhead!""}</p>
 </div>
 </div>
 </#list>
```

Looking at the code above we see that what we're doing is getting the key values for the promotional content associated with the home page and the looping over them. For each item we load the actual content from the SiteItemService and then use the item to build out our markup.

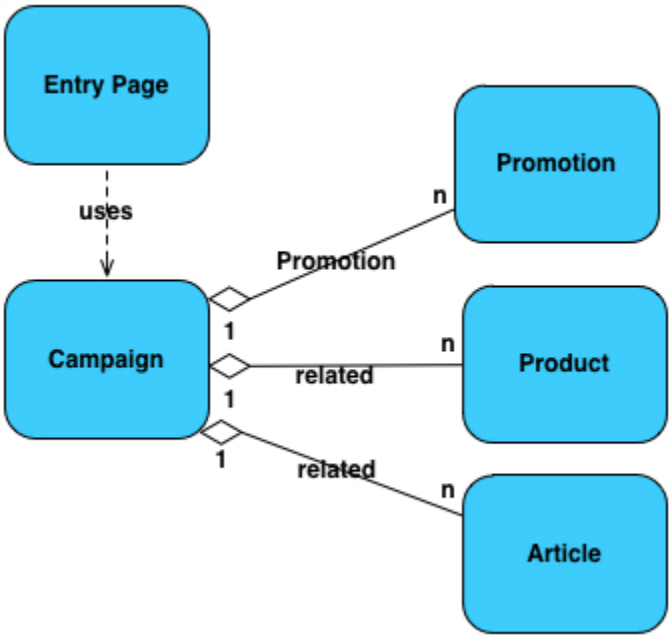
Query Driven Content

Our example of promotional objects on the home page is very simple. It assumes that every user will get the same content and also that authors will not want to stage more than a single scenario at a time. This of course may work well for some organizations but it's far too simplistic for others. Now that we've covered the basic mechanics of author managed content associations let's imagine a use case with much more complex requirements:

- Our authors need to create different variations of the home page that correspond to marketing campaigns which will take place throughout the year.
- Editors choose the content that goes with each campaign.
- A campaign may include a new offering and potentially other associated content that must go out at the same time.
- Campaign content production doesn't necessarily happen in sequence. Each campaign is developed by a different team and may be more or less complex which in turn means campaigns may complete production at vastly different times. Occasionally a content production will complete right before the campaign starts.
- Once a campaign is ready it needs to be approved and scheduled for a deployment. The team assigned to that campaign will move on to the next one.
- When a campaign is over it needs to stop showing up on the home page

For a set of requirements like the ones we see above we're going to rely on a number of features on both in both authoring and delivery.

First we need a content model that is going to support our use case. We need a campaign that can be created independently. The campaign will point to promotional content, products and other relevant content. Our content model might look something like the following:



We can imagine a campaign might have the following properties

- A campaign code that matches a code tracked in our CRM system
- A effective date that indicates when it allowed to begin operating on the site
- A expiration date that indicates when it must end its operation
- A disabled flag that would allow it to be shut down even if it is active
- Associations to all of the content that are related to the campaign. These content items will be selected by the author exactly as described in our earlier use case.

Next we need to update the home page template to use a campaign. Because a new campaign may come along at any time we will use a query to determine which campaign to show. At a minimum our query will look at

- All objects of type campaign
- That are equal to or beyond the effective date
- That are not beyond the expiration date
- That are not disabled.

The template code would look something like the following:

```
<#assign queryStatement = "content-type:/component/campaign " />
<#assign queryStatement = queryStatement + "AND effective_dt:[TODAY TO *] " />
<#assign queryStatement = queryStatement + "AND expiration_dt:[* TO TODAY] " />
<#assign queryStatement = queryStatement + "AND disabled_b:false " />

<#assign query = searchService.createQuery() />
<#assign query = query.setQuery(queryStatement) />
<#assign query = query.setRows(1) />

<#assign campaign = searchService.search(query).response.documents[1] />

<#list campaign.promos as promo>
 <#assign promoItem = siteItemService.getSiteItem(promo.localId) />
 <div>
 <img src="${promoItem.image!''}" />
 <div>
 <h1>${promoItem.headline!""}</h1>
 <p>${promoItem.subhead!""}</p>
 </div>
 </div>
 </#list>
```

In the code above we can see above that we are creating a query statement, limiting our results to a single record and then executing the query. You may recognize that the code after the query which processes the results and shows the promotional content associated to the campaign remains nearly identical to the code in our last example.

Now that we have our mechanics set up teams can begin creating content. Once campaigns are complete they can be queued up for approval and deployment. Because a campaign may require product updates that cannot be shown on the site before the start date content managers may leverage scheduled publishing. Scheduled publishing allows the editorial team to approve content for publish but to delay the deployment of the content until a specific date and time. Finally, editors can be sure all related updates from associated images to associated products will be deployed because Crafter Studio will seamlessly calculate and deploy the campaign's dependencies.

Targeted Content

Targeting content is all about getting the right content to the right audience at the right time. From our last example we've seen how facilitate a complex production process and more importantly for this topic how to add metadata to content objects and then how to use that metadata to dynamically determine which content to show on our site. In essence we've covered the "right content at the right time" part of targeting. Now we want to extend our discretion to a specific audiences. Let's extend our use case from above with a few additional requirements.

- All requirements from the previous section hold.
- Authors may create one or more campaigns that are eligible to be active on the site at any given time
- Campaigns may be specific to different audiences. In other words content may be targeted at that audience. For example: All users, users of a specific gender.

- Campaigns that match more specific criteria for a site user (are more targeted) should be preferred for use over less relevant content.

At this point all of our production mechanics will remain the same. Addressing the new requirements will require the following additional steps:

- We'll add criteria to our campaign definition and then update the content with the appropriate values.
- We'll add additional user specific criteria to our query.
- We'll provide the capability for our authors to review how well the targeting rules work through Crafter Studio.

Let's begin by updating our content model for campaign.

- We'll add a new targeting section to our form definition for campaign called "Targeting"
- To our new section we'll add a grouped check box for Gender and a data source linked to the values: Female, Male, and All

Next we'll update our existing campaigns with the appropriate targeting data. Perhaps we mark all existing items with the All value for the gender field.

Now we can turn our attention to the template and our query. We know we're going to target some content specifically to women, some to men and some to everyone regardless of gender. Let's take a look at the code:

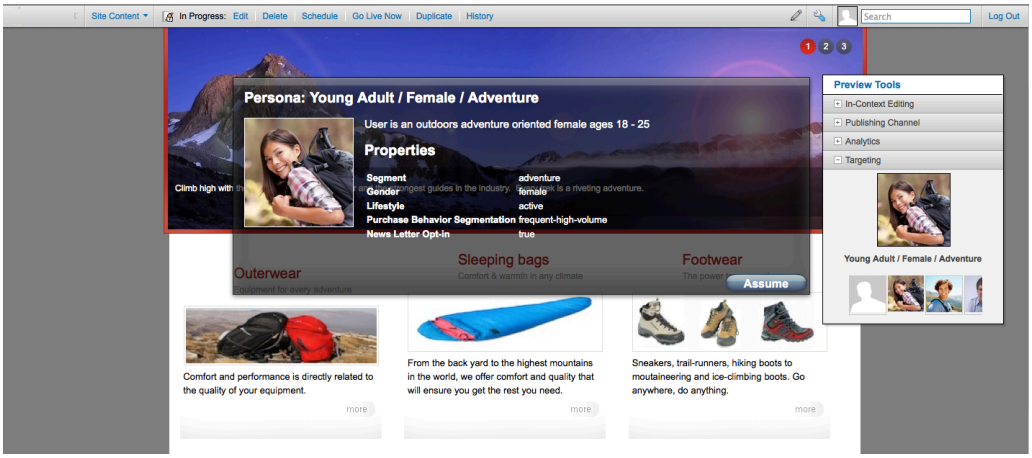
```
<#assign gender = profile["gender"]!"ALL" />

<#assign queryStatement = "content-type:/component/campaign " />
<#assign queryStatement = queryStatement + "AND effective_dt:[TODAY TO *] " />
<#assign queryStatement = queryStatement + "AND expiration_dt:[* TO TODAY] " />
<#assign queryStatement = queryStatement + "AND disabled_b:false " />
<#assign queryStatement = queryStatement + "AND (gender:\"" + gender + "\"^10 OR gender:\"ALL\") />

<#assign query = searchService.createQuery() />
<#assign query = query.setQuery(queryStatement) />
<#assign query = query.setRows(1) />

<#assign campaign = searchService.search(query).response.documents[1] />

<#list campaign.promos as promo>
 <#assign promoItem = siteItemService.getSiteItem(promo.localId) />
 <div>
 <img src="${promoItem.image!''}" />
 <div>
 <h1>${promoItem.headline!""}</h1>
 <p>${promoItem.subhead!""}</p>
 </div>
 </div>
 </#list>
```

In the code above we see two new lines of interest. The first is <#assign gender = profile["gender"]!"ALL" /> This code gets the value for the property gender from the current profile and puts it in to a variable.

The second is line of interest is <#assign queryStatement = queryStatement + "AND (gender:\"" + gender + "\"^10 OR gender:\"ALL\") /> In this line we see we're constraining our query by the gender coming from the profile OR records that have ALL selected. The key here is that we have ^10 on match for the profile's gender. This ensures that any records which match that aspect of the query will be given a higher relevance and thus chosen for display.

As you can see it is quite simple to create solutions for driving dynamic content targeted at specific attributes in a visitor's profile.

However, as we discussed previously, the other half of the issue is enabling the author to test. The reality today is that content authors are creating content for different devices, market segments and all sorts of other facets. With such a dynamic environment they need tools that will help them review and test their work.



Crafter Studio provides targeting tools for authors to address this issue. The authoring environment can be configured with any number of predefined persona(s). A persona is like a profile, in fact it behaves exactly the same way but instead of setting up and signing in as specific users to test different scenarios authors can simply switch back and forth between the available configured persona(s). Each persona has a name, image and a description to help authors identify the scenarios they represent. Authors can also change the property values of a given persona once they have assumed it.