# Rules Based Snippet

**Time to Read:** About 5-10 minutes  
**Intended for:** Sitecore Commerce developers and Sitecore developers  
**Key takeaway:** Creating a custom rendering to dynamically display content based on the Sitecore rules engine.

During a recent Commerce project there was a need to display product specific content on the product detail page. This could be a brand supplied image, extra details, or some kind of ad or special. Since the product data is displayed dynamically I needed to come up with a dynamic way to handle this.

I looked looked at personalization first. While that may work, it's best practice to only use it to display content based on customer behavior. The rules would also quickly become cluttered when mixing visitor rules and content rules together. However the rules engine is a powerful tool so I decided to utilize that with a new custom rendering to meet the requirements.

## Creating the Rules Based Snippet

I decided to use snippets as a starting point. I based it on the snippet rendering to give content authors flexibility on the content they could add. It allowed them to add anything from a simple image rendering to detailed product information using multiple renderings.

I started with making a clone of the Snippet rendering and added only an additional rules field on the data source item. The idea was to use the rules engine to select a snippet data item based on a set of conditions of the product being displayed. The premise was fairly straight forward, I just needed to change the data source item based on the result of the rules engine.

## The GetXmlBasedLayoutDefinition pipeline

When a snippet is rendered it's xml layout definition gets injected into the page items layout definition. To do this the data source item is resolved and its layout definition is read and injected. This occurs in the Sitecore.XA.Feature.Composites.Pipelines.GetXmlBasedLayoutDefinition.InjectCompositeComponents. Here I overrode the ResolveCompositeDatasource method to resolve the data based on the rules field rather than the rendering's data source field.

C# var datasourceItem = Context.Database.GetItem(datasourceId); if(datasourceItem.TemplateID == Templates.RulesBasedSnippetSnippet.ID) { var rulesBasedDatasource = rulesBasedSnippetRepository.GetRulesBasedSnippetDataSource(datasourceItem, contextItem); return rulesBasedDatasource ?? datasourceItem; } else { return datasourceItem; } The change is simple, if the datasource item is a rules based snippet get the item from the rules engine otherwise return the original datasource item. I'll go over rules engine code in the next section.

There is another change to the pipeline that needs to be considered. The resulting layout definition xml is stored in the dictionary cache based on the current context item id. This needed to be adjusted since I needed to return different results for the same page. So I overrode the CreateCompositesXmlCacheKey method to return a different cache key with the url appended only for rules based snippets. C# var pagePath = args.PageContext.RequestContext.HttpContext.Request.Path; if (currentItem.Name == "\*" && renderingIds.Any(r => r == Renderings.RulesBasedSnippet.ID)) { return $"SXA::{Constants.CompositesXmlPropertiesKey}::{siteItem.ID}::{Context.Database.Name}::{Context.Device.ID}::{Context.Language.Name}::{currentItem.ID}::{pagePath}"; } else { return $"SXA::{Constants.CompositesXmlPropertiesKey}::{siteItem.ID}::{Context.Database.Name}::{Context.Device.ID}::{Context.Language.Name}::{currentItem.ID}"; } ## Resolving the Data Source Item Finally I needed to resolve the datasource item using the rules engine. Lets start with the code snippet below: ```C# var commerceContextItem = siteContext.CurrentCatalogItem ?? contextItem; var rules = RuleFactory.ParseRules(contextItem.Database, XElement.Parse(rulesBasedSnippetSnippetItem[Templates.RulesBasedSnippetSnippet.Fields.SnippetRules])); var ruleContext = new RuleContext() { Item = commerceContextItem };

if (rules.Rules.Any()) { foreach (var rule in rules.Rules) { if (rule.Condition != null) { var stack = new RuleStack(); rule.Condition.Evaluate(ruleContext, stack);

if (ruleContext.IsAborted)  
 {  
 continue;  
 }  
 if ((stack.Count != 0) && ((bool)stack.Pop()))  
 {  
 rule.Execute(ruleContext);  
 var action = rule.Actions.FirstOrDefault();  
 var snippetId = action is SelectSnippet<RuleContext> ? ((SelectSnippet<RuleContext>)action)?.SnippetId : string.Empty;  
 return !string.IsNullOrEmpty(snippetId) ? contextItem.Database.GetItem(ID.Parse(snippetId)) : null;  
 }  
 }  
 else  
 {  
 rule.Execute(ruleContext);  
 }  
}

}

return null; ``` First, I parsed the rules field xml using the RuleFactory parser. Then, since I needed to execute the rules engine against the product item I retrieved the current catalog item from the commerce site context. I then use that as the rule engine's context item. Finally I looped through and evaluated each rule with the rules engine returning the final result.

SelectSnippet is a custom RuleAction that returns the the selected snippet id from the resolved rule. The SelectSnippetRuleContext is set using a custom macro for setting the snippet. I won't go into details on creating the macro in this post. However you can find tons of resources for creating custom macros online.

```C# public class SelectSnippet : RuleAction where T : RuleContext { public string SnippetId { get; set; }

public override void Apply(T ruleContext)  
{  
 SelectSnippetRuleContext dataSourceRuleContext = ruleContext as SelectSnippetRuleContext;  
 ID result;  
 if (dataSourceRuleContext == null || !ID.TryParse(SnippetId, out result))  
 {  
 return;  
 }  
 dataSourceRuleContext.SnippetId = result;  
}

} ```

## Wrapping up

This solution so far has worked well for our client. It's currently unclear if this solution poses any performance concerns as the number of rule conditions increase. So far it hasn't caused any issues for us with a dozen or so rules. One positive is that it works in conjunction with personalization. So it allows for different rule sets to be selected based on customer behavior.

I hope you have found this information useful. Please feel free to leave a comment if you have any questions.