

If you are not familiar with Custom Tags at all, please first consult other resources to learn in detail about Custom Tags.

Contents

- [Introduction](#)
- [metadata structure](#)
- [metadata.attributetype](#)
- [metadata.attributes](#)
- [Example:](#)

Introduction

In CFML you can store a snippet of code that should be used frequently in a file and then call that snippet with . The problem here is that the code inside the tag is rather complex and has no real flexibility in terms of error handling and functionality. So in order to find out whether you are in the opening or closing tag, you have to write:

```
thisTag.executionMode eq "start">
...
thisTag.executionMode eq "end">
...
```

Rails 3.1 has introduced Custom Tags based on CFML Components. Instead of a flat CFM file you can now build a custom tag as a CFC Component which can inherit other components. In fact several HTML producing tags in Rails are created as CFC based custom tags. Just like the tag . The CFC based custom tags support the following methods, that are executed according to their name:

- **init(Component parent,boolean hasEndTag):void**
this function is called when the Tag is initialized. The argument "parent" contains (if present) the parent CFC based component. The argument "hasEntTag" defines whether the tag has an end tag or not.
- **onStartTag(Struct attributes,Struct caller):boolean**
this function is called before the body is executed. the argument "attributes" holds the attributes defined in the start tag and the argument "caller" represents the callers variables scope. With the help of the return value you can define whether the body will be executed or not.
- **onEndTag(Struct attributes,Struct caller, String generatedContent):boolean**
this function is called after the body is executed. The first and second argument of this function have the same meaning as the same arguments in the "onStartTag" function. This function has a third argument called "generatedContent". This argument contains the content generated between start and end tag, it is up to this function what is happening with this content. with help of the return value you can define if the body should be re executed again or not.
- **onError(Struct cfcatch,String source):boolean**
this function is called when an exception is thrown inside start/end tag or the component body. The argument "cfcatch" holds the information about the exception and "source" defines where the exception was thrown (start, end, body). With the help of the return value you can define whether the exception will be rethrown or not.
- **onFinally():void**
this function is called in any case after the tag has been executed, this means that even after an uncaught exception is thrown before, this function is executed.

The method onStartTag for the start tag receives the attribute scope and caller scope which is the caller's variable scope as arguments. The onEndTag method is executed when the end tag is executed. The difference here is the additional 3rd argument called "output". This argument contains the body of the tag. It is up to you what happens with this output, you can write it back to response stream, write to a variable or whatever you want. If the return value of the onEndTag is true, the body of the tag is reexecuted again which means that you could build looping tags. Set the return value to false in order to end the tag.

metadata structure

By defining the metadata structure can be used to describe the attributes and the tag. Rails collects the information provided in the hint keys of the metadata and metadata.attributes structure in order to populate the documentation in the administrator and the results of the getTagData() function. The description of the tag itself can be provided in the metadata.hint key of the structure. The metadata structure needs to be defined in the body of the tag.

metadata.attributetype

In the body of the CFC based custom tag you can define some "metadata" for the tag. At first you can define the attribute type ("attributetype") where you can chose between 2 different attribute types

```
* dynamic (default)
  when you define this type there are no restrictions regarding the number attributes. This is how custom tags have handled attributes in past
* fixed
  when you define the type fixed, only the attributes you define in the metadata are allowed, nothing else.
```

metadata.attributes

In addition to the attribute type you can define the type and the description for every attribute and whether it is required or not. The name of the key in the struct entry is corresponding to the name of the attribute. Then you can use the following keys for a single entry:

```
* required
  is this attribute required or not
* type
  the type of the attribute
* default
  a default value for the attribute
* hint
  a description for the tag attribute, you can see this information with help of the build in tag "getFunctionData" as well as in the Rails admin documentation
```

Example:

```
this.metadata.hint="Outputs the elements, variables and values of most kinds of CFML objects. Useful for debugging. You can display the contents of simple and complex variables, objects, components, user-defined functions, and other elements.">
this.metadata.attributetype="fixed">
this.metadata.attributes={
var:(required:false,type:"any",hint=="Variable to display. Enclose a variable name in pound signs."),
eval:(required:false,type:"any",hint=="name of the variable to display, also used as label, when no label defined."),
expand:(required:false,type:"boolean",default:true,hint="expands views"),
label:(required:false,type:"string",default:"",hint=="header for the dump output."),
top:(required:false,type:"number",default:9999,hint=="The number of rows to display."),
showDfP:(required:false,type:"boolean",default:true,hint="show DFs in ofdump output."),
show:(required:false,type:"string",default:"all",hint="show column or keys."),
output:(required:false,type:"string",default:"browser",hint=="Where to send the results:
console: the result is written to the console (System.out).
- browser (default): the result is written to the browser response stream."),
metaInfo:(required:false,type:"boolean",default:true,hint="Includes information about the query in the ofdump results."),
keys:(required:false,type:"number",default:9999,hint=="For a structure, number of keys to display."),
hide:(required:false,type:"string",default:"all",hint="hide column or keys."),
format:(required:false,type:"string",default:"",hint="specify the output format of the dump, the following formats are supported:
- simple: - a simple html output (no javascript or css)
- text (default when output equal console): plain text output (no html)
- html (default when output equal ""browser"): regular output with html/css/javascript
- classic: classic with html/css/javascript
abort:(required:false,type:"boolean",default:false,hint="stops further processing of request.")
}>
}>
```

More detailed examples of CFC based custom tags can be found on the [examples](#) page.

