

# Nesting Custom Tags

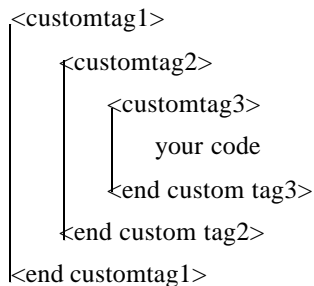
Andrew Cripps  
February 14, 2001

## 1.0 Introduction

Saving development costs by reusing code has been a goal of the computing industry for a long time. With a little forethought you can save time and money developing web systems using ColdFusion by encapsulating functionality in custom tags. This article describes how to nest custom tags. A working knowledge of custom tags is assumed. If you need more information on paired custom tags, see Charles Areharts' article in Volume 3, issue 1 (January 2001).

Any custom tag can be turned into a container for other custom tags. The container is called the parent tag, and the other custom tags are called descendants or child tags. As with any family tree, there can be several levels of nesting, as shown in Figure 1 on page 1.

FIGURE 1. Custom Tag Hierarchy



## 1.1 Example Use of Nested Tags

You may find that you have written a custom tag that need a lot of data. You can make some custom tags a lot easier to use if you turn them into nested tags. For example, consider a custom tag that will build a formatted, numbered list of items for you. You could make a custom tag such as “cf\_buildlist” that would take perhaps two attributes, one a list of styles (e.g. liststyles=“italic,bold,plain,plain,italic”), and the other a list of list items (e.g. listitems=“forests, jungles, oceans, lakes, rivers”).

```
<cf_buildlist liststyles="italic,bold,plain,plain,italic" listitems="forests,jungles,oceans,lakes,rivers" >
```

The cf\_buildlist tag would generate something like:

1. *forests*
2. **jungles**
3. oceans
4. lakes
5. *rivers*

In this custom tag design, your users a lot of responsibility for creating two lists and passing them to the “cf\_buildlist” custom tag. It would be much easier for them if they could specify the formatting for each item at the same time that each list item is specified.

Nested custom tags can make things easier for your users. In this example, you could create a “cf\_buildlist” tag that had a closing tag, and another child tag “cf\_listitem” that accepted list items:

```
<cf_buildlist>
  <cf_listitem item="forests" style="italic">
  <cf_listitem item="jungles" style="bold">
  <cf_listitem item="oceans" style="plain">
  <cf_listitem item="lakes" style="plain">
  <cf_listitem item="rivers" style="italic">
</cf_buildlist>
```

Now users can more easily specify each item and the style for each item. You could extend “cf\_buildlist” to accept other attributes about the list as a whole. For example, you might add an attribute that specifies whether the list is bulleted or numbered.

A good guideline for deciding whether to turn a custom tag into a nested tag is this. If you can identify repeating groups of information in your existing attributes, then you have a candidate for creating parent/child tags.

## 1.2 Parents Getting at Child Data

Listing 1 is an example ColdFusion template that calls a custom tag “cf\_myparenttag” (see listing 2). cf\_myparenttag has a child tag, “cf\_mysubtag” (see listing 3), and cf\_mysubtag has its own child tag, “cf\_mysubsubtag” (see listing 4). In listing 1, lines 16 through 26 are the call to the cf\_myparenttag. Lines 19-23 are the call to the cf\_mysubtag.

CF\_ASSOCIATE is used to allow parent tags to get at the attributes defined for child tags. Let’s say you have a custom tag called “cf\_myparenttag” that calls a child tag “cf\_mysubtag”. You want cf\_myparenttag to be able to access the attributes defined for cf\_mysubtag. The way you do this is by using CF\_ASSOCIATE in cf\_mysubtag (see listing 3), and then accessing the attributes in cf\_myparenttag (see listing 2).

Look at listing 3, lines 10-12. Here, we say that if the execution mode is “start” then associate the attributes defined for cf\_mytag with the parent tag, cf\_myparenttag. Look now at listing 2, lines 19-25. Here, if the execution mode is “end”, we access the array of structures made available by the cfassociate function, and print out the structure key and its corresponding value.

The output of listing 1 is given in listing 5. You can see the nesting effect, in the printed statements “Before calling myparenttag”, “Before calling mytag”, “Before calling mysubtag”, “After calling mysubtag”, “After calling mytag”, “After calling myparenttag.” All the custom tags have been called in sequence.

### 1.3 Thistag scope

CFML provides functions that give information about the execution and type of custom tags and about the content being generated by the tags. The structure “thistag” provides access to these variables:

- AssocAttrs - associated attributes from child tags.
- ExecutionMode - valid values are "start", "end", and “inactive”
- HasEndTag - used for code validation, it distinguishes between custom tags that have and don't have end tags for ExecutionMode=start. The name of the Boolean value is ThisTag.HasEndTag.
- GeneratedContent - can be processed as a variable.

We have already seen how to use cfassociate to populate the “AssocAttrs” variable. All the other variables are populated for you by ColdFusion. In listing 5, which shows the output of running listing 1, you can see the values for the executionmode and hasendtag variables for each custom tag call. Note that “cf\_mysubtag” is not a paired custom tag there is no corresponding end tag “</cf\_mysubtag>” for the opening “<cf\_mysubtag>” call. The parent tag for cf\_mysubtag is cf\_mytag, and in listing 3, line 16, the variable #thistag.hasendtag# has the value “NO” because there is no end tag.

The variable “thistag.generatedcontent” provides access to the content that is generated by the custom tag. If you code your custom tag so that it produces no output (by using cfsetting or cfsilent for example) you can process the generated content before it is displayed. For example, you could use ColdFusion's string functions to replace a string in the generated content before it is displayed.

### 1.4 Children Getting at Parent data

In the Section 1.2 on page 2 we looked at how a parent tag can access the attributes of its children. In this section, we look at how a child tag can access information about and data from its parent.

Two ColdFusion functions provide information about the execution of custom tags:

- `GetBaseTagList()` and
- `GetBaseTagData(parent tag name)`

**GetBaseTagList** returns a comma-delimited list of execution scopes. For example in listing 2, line 14, the call to `#getBaseTagList()` (when executed by running listing 1) produces: “[CFOUTPUT,CF\_MYPARENTTAG]”. This gives the programmer the information that the current execution scopes are, first, CFOUTPUT, and then the custom tag “cf\_myparenttag”. Now look at listing 3, line 17. Here the call to `#getBaseTagList()` (when executed by running listing 1) produces the list :

```
“[CFOUTPUT,CF_MYSUBTAG,CF_MYPARENTTAG]”
```

Now a programmer could use this information to determine in what context the custom tag is executing. This might be important when the custom tag is included in an IF clause for example. The programmer could check the list generated by `#getBaseTagList#` and determine whether the call the current custom tag was made from within in IF clause.

**GetBaseTagData** returns a structure that contains information about the specified parent tag. Look at `cf_myssubsubtag` (listing 4, lines 21-22). Here there is a call to `#GetBaseTagData()` for the tag “cf\_myssubsubtag” - the parent of `cf_myssubsubtag`. Since the parent we specified was a custom tag, we have access to the `#thistag#` set of variables. Line 22 access the `thistag.executionmode` variable for `cf_myssubsubtag`. `GetBaseTagData` will work for any parent tag. For example, we could add a line at line 23 to say:

```
<cfset myparenttagdata = #getBaseTagData(“cf_myssubsubtag”,1)#>
```

This would give access to all the “thistag” variables for `cf_myssubsubtag`.

Since you know the list of parent tags from `#getBaseTagList#`, you could write code that obtains data for each of parent tag in the list. Note that some parents, such as “CFIF” do not have any data associated.

## 1.5 Summary

Nested custom tags provide a very flexible way to abstract complexity, and make it easy for others to use your code. You can create nested tags easily, but as the complexity of your solution grows, you may find that you need to call on the functions ColdFusion provides to allow parent tags to access attributes defined in child tags, or to allow child tags to access information about their parents. You can control with absolute precision what ColdFusion will display to the user by working with the generated content.