Creating and Using ColdFusion Custom Tags

Charles Arehart
SysteManage
November, 2000

updated 11/14/00

Topics

- Custom Tags: What they are, how they work
- Protection of Variables
- Passing Data to and From Custom Tags
- Creating Return Codes for Custom Tags
- Using Paired

Notice

This presentation was created and last updated in November 2000, well prior to the release of CF5 and its support for User Defined Functions (UDFs).

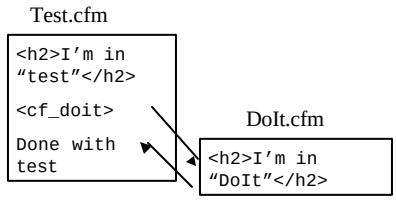
UDFs certainly present yet another way to reuse code, and should be considered separately. The information presented here, however, is still accurate and useful.

Tips and Tricks to be Covered

- Will cover the basics of Custom Tags, for those new to them
- But those already familiar with them may learn a few things not often understood:
  - Why you don't need to pass form, url, session vars, etc. to a custom tag
  - How to provide custom tag return codes
  - How to manipulate data between an opening and closing custom tag pair
  - And more

Custom Tags: What they are, how they work

- Typically used to allow reference to reusable code
- Somewhat equivalent to subroutines, functions, or procedures in other languages
  - One template calls another, to perform some action
- *Called* program referred to as a "custom tag"
  - Term comes from the approach typically used to call them, <cf_*template*>



Result to user running test.cfm

I'm in "test"

I'm in "DoIt"

Done with test

Custom Tag Location

- Can place custom tag template in same directory as caller
  - So it is accessible only by templates in that directory
- Or can place custom tag in special "globally accessible" directory, c:\cfusion\customtags
  - So it is accessible to all templates on server
- Using the syntax we've seen, <cf_*tagname*>, ColdFusion looks for custom tag first in directory of caller, then in global directory
- Can even store and call a custom tag in another directory relative to the caller, or relative to the webroot, using CFMODULE
  - beyond the scope of this class to discuss

Why are they useful?

- That simple example doesn't do justice
- Custom Tags are typically much more useful
  - Can relieve entering code repetitively in several templates
  - Can encapsulate complicated code to be reused often
  - Can hide complexity of solving a given problem
  - Can help segregate business rules from display and page processing logic
  - Can provide solutions to common problems
    - offered for all CF developers at Allaire's Developer's Exchange

How are they different from subroutines and functions?

- ColdFusion has no notion of subroutines or user defined functions
  - As are offered in many other languages
  - Such reusable code is typically embedded within the program making the call
- Calls to a Custom Tag always call another program
  - Typically a ColdFusion template
  - Can be C++ routines
    - Beyond scope of this seminar to discuss

Similarity: Variables are Protected

Test2.cfm

```
<h2>I'm in
"test2"</h2>
<cfset x=1>
<cf_doit2>
Done with
test2
```

DoIt2.cfm

```
<h2>I'm in
"DoIt2"</h2>
<cfoutput>#x#
</cfoutput>
```

Result to user running test2.cfm

I'm in "test2"

CF ERROR
#x# is undefined
in DoIt2.cfm

- Like most languages' subroutines and user-defined functions
  - Access to variables is restricted between caller and custom tag
    - Custom tag cannot refer to local variables created in calling routine
    - Attempt will lead to error
  - Will learn how to pass caller data to custom tag

Likewise: Caller is Also Blind

- Variables created within custom tag cannot be viewed or manipulated by the calling routine
  - Protects variables in caller from being overwritten by custom tag
  - Frees developer of calling template from worry
    - Custom tag is a "black box"
  - May want to return data intentionally
    - Will show later

Test3.cfm

```
<h2>I'm in
"test3"</h2>
<cfset x=1>
<cf_doit3>
Done with
test3
<cfoutput>
<br>x=#x#
</cfoutput>
```

DoIt3.cfm

```
<h2>I'm in
"DoIt3"</h2>
<cfset x = 2>
```

Result to user running test3.cfm

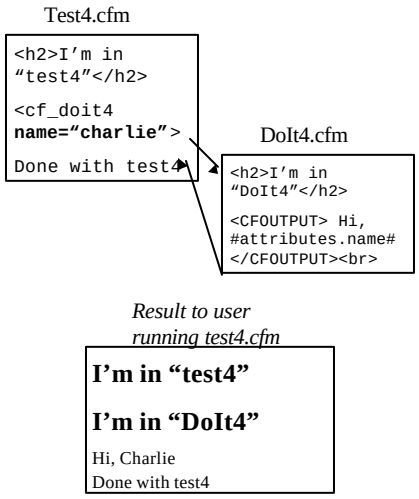
I'm in "test3"

I'm in "DoIt3"

Done with test3
x=1

Note x=1, not 2

Passing Data To a Custom Tag



- For custom tag to perform operation based on data in caller, data must be passed to it
  - Passed in as *attribute*=*value* pairs
  - Any number of pairs may be passed
  - Order, case is not significant
- Data is accessible within custom tag using "attributes." prefix
  - One attributes.*varname* variable for each *attribute*=*value* pair passed

What Can Be Passed To Custom Tag?

- Any ColdFusion element can be passed to a custom tag, including:
  - String
  - Number
  - Variable
  - Array
  - Structure
  - Query
  - Etc.

Passing a Variable to a Custom Tag

Test5.cfm

```
<cfset
fname="charlie">
<cf_doit5
name="#fname#">
```

DoIt5.cfm

```
<CFOUTPUT> Hi,
#attributes.name#
</CFOUTPUT><br>
```

Result to user running test5.cfm

Hi, Charlie

- Passing a variable to a custom tag is straightforward
- Best practice:
  - Surround variable with pound signs *within quotes*
- Note the attributes.*variable* name
  - It's "name", not "fname", in example

Passing a Structure

- Passing a structure, or even a query, to a custom tag is also easy
  - Pass it like any other variable
- Two modest challenges
  - Note the attributes.*variable* name
    - It's "person", not "emp", in example
  - Also, note use of structure.*keyname* notation in attributes.*variable* references

Test6.cfm

```
<cfset emp = structnew()>
<cfset
emp.fname="charlie">
<cfset
emp.lname="arehart">
<cf_doit6 person="#emp#">
```

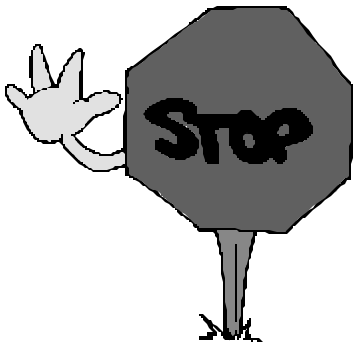
DoIt6.cfm

```
<CFOUTPUT> Hi,
#attributes.person.fname#
#attributes.person.lname#
</CFOUTPUT><br>
```

Result to user running test6.cfm

Hi, charlie arehart

What Variables Need NOT Be Passed?



- **No need to pass these to a custom tag, as they're always available!**
  - Variables passed "to" the caller itself:
    - URL, FORM
    - CGI, COOKIE, etc.
  - "Global" variables:
    - Session
    - Client
    - Application
    - Server
    - Request
- Only local variables (and structures and queries) created in caller need to be passed

Overriding Variable Protection

- Can override variable protection
- Custom tags *can indeed* read local variables created in caller
  - Even if not passed
- Special "caller." prefix removes protection of variable
  - Allows custom tag to refer to variables created in "caller"
- Should use carefully
  - Removes "black box" characteristic

Test7.cfm

```
<cfset x=1>
<cf_doit7>
```

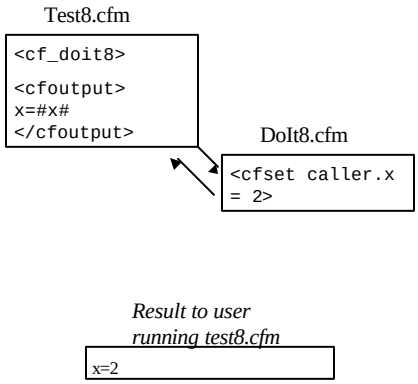
DoIt7.cfm

```
<cfoutput>
x=#caller.x#
</cfoutput>
```

Result to user running test7.cfm

```
x=1
```

Using “Caller.” to Return Data



- Similarly, custom tag can even *write* to a local variable created in caller
  - Or create a new one!
- Using “caller.” prefix for assignment creates variable intended to be seen by caller
  - Refer to this in caller just as if variable was created in caller
- Again, can be risky
  - But a good use is to return a result variable or “return code” from a custom tag

Creating Return Codes

- Can use “caller.” scope to create return codes from custom tags
  - Good practice, furthers similarity to subroutines and user-written functions
  - In previous example, variable “x” could have been named “rc” or “retcode”, etc.
- Be careful choose name that doesn’t overwrite a variable existing in the caller
  - One solution is to use a very unique name, such as caller.*ctname_variablename*
  - Yet another approach is to use CF’s flexibile “object.property” notation for variables
    - Consider using caller.*ctname.variablename*
    - Don’t even need to create a *ctname* structure first to use this approach

Passing Returncode Name on Call

- Yet another approach is to design custom tag to allow caller to specify name of returncode
  - Provides greatest flexibility to caller
  - Caller can choose name it will expect returned
- Requires just a little more programming effort in custom tag
  - Gets into issues of supporting optional parameters
  - As well as creating variable names dynamically
- Covered in the Allaire Advanced ColdFusion Development Course

Comparing Custom Tags to Using CFInclude

- Newcomers inevitably wonder about this
  - Both are a way to reuse code
- Should be clear by now, if familiar with CFINCLUDE
  - Custom tags are quite different

CFINCLUDE vs. Custom Tags

CFINCLUDE	Custom Tags
■ pulls code into caller ■ to be executed inline with caller code	■ puts caller on hold ■ executes custom tag code in own scope ■ returns control to caller

- Key differences in Custom Tags
  - Protected variables
  - Passing and returning data
  - Custom tags accessible to all when placed in c:\cfusion\customtags directory

Paired (Start and End) Tags

- One last topic, CF 4.0 introduced notion of paired custom tags, as in:

  <cf_mytag>
  </cf_mytag>

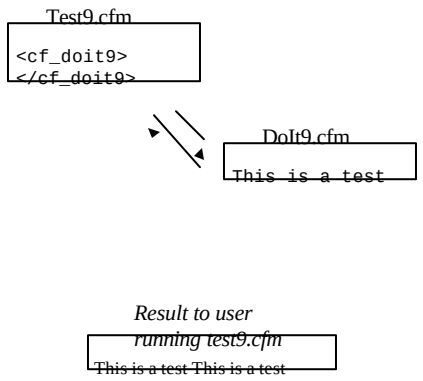
- Two primary benefits to this approach:
  - Ability to process data between the pair
  - Ability to have nested tags within the pair
- Will focus first on passing data between paired tags
  - Will conclude with a look at nested tags

Passing Data Between Paired Tags

- Until now, only way to pass data to custom tag was by way of attributes
- With paired tags, data between tags is "passed" in
  - Available as variable called "thistag.generatedcontent"
- Not quite as straightforward as that
  - Need to understand double execution of custom tag template

Double Execution of Paired Tags

- With paired tag, template actually runs twice
  - Not as silly as it seems
  - But must anticipate this
- Need to further understand another special variable



Thistag.executionmode

Test10.cfm

```
<cf_doit10>
</cf_doit10>
```

DoIt10.cfm

```
In ct:
<cfoutput>
#thistag.executionmode#
</cfoutput><br>
```

Result to user running test10.cfm

```
In ct: start
In ct: end
```

- Variable available inside custom tag whenever paired custom tags are used
  - Has value of “start” when template is being run by way of “opening” tag call
  - Has value of “end” when run by way of “closing” tag call

Thistag.generatedcontent

- Data between tags is available as thistag.generatedcontent
  - available only in “end” mode
- Can use it to process the text between the tags
  - Perhaps store data in db
- Note that the text between the tags is displayed before the processing of closing tag
  - Can stop that

Test11.cfm

```
<cf_doit11>
test
lines
</cf_doit11>
```

DoIt11.cfm

```
<cfif thistag.executionmode is
“end”>
   <br>Storing in DB:
   <!--- cfquery to store
    #thistag.generatedcontent#
   in db --->
</cfif>
```

Result to user running test11.cfm

```
test lines
Storing in DB:
```

Preventing display of the text between paired tags

Test12.cfm

```
<cf_doit12>
test
lines
</cf_doit12>
```

DoIt12.cfm

```
<cfif thistag.executionmode is
“end”>
   Storing:
   <!--- cfquery to store
    #thistag.generatedcontent#
   in db --->
   <cfset
thistag.generatedcontent=“”>
</cfif>
```

Result to user running test12.cfm

```
Storing:
```

- Text between the tags is displayed before executing in end mode
- To prevent this:
  - simply assign an empty string to the special variable during the end mode after processing
  - may not seem logical that assigning it an empty string inside the custom tag will prevent it displaying before the end mode, but it does

Altering the text between tags

- A natural next step
  - manipulate the text between the tags
  - Produce that as the result of calling the custom tag
- Just assign the manipulation of thistag.generatedcontent back to itself
  - That result will be displayed, not the original text

Test13.cfm

```
<cf_doit13>
test
lines
</cf_doit13>
```

DoIt13.cfm

```
<cfif thistag.executionmode is
“end”>
   <cfset
thistag.generatedcontent=ucase(th
istag.generatedcontent)>
</cfif>
```

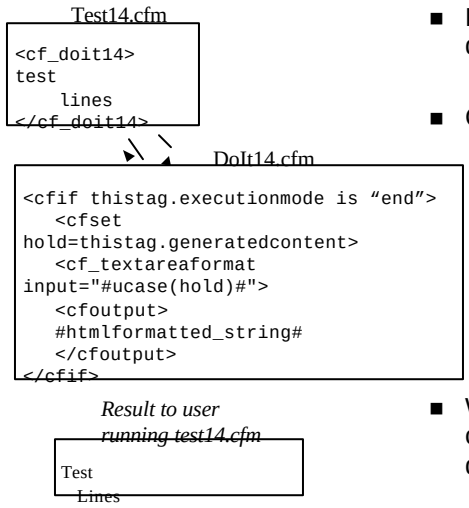
Result to user running test13.cfm

```
TEST LINES
```

Formatting such text for display

- May have noticed the input:
  test
  lines
- Was presented to the user as a single line of output:
  test lines
- This is simply a matter of understanding that HTML ignores line breaks and spaces
- CF's paragraphformat() function might seem the solution
  - but it only converts two line breaks to a <p>
  - It does not convert one line break to a

My Textareaformat custom tag



- Found on the Allaire developer's exchange
  - Solves this problem
- Converts:
  - Single line break to
  - Double line break to <p>
  - Multiple spaces to multiple 's
  - Converts Tabs to 8 's
- Would itself be a good candidate for paired tag design

Thistag.hasendtag

- Finally, thistag.hasendtag is used to determine if a paired tag has a closing tag
  - May want to stop if closing tag not provided
  - May want to operate differently when there is versus when there isn't
    - Probably should also test if generatedcontent is not empty
- Simply has a value of "yes" if there is, "no" if there isn't
  - Available in both start and end mode

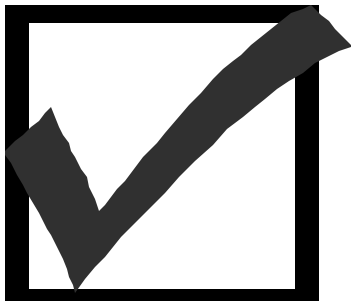
Nested Tags (Parent-Child Tags)

- Nested Tags are an extension of paired tags
- Example:

```
<cf_myparenttag>
   <cf_mychildtag>
   <cf_mychildtag>
</cf_mytag>
```

- Tags executed in order, as with paired tags
  - Can simply use this to segment code into components
- Can also use child tags to gather data to pass to parent's end tag
  - See CFASSOCIATE tag for more information

Summary



- Custom Tags: What they are, how they work
- Protection of Variables
  - Isolated from harm
- Passing Data to and From Custom Tags
  - Attributes and caller scopes
- Creating Return Codes for Custom Tags
  - Setting caller variable
- Using Paired Tags
  - executionmode
  - Generatedcontent
  - hasendtag

Resources

- Allaire Developer's Exchange
  - free and low-cost reusable components
  - Many categories available
- Allaire Advanced CF Course
  - Covers many more aspects
- April &May ColdFusion Developer's Journal
  - "Calling All Custom Tags", parts 1 & 2
  - Covers still more details
- Of course, the Allaire manuals, particularly "Developing Web Apps with CF"
  - Available online within Studio and CF server
- Any of the several CF books now or soon to be on the market

Q&A and evals