User Defined Functions in CF 5: Best Practices and More

Rob Brooks-Bilson  
Web Technology Manager  
Author, Programming  
ColdFusion (O'Reilly)  
03/14/2002

Presented to the MD CFUG  
By Charlie Arehart  
CAREHART@SYSTEMMANAGE.COM  
May 13, 2002

User Defined Functions in CF 5: Best Practices and More

Rob Brooks-Bilson  
Web Technology Manager  
Author, Programming  
ColdFusion (O'Reilly)  
03/14/2002

Presented to the MD CFUG  
By Charlie Arehart  
CAREHART@SYSTEMMANAGE.COM  
May 13, 2002

Agenda

• UDF Overview  
• Best Practices  
• Extending UDFs  
• Gotchas and Additional Considerations  
• Additional UDF Resources  
• Q&A

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

What are UDFs?

• UDF stands for User Defined Function  
• Probably the most requested feature in CF 5.0  
• Lets you extend the core CFML language by encapsulating and abstracting commonly used code  
• They make your life easier

Why UDFs?

• THEY MAKE YOUR LIFE EASIER !!!  
• Provide a cleaner way to call abstracted code than custom tags  
• Return values inline  
• Speed - UDFs are typically faster than similar code written as a custom tag  
• Portability - UDF syntax in ColdFusion is similar to other scripting languages such as JavaScript, ASP, and PHP

UDF’s vs. Custom Tags

• UDF’s  
  – CF 5.0+  
  – Called inline  
  – Takes ordered parameters  
  – Always return a value  
  – Multiple UDFs can be grouped together or included in the same template  
  – UDF’s must be written in CFSCRIPT in CF 5 and can’t contain tags  
  – Will discuss CFMX changes at end of talk  

• Custom Tags  
  – CF 2.0+  
  – Called as tags (CF_) or via CFMODULE  
  – Takes attributes as name=value pairs  
  – Can return one or more variables, or nothing at all  
  – Has it’s own protected memory space  
  – Can use any CFML tag or built in function

Defining UDFs

• UDF’s are defined within a <CFSCRIPT> block  
• Only CFSCRIPT may be used to write UDF’s in CF 5 (this means you can’t include CFML tags within your UDF’s)  
• All UDF’s begin with a function statement  
• All UDF’s must return a value:  
  – Preferably using a return statement  
  – If return statement is omitted, returns value from last expression

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.

Enabling a Microelectronic World

© 2002 Amkor Technology, Inc.
BP1: Naming Considerations

- UDF names must begin with a letter and can contain only letters, numbers and underscores. Same rules as CF variables.
- UDFs cannot have the same names as existing BIF’s (Built In Functions)
- UDF names cannot contain periods
- UDF names cannot begin with CF, CF_, or ColdFusion
- You can’t use the same name for more than one UDF in a template
- Consider the naming conventions used by ColdFusion’s BIFs– ListFoo(), ArrayFoo(), StructFoo(), etc.

Writing a Simple UDF

```cfscript```
// Return the day of the week, starting Monday as 1
function ISODayOfWeek()
  //If the current day is Sunday, return 7 instead of 1
  if (DayOfWeek(Now()) EQ 1)
    Return 7;
  else
    Return DayOfWeek(Now())-1;
```

Calling UDF’s

- UDF’s can be used anywhere you would use a normal (built in) CFML function:
  - Within a `<CFOUTPUT>` block
  - Within `<CFSET>` tags
  - In tag attributes
  - Within other functions
  - Within `<CFSCRIPT>` blocks

Calling Our Simple UDF

```cfscript```
// Return the day of the week, starting Monday as 1
function ISODayOfWeek()
  //If the current day is Sunday, return 7 instead of 1
  if (DayOfWeek(Now()) EQ 1)
    Return 7;
  else
    Return DayOfWeek(Now())-1;
```

```cfoutput```
Today is day #ISODayOfWeek(now)# of the week.
```

Modifying ISODayOfWeek()

```cfscript```
//Modify ISODayOfWeek() to require a parameter
// Return the day of the week, starting Monday as 1
function ISODayOfWeek(date)
  //If the specified day is Sunday, return 7 instead of 1
  if (DayOfWeek(date) EQ 1)
    Return 7;
  else
    Return DayOfWeek(date)-1;
```

```cfoutput```
Today is day #ISODayOfWeek(now)# of the week.
```
UDF's and Variable Scope

- All variables in all scopes are automatically available within UDF's
- A special "function" scope is available to UDF's
  - "function scoped" or "function-local" variables are declared using `var`
  - No special prefix to refer to them
  - They do not overwrite variables outside of the UDF with the same name
  - They are not available outside of the UDF

Failure to var a Variable

```cfscript```
function foo(){
x=5;
return x;
}
```

```cfset x=1```

```cfoutput```
x defined in the UDF is # Foo()#
x defined in the template is #x# (it should be 1)
```

Using "Function Scope" Variables

```cfscript```
// Takes a string and returns it padded with a characters
function PadString(string, char, count){
//set the padding by repeating char count number of times
var Padding = RepeatString(char, count);
//append the padding to the beginning of the string
return Padding & string;
}
```

```cfset x=123```
```cfset y="test```
```cfoutput```
"#PadString (x, 0, 4)#"
"#PadString (y, ",", 5)#"
```

BP2: Always var Your Variables

- Variables declared with `var` must be defined at the top of the function, before any other CFscript statements, and take precedence over any other variable with the same name, regardless of the variable's scope.
- Variables declared with `var` follow the same naming rules as other variables. Additionally, they may not be compound variable names such as `My.Var.Name`.
- Any valid expression can be used to initialize a variable:
  - `var x=1;`
  - `var y="Hello";`
  - `var z=ArrayNew(1);`
- You must always supply an initial value or expression when declaring a variable with `var`. This means you can't do things like `var x`.
- Don't forget to `var` variables used in loops and recursion!

Accepting Optional Parameters

- UDF's can accept both required and optional parameters
  - Any named parameter is automatically required
  - Failing to pass a required parameter results in an error
  - Any additional arguments passed in are considered optional
  - Any number of optional arguments may be passed to a UDF - it is up to the function to deal with them
  - All parameters (named and optional) are available within the UDF in an array called `Arguments`
  - Use `ArrayLen()` to determine the number of parameters passed to a UDF

Handling Optional Parameters

```cfscript```
//Returns a date object representing the next occurrence of the specified day. The default is the next occurrence of the current day.
function NextOccOfDOW()
{
//set the default day and day offset
var day = DayOfWeek(Now());
var dayOffset = 7;
//if a date is passed, set the day to that value
if(ArrayLen(Arguments)) day = Arguments[1];
//if the day is greater than the current day, set the offset to 0 (this week)
if(DayGT DayOfWeek(Now()); dayOffset = 0;
//return the date for the next occurrence of the day
return DateAdd("d",(dayOffset + (day - DayOfWeek(Now()))),Now());
}
```

```cfoutput```

Calling NextOccOfDOW()

<CFOUTPUT>
<CFLOOP INDEX="i" FROM="1" TO="7">
The next occurrence of # DayOfWeekAsString(i)# is
#DateFormat(NextOccOfDOW (i), 'mmmm dd, yyyy')#.<BR>
</CFLOOP>
</CFOUTPUT>

Handling Required & Optional Parameters

//A left() function for lists.  Returns the n leftmost list elements
function ListLeft(list, numElements){
  var tempList = "";
  var i = 0;
  var delimiter = ",";
  if (ArrayLen(arguments) > 2){
    delimiter = arguments[3];
  }
  if (numElements > ListLen(list, delimiter)){
    numElements = ListLen(list, delimiter);
  }
  for (i = 1; i <= numElements; i = i + 1){
    tempList = ListAppend(tempList, ListGetAt(list, i, delimiter), delimiter);
  }
  return tempList;
}

Handling Multiple Optional Parameters

//Compares one list against another to find the elements in the first list that don't exist in the second list.
function ListCompare(List1, List2){
  var TempList = "";
  var Delim1 = "",
  var Delim2 = "",
  var Delim3 = "",
  var i = 0;
  switch(ArrayLen(arguments)) {  // Handle optional arguments
    case 3: { Delim1 = Arguments[3]; break; }
    case 4: { Delim1 = Arguments[3]; Delim2 = Arguments[4]; break; }
    case 5: { Delim1 = Arguments[3]; Delim2 = Arguments[4]; Delim3 = Arguments[5]; break; }
  }
  //Add any elements from the full list not found in the partial list to the temporary list
  for (i = 1; i <= ListLen(List1, Delim1); i = i + 1) {
    if (ListFindNoCase(List2, ListGetAt(List1, i, Delim2), Delim3) IS "No"){
      TempList = ListAppend(TempList, ListGetAt(List1, i, Delim1), Delim3);
    }
  }
  return TempList;
}

Calling ListLeft()

<CFSET List="1,2,3,4,5,6,7,8,9,10">
<CFSET List2="a|b|c|d|e|f|g|h">

<CFOUTPUT>
#ListLeft(List, 3)#
#ListLeft(List2, 50, ",")#<BR>
</CFOUTPUT>

Calling ListCompare()

<CFSET FullList = "1,2,3,4,5,6,7,8,9,10">
<CFSET FullList2 = "1,2,3,4,5,6,7,8,9,10">
<CFSET PartialList = "a,b,c,d,e,f,g,h">
<CFSET PartialList2 = "a,b,c,d,e,f,g,h">

<CFOUTPUT>
#ListCompare (FullList, PartialList , ",", ",")#<BR>
#ListCompare (FullList2, PartialList2)
#ListCompare (FullList, PartialList2)
#ListCompare (FullList2, PartialList)
</CFOUTPUT>

BP3: Don’t Forget to ‘Break’ your Case

• In CFScript, case statements must be terminated with break; to avoid “falling through”
  – If break; is missing, all subsequent case statements will execute regardless of whether
    it’s True or False until a break; is reached, or all cases have been executed
• This is similar to other languages such as JavaScript
• This is not required outside of CFScript when using <CFCASE>
Switch/Case without break;

```cfscript```
function QuarterAsString(date)
{ // assign the numeric quarter associated with the passed in date
    var theQuarter = Quarter(date);
    var q = 4;
    // evaluate the quarter and convert to string
    switch (theQuarter) {
        case 1: q = "1st";
        case 2: q = "2nd";
        case 3: q = "3rd";
        default: q = "4th";
    } return q;
}
```

```cfset```
TheDate = "01/01/2002"
```cfoutput```
#MonthAsString(Month(TheDate))# is in the #QuarterAsString(TheDate)# quarter of the year.
```cfoutput```

Switch/Case with break;

```cfscript```
function QuarterAsString(date)
{ // assign the numeric quarter associated with the passed in date
    var theQuarter = Quarter(date);
    var q = 4;
    // evaluate the quarter and convert to string
    switch (theQuarter) {
        case 1: { q = "1st"; break; }
        case 2: { q = "2nd"; break; }
        case 3: { q = "3rd"; break; }
        default: q = "4th";
    } return q;
}
```

```cfset```
TheDate = "01/01/2002"
```cfoutput```
#MonthAsString(Month(TheDate))# is in the #QuarterAsString(TheDate)# quarter of the year.
```cfoutput```

BP4: By Value vs. by Reference

- Different data types are passed to UDFs in different ways:
  - Strings, numbers, date/time values, and arrays are passed by value (copy)
  - Structures, queries, and objects (COM, CORBA, and Java) are passed by reference (pointer)
  - Be careful of "complex combos". If you pass an array of structures, the array is passed by value while the structures are passed by reference.
  - Use Duplicate() if you need to make a copy of a variable that is passed by reference
    - Otherwise, change to parameter inside UDF will change the original that was passed as a parameter

BP5: Recursion in UDFs

- UDF's can be called recursively, meaning that a UDF can call itself
- Too much recursion can be a bad thing
  - Processor/memory intensive
  - Can drain resources to the point of crashing the server
  - Max is ~800 levels deep
  - Consider looping instead of recursion where appropriate

Writing a Function Using Recursion

```cfscript```
// Returns the factorial (n!) of a positive integer (ie. 5! = 5*4*3*2*1)
function Factorial(integer)
{ if (integer LE 1) return 1;
    else return integer * Factorial(integer - 1);
}
```

```cfset```
N = 5
```cfoutput```
Given n=5
n! is #Factorial(N)#
```cfoutput```

Looping Instead of Recursion

```cfscript```
// Returns the factorial (n!) of a positive integer (ie. 5! = 5*4*3*2*1)
function Factorial(integer)
{ var theFactorial = 1;
    while (integer GT 0) {
        theFactorial *= integer--;
    } return theFactorial;
}
```

```cfset```
N = 5
```cfoutput```
Given n=5
n! is #Factorial(N)#
```cfoutput```
BP6: Managing UDF's

- Consider creating UDF Libraries
  - Group related functions into individual CFM files and include them in your applications as needed with <CFINCLUDE>
  - Using <CFINCLUDE> results in very little overhead
  - If certain function libraries are used throughout your application, consider placing the <CFINCLUDE> within your Application.cfm template.

Sample UDF Library

```cfc
<CFSCRIPT>
function FahrenheitToCelsius (fahrenheit) {
  Return (100/(212-32) * (fahrenheit - 32));
}
function CelsiusToFahrenheit (celsius) {
  Return ((212 -32)/100 * celsius + 32);
}
function CelsiusToKelvin (celsius) {
  if (celsius < -273.15)
    Return -1;
  else
    Return celsius + 273.15;
}
</CFSCRIPT>
```

BP7: UDF's and Custom Tags

- UDF's are not automatically available inside of Custom Tags.
  - When the custom tag is called from page declaring UFDs
    - Two options here:
      - Code/include the UDF directly within the custom tag
        § Keeps the tag portable
        § Possibility of maintaining multiple copies
      - Assign the UDF to the Request scope
        § Simplifies management
        § Not as clean to call

Copying a UDF to the Request Scope

```cfc
<CFSCRIPT>
function AreaCircle (radius) {
  return Pi()*(radius^2);
}
//Copy the UDF to the Request scope
Request.AreaCircle = AreaCircle;
</CFSCRIPT>
```

In a custom tag called from this template, to call the AreaCircle() function in the Request scope, you would use the following code:

```cfm
<P>
<CFOUTPUT>
The area of a circle with a radius of 3 is #Request.AreaCircle(3)#</CFOUTPUT>
</P>
```

BP8: UDF's and Persistent Variable Scopes

- Although it is possible to assign UDF's to persistent variable scopes (Application, Session, Server), it is generally advisable not to do so in CF 5 due to locking considerations and performance implications.
  - An Exclusive lock must be placed around the code where the UDF is copied to the Server scope
  - A ReadOnly lock must be placed around any calls to the persistent UDF
  - Don’t forget that an Application.cfm template must be defined with the appropriate variable type(s) enabled before you can copy a UDF to the Application or Session scope
Copying a UDF to a Persistent Scope: Method 1

```cfscript
<cflock scope="Server" type="Exclusive" timeout="5">
<cfscript>
functionAreaCircle(radius) {
    return Pi()*(radius^2);
}
//Copy the UDF to the Request scope
Server.AreaCircle = AreaCircle;
</cfscript>
</cflock>
```

In a custom tag called from this template or indeed in any other template on entire server, to call the `AreaCircle()` function in the Server scope, you would use the following code:

```cflock scope="Server" type="ReadOnly" timeout="5">
<cfoutput>
The area of a circle with a radius of 3 is #Server.AreaCircle(3)#
</cfoutput>
</cflock>
```

Copying a UDF to a Persistent Scope: Method 2

```cfscript
<cflock scope="Server" type="Exclusive" timeout="5">
<cfset Server.AreaCircle = AreaCircle>
</cflock>
```

In a custom tag called from this template or indeed in any other template on entire server, to call the `AreaCircle()` function in the Server scope, you would use the following code:

```cflock scope="Server" type="ReadOnly" timeout="5">
<cfoutput>
The area of a circle with a radius of 3 is #Server.AreaCircle(3)#
</cfoutput>
</cflock>
```

Extending UDFs

- Use COM, Java, or CORBA to extend UDF capabilities
  - In following slides, COM example will work only on Windows systems
  - Java example will work only if CF 5 has been properly configured to support Java
    - CFMX executes Java programs w/out need to configure
  - Objects are called using the `CreateObject()` BIF

Example: Obtaining the Drive Size via COM

```cfscript
function FreeSpace (drvPath) {
    Var fso = CreateObject("COM","Scripting.FileSystemObject");
    Var drive = fso.GetDrive(drvPath);
    Return drive.FreeSpace;
}
```

```cfoutput
Free space available on C: # NumberFormat(FreeSpace("c:"))# bytes
</cfoutput>
```

Example: Reading a Text File with Java

```cfscript
/**
* Reads a file.
* 
* @param file The filename to read.
* @param from The line number specifying where to begin reading.
* @param to The line number specifying where to stop reading.
* @param NL Character to use for newlines. Defaults to Chr(13) Chr(10)
* @return Returns a string.
* @author Raymond Camden (ray@camdenfamily.com)
* @version 1, December 3, 2001
*/
function FileRead(filename) {
    var fileStr = "";
    var fileReaderClass = createObject("java","java.io.FileReader");
    var fileReader = fileReaderClass.init(filename);
    var lineNumberReaderClass = createObject("java","java.io.LineNumberReader");
    var lineReader = lineNumberReaderClass.init(fileReader);
    var notDone = true;
    var lastLine = 0;
    var thisLine = 0;
    var NL = Chr(13) & Chr(10);
    var from = 0;
    var to = 0;
    var line = "";
    if(arrayLen(arguments) gt 1) from = arguments[2]; //optional FROM
    if(arrayLen(arguments) gt 2) to = arguments[3]; //optional TO
    if(arrayLen(arguments) gt 3) NL = arguments[4]; //optional NL
    if(not fileExists(filename)) return "";
    while(notDone) {
        line = lineReader.readLine();
        thisLine = lineReader.getLineNumber();
        if((from is 0 OR thisLine gte from) AND (to is 0 OR thisLine lte to)) fileStr = fileStr & line & NL;
        if(thisLine is lastLine) notDone = false;
        lastLine = thisLine;
    }
    return fileStr;
}
```

Example: Obtaining the Drive Size via COM

```cflock scope="Server" type="ReadOnly" timeout="5">
<cfoutput>
The area of a circle with a radius of 3 is #Server.AreaCircle(3)#
</cfoutput>
</cflock>
```

Example: Reading a Text File with Java

```cflock scope="Server" type="ReadOnly" timeout="5">
<cfoutput>
#x#
</cfoutput>
</cflock>
```
Gotchas and Additional Considerations

- **DO NOT FORGET TO VAR YOUR VARIABLES**
- You cannot loop over COM collections with CFSCRIPT in CF 5
- Exception handling can be done using TRY/CATCH with `<CFCAUGHT TYPE="Expression">` in CF 5
- ColdFusion MX (Neo) will include expanded UDF support

Some UDF Changes in CFMX

- CFMX will add new CFFUNCTION tag
  - Will work like UDF, but allows tags inside function
- Can even create “function-scooped” variables:
  - `<CFSET VAR=x5>`
- Not working in beta or RC, but planned for final version
- UDFs will no longer have to return a value
  - Same for CFFUNCTION routines
- CFMX adds try/catch within CFSCRIPT
  - Useful within UDFs, of course
- CFMX performs automatic locking
  - Reducing need for some CFLOCKS
  - May reduce argument against copying UDFs to persistent scope
- CFMX code is compiled
  - Speed issues reduced

CFFUNCTION

```cfscript
<cffunction name="sayHello">
  Hello World!
</cffunction>
```

```cfscript
<cffunction name="sayHello2">
<cfreturn "Hello World!">
</cffunction>
```

```cfscript
<cffunction name="sayHello3">
<cfmail to="..." from="..." subject="...">Hello World!</cfmail>
</cffunction>
```

```cfoutput
#sayHello()#
<br>
#sayHello2()#
<br>
#sayHello3()#
```

External Resources

- Common Function Library Project at http://www.cflib.org - the largest repository for open-source UDF’s (currently over 400)
- Making More of UDFs (Raymond Camden), http://www.sys-con.com/coldfusion/article.cfm?id=397
- What’s so Great about User Defined Functions? (Tim Buntel), http://www.allaire.com/handlers/index.cfm?id=20763
- Extending ColdFusion Pages with CFML Scripting chapter in the Developing ColdFusion Applications documentation (included with CF 5.0)

External Resources - UDFDoc

- “Standardized” way to document UDFs
- Based on JavaDoc
- Tells what the function does as well as what parameters it takes, and what value(s) it returns
- Custom tag for auto-generation available from www.cflib.org
Q&A

- Feel free to email me questions at rbils@amkor.com
- This presentation is available at www.cflib.org under the Resources section