Working with Application.CFC

By Raymond Camden

ColdFusion MX 7 added a number of really cool features to the ColdFusion developer's toolkit. One of the most powerful of these was support for a CFC-based approach to application management. Prior to CFMX 7, ColdFusion applications were (typically) defined by the existence of a CFAPPLICATION tag, usually contained within an Application.cfm file. While this worked well enough, there were a lot of things that we either couldn't do or couldn't do easily with these tools. For example, setting memory variables once and only once usually involved a check for existence, a lock, and another check, or other strange coding techniques.

The Application.cfc feature changes all of that. It improves our options for designing ColdFusion applications. In this article, I will show you how to leverage ColdFusion MX 7's new event-driven methods in your own Application.cfc. We will build a new application and slowly "grow" the Application.cfc file along with the features we add. You may want to follow along, and have your CFMX 7 server up and running along with a good editor. (CFEclipse, of course!)

What is an Application?

ColdFusion defines an application as a collection of files that all contain a CFAPPLICATION tag with the same name attribute. In the "old days," the easiest way to make sure the CFAPPLICATION tag was part of each file was to place the tag within an Application.cfm, which ColdFusion automatically prepends to any page called in the same and any subdirectories. ColdFusion automatically runs the Application.cfm file and creates the *Application* scope, and potentially a *Session* and *Client* scope. The data is unique to and contained within that particular application. The code for this would look like:

```
<cfapplication name="myApplication" sessionManagement=true>
```

As long as you didn't have any other CFAPPLICATION tags, all the files in the folder containing the Application.cfm file, and in any subfolder, would all be considered one application.

So how did we typically use the Application.cfm file? Most of us used it as an "auto-include." In other words, since we knew it was always run, we would use it to:

1) Initialize application variables like DSN and the administrator's email.
2) Include user-defined functions for other pages to use.
3) Control what happens when an error occurs with the CFERROR tag.
4) Add security to the application, or to parts of the application.

Let's take a quick look at a simple, old school Application.cfm file:

Con't on next page

Listing 1: Example1.cfm

```
1   <!--- define application --->
2   <cfapplication name="oldSchool" sessionManagement=true>
3   <!--- initialize application variables --->
4   <cfif not isDefined("application.init") or isDefined("url.reinit")>
5     <cfset application.dsn = "foo">
6     <cfset application.adminemail = "ray@camdenfamily.com">
7     <cfset application.init = true>
8   </cfif>
9   <!--- load UDFs that my pages may need --->
10  <cfinclude template="udfs.cfm">
11  <!--- If something goes wrong, run this file and blame the management --->
12  <cferror type="exception" template="error.cfm">
13  <!--- secure the admin folder --->
14  <cfif findNoCase("/admin", cgi.script_name) and not
        isDefined("session.loggedIn")>
15    <cfinclude template="login.cfm">
16    <cfabort>
17  </cfif>
```

I won't spend a lot of time on this file because it represents the old way of coding, but it does provide an example of all four uses of the Application.cfm. (Of course, you can do more with Application.cfm, but this resembles what most of us are doing out there in the real world.) Now let's look at how this code can be rewritten in the new Application.cfc format. I like to start simple and expand piece by piece. So the first version will have a grand total of two lines:

Listing 2: Application.cfc (Version 1)

```
1   <cfcomponent output="false">
2   </cfcomponent>
```

Woohoo! We have a file that does absolutely nothing! (But boy, does it run fast.) You will want to save this file in a folder by itself somewhere under your web root. It doesn't really matter what you name the folder.

The next thing you will probably want to do is configure the application. In the past, this would all have been done with attributes in the CFAPPLICATION tag. In our component-based Application.cfc file, we simply use the *This* scope. Let's add a few more details to the application:

Listing 3: Application.cfc (Version 2)

```
1   <cfcomponent output="false">
2     <cfset this.name = "OurApplication">
3     <cfset this.applicationTimeout = createTimeSpan(0, 2, 0, 0)>
4     <cfset this.clientManagement = false>
5     <cfset this.sessionManagement = true>
6   </cfcomponent>
```

So what did I do here? I named the application, set a specific application timeout, turned off client management, and turned on session management. In the past, this would have been:

```
<cfapplication name="OurApplication" applicationTimeout="#createTimeSpan
  (0, 2, 0, 0#" clientManagement="false" sessionManagement="true">
```

While the CFC version has more lines, I find it a bit easier to read. We should quickly check to make sure everything is working OK, so let's create a simple index.cfm. This should be saved in the folder where you saved the Application.cfc file.

Listing 4: index.cfm

```
1    <cfdump var="#application#" label="Application Scope">
```

If you run the index file, you should see a dump of the application scope. Since we didn't actually put anything in it, you should just see the application name.

The onApplicationStart Method

Next let's look at how you would add application variables. The "old style" code would do something like this:

```
<cfif not isDefined("application.init")>
    ... lots of application variables ...
    <cfset application.init = true>
</cfif>
```

This works, but it is a bit of a hack. It would be nice if there was a way to have code that would only run once, when the application started. Luckily, we can do that by simply adding the onApplicationStart method to the Application.cfc file. Here is the next version of the Application.cfc file:

Listing 5: Application.cfc (Version 3)

```
1    <cfcomponent output="false">

2      <cfset this.name = "OurApplication">
3      <cfset this.applicationTimeout = createTimeSpan(0, 2, 0, 0)>
4      <cfset this.clientManagement = false>
5      <cfset this.sessionManagement = true>

6      <cffunction name="onApplicationStart" returnType="boolean"
           output="false">
7        <cfset application.dsn = "foo">
8        <cfset application.adminEmail = "ray@camdenfamily.com">
9        <cfreturn true>
10     </cffunction>

11   </cfcomponent>
```

The method we added, onApplicationStart, will be automatically called by ColdFusion the first time a user hits your application. Notice there is no isDefined or wasted application variable to check. I just set the application variables for my site and I was done with it. Also, notice that the method returned true. If you return false, the application won't start. Why would you ever return false? Well, you might write some code that checks to make sure the datasource is working. If not, you could fire off an email to the administrator and then return false.

The onSessionStart Method

An Application.cfm is also commonly used to initialize *Session* variables, typically using the same type of logic. The code checks for some *Session* variable used as a marker, sets some variables, then sets the marker. Once again, we can do this a lot easier by adding the onSessionStart method to the Application.cfc file. Here is version 4:

Listing 6: Application.cfc (Version 4)

```
1   <cfcomponent output="false">
2     <cfset this.name = "OurApplication">
3     <cfset this.applicationTimeout = createTimeSpan(0, 2, 0, 0)>
4     <cfset this.clientManagement = false>
5     <cfset this.sessionManagement = true>
6     <cffunction name="onApplicationStart" returnType="boolean"
          output="false">
7       <cfset application.dsn = "foo">
8       <cfset application.adminEmail = "ray@camdenfamily.com">
9       <cfreturn true>
10    </cffunction>
11    <cffunction name="onSessionStart" returnType="void" output="false">
12      <cfset session.started = now()>
13      <cfset session.numberofpagesvisited = 0>
14    </cffunction>
15  </cfcomponent>
```

This method is much like the onApplicationStart code. I set a few random *Session* variables and I'm done. But unlike onApplicationStart, the onSessionStart method doesn't return a value.

The onRequestStart Method

So far we've covered how to handle creating variables at startup. Now let's look at how we can load templates that contain information that we want associated with the page. At the beginning of each request, ColdFusion will look for a method named onRequestStart. We can use this method to load a template containing all of our user-defined functions (UDFs). Here is the next version of our Application.cfc file:

Listing 7: Application.cfc (Version 5)

```
1   <cfcomponent output="false">
2     <cfset this.name = "OurApplication">
```

```
4       <cfset this.applicationTimeout = createTimeSpan(0, 2, 0, 0)>
5       <cfset this.clientManagement = false>
6       <cfset this.sessionManagement = true>
7       <cffunction name="onApplicationStart" returnType="boolean"
           output="false">
8          <cfset application.dsn = "foo">
9          <cfset application.adminEmail = "ray@camdenfamily.com">
10         <cfreturn true>
11      </cffunction>
12      <cffunction name="onRequestStart" returnType="boolean" output="false">
13         <cfargument name="thePage" type="string" required="true">
14         <cfinclude template="udfs.cfm">
15         <cfreturn true>
16      </cffunction>
17      <cffunction name="onSessionStart" returnType="void" output="false">
18         <cfset session.started = now()>
19         <cfset session.numberofpagesvisited = 0>
20      </cffunction>
21   </cfcomponent>
```

When the onRequestStart method is called, ColdFusion passes it an argument that contains the script name of the file being requested. You could use this for security and check for an admin folder. If it exists and the user isn't logged on, you could forward the user to a login page. Or you can ignore it, as I did! You don't need to do anything at all with it. Like the onApplicationStart method, the returnType is boolean and if you return false, the request will abort. All I did in the method was include a file of UDFs.

There is something you should know. If you set a variable in the onRequestStart method, it will *not* be available in the page. I know; this is confusing. (And it gets even more confusing when you use the onRequest method, which is why I'll cover it at the end.) So why bother including the UDFs file at all? In my case, I used the *Request* scope for my UDFs. Here is udfs.cfm:

Listing 8: udfs.cfm

```
1   <cfscript>
2      request.udf = structNew();
3      function doNothing() {
4      return "";
5      }
6      request.udf.doNothing = doNothing;
7    </cfscript>
```

Most folks use UDFs by including them in the *Request* scope as I did here. The only difference is that I copied the UDFs to a structure. This has two benefits. First, it means that when onRequestStart loads the file, I'll have access to the UDFs in my pages. I'll just need to use the *Request* scope. The second benefit is that if I use any custom tags that need the UDFs, they will be accessible from the *Request*-scoped structure, especially if I have deeply nested tags. So

remember to use a non-*Variables* scope if you want to use the onRequestStart method to initialize some variables for your pages.

The onError Method

So far we've covered everything from our list except error handling. If you want, you can simply use the cferror tag in your Application.cfc file. That is perfectly fine. But if you really want to move into the future, use the onError method. As you can guess, this error handler runs whenever an exception that is not captured within a cftry/cfcatch is thrown by your site's code. Let's look at the latest version of our Application.cfc file:

Listing 9: Application.cfc (Version 6)

```
1   <cfcomponent output="false">
2     <cfset this.name = "OurApplication">
3     <cfset this.applicationTimeout = createTimeSpan(0, 2, 0, 0)>
4     <cfset this.clientManagement = false>
5     <cfset this.sessionManagement = true>
6     <cffunction name="onApplicationStart" returnType="boolean"
          output="false">
7       <cfset application.dsn = "foo">
8       <cfset application.adminEmail = "ray@camdenfamily.com">
9       <cfreturn true>
10    </cffunction>
11    <cffunction name="onRequestStart" returnType="boolean" output="false">
12      <cfargument name="thePage" type="string" required="true">
13      <cfinclude template="udfs.cfm">
14      <cfreturn true>
15    </cffunction>
16    <cffunction name="onSessionStart" returnType="void" output="false">
17      <cfset session.started = now()>
18      <cfset session.numberofpagesvisited = 0>
19    </cffunction>
20    <cffunction name="onError" returnType="void" output="false">
21      <cfargument name="exception" required="true">
22      <cfargument name="eventname" type="string" required="true">
23      <cfmail to="#application.adminEmail#" from="#application.adminEmail#"
            subject="Error!" type="html">
24        <cfdump var="#exception#" label="Exception">
25        <cfdump var="#cgi#" label="cgi">
26        <cfdump var="#form#" label="form">
27      </cfmail>
28      <cflocation url="error.cfm" addToken="false">
29    </cffunction>
30  </cfcomponent>
```

When the onError method is called (line 20), two arguments are passed to it. The first is the exception object. The second is the name of the event that threw an error, if and only if it was another event in the Application.cfc file. I've done nothing special here except to email the administrator some details about both the error and the current request.

Note the cflocation tag on line 28. Why did I add that? When you use cferror and an exception occurs, the layout that had already been displayed is cleared. When the onError method is called, any previously-generated HTML isn't cleared. Using cflocation will let you present a clean error page to the user.

The onApplicationEnd Method

Still with me? So far, we haven't done anything that couldn't be done in an Application.cfm file. Now let's take a look at some things that would *not* have been possible with an Application.cfm. The first method we added to the Application.cfc file was onApplicationStart. There is a corresponding method named onApplicationEnd. This method will be run when the application times out. Since the file is getting a bit large now, I'll just paste in the method:

Listing 10: onApplicationEnd

```
1   <cffunction name="onApplicationEnd" returnType="void" output="false">
2     <cfargument name="applicationScope" required="true">
3     <cflog file="#this.name#" text="Application timed out.">
4   </cffunction>
```

There are two important things to note here. First, an application "dies" when no one hits the site for a certain period of time (the application timeout). Because no one is actually at the site, you can't use the method to display a message. Who would the message be displayed to? As you can see, all I did in this method was to cflog the event. The second thing to note is that officially, at this point, the *Application* scope no longer exists. But since ColdFusion passes in the *Application* scope as the one and only argument to the method, if you wanted to email the address specified in application.adminEmail, you would need to use arguments.applicationScope.adminEmail instead. By the way, the names for the arguments that you use in these methods are up to you. While ColdFusion has a specific "API" for what it sends to each method, you can use whatever names you desire.

The onSessionEnd Method

Besides knowing when an application ends, the Application.cfc file gives you access to the end of the session event. As before, I'll paste in just this new method:

Listing 11: onSessionEnd

```
1   <cffunction
2     <cfargument name="sessionScope" type="struct" required="true">
3     <cfargument name="appScope" type="struct" required="false">
4     <cfset var duration = dateDiff("n", arguments.sessionScope.started, now())>
5     <cflog file="#this.name#" text="Session timed out after #duration#
        minutes.">
6   </cffunction>
```

Like the onApplicationEnd method, we can't use this method to display a message, but we can use it to log information. Also, like the onApplicationEnd method, we don't actually have access to the *Session* scope (it did end, after all), nor do we have access to the *Application* scope. Both scopes are passed to the method, however. Note that since the *Application* scope is passed as a pointer, if you modify the *Application* scope via the argument, you will be modifying the "real" *Application* scope.

In my simple example above, I used the fact that I stored the session start time to log the duration of the session. This is an excellent use of onSessionEnd, as I can see how long people are sticking around my site.

The onRequestEnd Method

Since we are talking about "end" methods, let's look at one more — onRequestEnd. This is the opposite of onRequestStart. Technically, this type of operation was possible in the past using the onRequestEnd.cfm page. Like Application.cfm, if the onRequestEnd.cfm file existed, it would be automatically included at the end of a request. Here is a simple example of the onRequestEnd method:

Listing 12: onRequestEnd

```
1  <cffunction name="onRequestEnd" returnType="void" output="true">
2    <cfargument name="thePage" type="string" required="true">
3    <cfoutput>
4      <p align="right">
5        Copyright #year(now())#
6      </p>
7    </cfoutput>
8  </cffunction>
```

In this example, I'm using the onRequestEnd method to display a copyright notice that will run on each and every page request. Let me be clear. I do *not* like, nor approve of, doing outputs like this in the Application.cfc file. I typically use custom tags to do my layout outputs. However, this was a simple enough example of how to use the method.

The onRequest Method

The onRequest method is definitely the "problem child" of the Application.cfc family. It can be a bit confusing to use and has some serious side effects that I'll talk about later.

Let's start with a very simple and bare bones example:

Listing 13: onRequest

```
1  <cffunction name="onRequest" returnType="void">
2    <cfargument name="thePage" type="string" required="true">
3    <cfinclude template="#arguments.thePage#">
4  </cffunction>
```

As it does in the onRequestStart method, ColdFusion passes the name of the script being executed to the onRequest method. However —and this is very important to note —you *must* actually include the file. If you don't, it won't be processed. The reason for this is that the onRequest

method overrides the normal behavior of how files are processed. It gives you precise control over the entire request, and therefore, you have to do the work that ColdFusion would normally do — in this case, simply including the template.

To be honest, I haven't seen very many uses for onRequest, and I rarely use this method myself. In the past, I've used it to enable a printable version of a site. I did this by building my page templates to use HTML comment markers. I then used the onRequest method to check for a URL variable. If it existed, I simply used regular expressions to remove the header and footer, which was easy with the markers I had in the layout. Here is a simple example of that:

Listing 14: onRequest with Print Functionality

```
1   <cffunction name="onRequest" returnType="void">
2     <cfargument name="thePage" type="string" required="true">
3     <cfset var buffer = "">
4     <cfset var thisPage = "">
5     <cfif isDefined("url.p")>
6       <cfsavecontent variable="buffer">
7         <cfinclude template="#arguments.thePage#">
8       </cfsavecontent>
9       <cfset buffer = reReplace(buffer,".*?<!-- Page Body -->","")>
10      <cfset buffer = reReplace(buffer,"<!-- Page Body -->.*","")>
11      <cfoutput>#buffer#</cfoutput>
12    <cfelse>
13      <cfinclude template="#arguments.thePage#">
14    </cfif>
15  </cffunction>
```

The code simply notices a URL variable. When the variable exists, I used a regular expression to remove everything before and after a set of HTML comments.

Another possible use is to strip out the extra white space generated by ColdFusion. (One thing ColdFusion likes to do is generate white space.)

Using onRequest has three negative side effects, though. The first "gotcha" is the fact that all of the methods and the This scope for the Application.cfc will be copied to the local Variables scope for your script. While this isn't a huge big deal, it may be surprising if you dump the Variables scope.

The second and third negative side effect is that the mere presence of the onRequest method will immediately break Flash Remoting and Web Service calls to CFCs in the application. This is because there are certain data transformations that take place as part of a Flash Remoting or Web Service request that are not included in the execution of the onRequest method.

This *is* a big deal! There are two ways you can get around it. The first is to place your CFCs in a subdirectory and add another Application.cfc (one without the onRequest method, of course) within the same folder as the CFCs, or you can use the workaround developed by Sean Corfield.

It's a bit weird looking, but works well and I recommend it. The idea is simple. In the onRequestStart method, notice a call for a CFC and hide the onRequest method. We do this by taking advantage of how methods are stored inside a CFC. All methods are stored to a structure called *This*. The "*This*" structure holds all of the information about the CFC and what it contains. The code below checks to see if the page requested is a CFC or if it uses the Flash Services gate-

way URL. It then deletes the onRequest method from the *This* structure. This is done only for the current request. It doesn't remove the method permanently. Here is a code sample that can be used in the onRequestStart method:

```
<cfif listLast(arguments.thePage,".") is "cfc" or arguments.thePage is
   "/flashservices/gateway">
   <cfset structDelete(this,"onRequest")>
</cfif>
```

Application.CFC: A New Application Structure

The Application.cfc feature has truly added a good deal of power to our ColdFusion applications. It pulls together all the pieces of an application into a single, streamlined template that provides us with functionality we previously didn't have. We can finally determine how many sessions are active in an application. We can determine when a session ends and when an application times out. We can determine both the start-time and end-time of any application or session. We have these new powers and the benefit of a much more organized application structure.

While this article has barely scratched the surface of what you can do to take advantage of the new Application.cfc feature, I highly recommend you begin today to convert your Application.cfm files to the new format. If you need a handy reference, you can find a "skeleton" Application.cfc PDF at http://ray.camdenfamily.com/downloads/app.pdf. Personally, I find the Application.cfc files much easier to parse than older Application.cfm files.

Raymond Camden is Vice President of Technology for roundpeg, Inc. A long-time ColdFusion user, Raymond has worked on numerous ColdFusion books and serves as a technical editor and contributor for the *ColdFusion Developer's Journal.* He also presents at numerous conferences and contributes to online webzines. He and Rob Brooks-Bilson created and run the Common Function Library Project (www.cflib.org), an open source repository of ColdFusion UDFs. Raymond has helped form three ColdFusion User Groups and is the manager of the Acadiana MMUG. You can reach him at ray@camdenfamily.com.

Where You Should Go to Be Up-to-Date

Blogs:

Ray Camden's Blog:
http://ray.camdenfamily.com/index.cfm

Sean Corfield's Blog:
http://corfield.org/blog/

Michael Dinowitz's Blog:
http://www.blogoffusion.com/

Ben Forta's Blog:
http://www.forta.com/blog/

Fullasagoog:
http://www.fullasagoog.com/

Macromedia XML News Aggregator:
http://weblogs.macromedia.com/mxna/

PROFESSIONAL COLDFUSION CONTENT FOR THE COLDFUSION PROFESSIONAL

The Fusion Authority Quarterly Update

Summer 2006

Why You Should Use Application.cfc
by the CF Jedi Master, Raymond Camden

The Why, When and How of Flash Forms
Matt Woodward's Cool Tricks

An Honest Look at Integrated Reporting
Kay Smoljak Investigates

Let Asynchronous Processing Save You Time and Money
Doug Boude's Experience

What's Hot? What's Not? What's on the Guru's minds?
by Ben Forta * Michael Dinowitz * Hal Helms
Sean Corfield * Raymond Camden * Jared Rypka-Hauer

and more...

ColdFusion MX 7 Features you need to know

SUMMER 2006
ISSN 1932-0264