Schedule Until...

Posted At : October 9, 2007 6:00 AM | Posted By : Steve
Related Categories: ColdFusion, com.sebtools

This week I have been working on a task that will send out several email messages many more than I want to send to the mailserver at one time. I need some way to send them out in batches until they are all done. My solution led to something that will handle anything that needs to repeat across large time spans until complete.

The last time that I ran into this problem, I added a command email to the end of the batch and had a scheduled task check the batch for that command email to send the next batch. Although I still like this approach, I don't need anything that complicated this time.

I do, however, want something that solves the general problem of repeating tasks with large breaks between iterations.

First I turned to Scheduler.cfc to help me. I discovered that I had interval="once" set up incorrectly (it executed the task immediately). So, I now have a new beta (#2) that correctly runs interval "once" only one time, but doesn't do so immediately.

I put together an example CFC using the technique. In the example, I am creating records at spaced intervals instead of sending email, but the concept is still the same. The example CFC requires DataMgr and Scheduler.cfc to work.

Scheduler.cfc requires a scheduled task running that calls it.

I generally create a file called schedule.cfm for any site that uses Scheduler.cfc. I set up a scheduled task (either through CF Admin, CFSCHEDULE, or perhaps an outside monitering script) to call that file at least once an hour.

schedule.cfm has the following code:

```
<cfset Application.Scheduler.runTasks()>
```

Here is the init method for my example CFC that uses Scheduler.cfc:

```
<cffunction name="init" returntype="any" access="public" output="no">
   <cfargument name="DataMgr" type="any" required="yes">
   <cfargument name="Scheduler" type="any" required="yes">

   <cfset variables.DataMgr = arguments.DataMgr>
   <cfset variables.Scheduler = arguments.Scheduler>

   <cfset variables.datasource = variables.DataMgr.getDatasource()>
   <cfset variables.DataMgr.loadXML(getDbXml(),true,true)><!--- Make sure needed tables/columns exist --->

   <cfreturn this>
</cffunction>
```

This merely creates references to DataMgr and Scheduler and any tables it is using internally.

The do method is the heart of the example:

```
<cffunction name="do" returntype="void" access="public" output="no">

   <cfset var ii = 0>
   <cfset var sData = 0>
   <cfset var qTasks = 0>
```

```
   <!--- Insert 30 random records --->
   <cfloop index="ii" from="1" to="30" step="1">
      <cfset sData = StructNew()>
      <cfset sData["UUID"] = CreateUUID()>
      <cfset variables.DataMgr.insertRecord("exTasks",sData)>
   </cfloop>

   <!--- Get all of the records after the inserts --->
   <cfset qTasks = variables.DataMgr.getRecords(tablename="exTasks",fieldlist="TaskID")>

   <!--- If less than 300 records were inserted, have the scheduler call this method again the next chance it gets --->
   <cfif qTasks.RecordCount LT 300>
      <cfinvoke component="#variables.Scheduler#" method="setTask">
         <cfinvokeargument name="Name" value="DoTasks">
         <cfinvokeargument name="ComponentPath" value=" Tasks">
         <cfinvokeargument name="Component" value="#this#">
         <cfinvokeargument name="MethodName" value="do">
         <cfinvokeargument name="Interval" value="once">
         <cfinvokeargument name="hours" value="#Hour(DateAdd('h',1,now()))#,#Hour(DateAdd('h',2,now()))#">
      </cfinvoke>
   </cfif>

</cffunction>
```

This function will add 30 records at a time until it has 300 records. The first part of the function is simply to add 30 records and the get the records from the table. After that, the function will check to see if it has 300 records.

If it has fewer than 300 records it will make a call to Scheduler to have Scheduler execute the method again (at which point it will perform the same task). This is an indirect recursive call, so care must be taken to make sure that it quits running eventually.

Here is an examination of the arguments of Scheduler.setTask:

- Name: Just a unique name for this task
- ComponentPath: A path to this component, just used to uniquely identify the component
- Component: #this#
- MethodName: In this case, the name of the calling method make the call indirectly recursive
- Interval: "Once" ensures that Scheduler will only call this method once, at which point the method may create the task again.
- hours: A list of hours. I ensure that the next call is at least an hour away so that the method doesn't get called again immediately.
- args: Although I didn't need it for this example, I could have passed #arguments# in here to call the method with the same arguments it was originally give.

The hours argument lists the next two hours. Since Interval is "once", it will only get called the once regardless. Listing two different hours ensures it will get called even if there is some oddity in the timing.

What I effectively have now is a do...until loop that runs over several hours without tying up system resources the whole time.

Scheduler.cfc only allows one task of the same name to be operating at the same time. So, if you want to have the same method generating several do...until loops, you would need to name each task a unique name (you could toss "#CreateUUID()#" in the name and that would take care of it).

Scheduler.cfc is open source and free for any user. It does require **DataMgr** in order to work.