Using Stored Procedures? Take SPCaller for a Spin

Posted At : July 9, 2008 11:32 AM | Posted By : Bob Silverberg
Related Categories: SPCaller, ColdFusion

The code for the SPCaller component that I discussed in a previous posting is now available for download at RIAForge.

I have added some tickets to the Issue Tracker for possible enhancements. If anyone downloads the code and is interested in these, or any other enhancements, please contact me. I'd also be very interested to hear from anyone that gives it a try and finds it useful.

What follows are some implementation and usage notes. They will probably be less than interesting to anyone who isn't planning on trying out the component ;-)

Implementation

It is recommended, but not required, that this component be instantiated as a singleton. You can implement it in your model via composition, inheritance or simply as a standalone object in the application scope. I commonly compose it into my other objects, using Coldspring, so here's an example of that first:

```
<bean id="SPCaller" class="path_to_cfc.SPCaller">
 <constructor-arg name="DSN">
  <value>MyDatasourceName</value>
 </constructor-arg>
</bean>
<bean id="MyGateway" class="path_to_cfc.MyGateway">
 <property name="SPCaller">
  <ref bean="SPCaller" />
 </property>
</bean>
```

You'd then need to add a setSPCaller() method in your MyGateway.cfc, for example:

```
<cffunction name="setSPCaller" access="public" returntype="void" output="false" hint="I set the SPCaller.">
 <cfargument name="SPCaller" type="any" required="true" />
 <cfset variables.instance.SPCaller = arguments.SPCaller />
</cffunction>
```

Then, to call it you'd write something like this:

```
<cfset qryTest = variables.instance.SPCaller.callSP("mySP") />
```

If you prefer inheritance, you could also simply extend it with an object. In that case you wouldn't use Coldspring, you'd just define your component like so:

```
<cfcomponent displayname="MyGateway" output="false" extends="path_to_cfc.SPCaller">

<cffunction name="Init" access="Public" returntype="any" output="false" hint="I build a new MyGateway">
 <cfargument name="DSN" type="string" required="true" hint="The name of the default datasource" />
 <cfset super.Init(arguments.DSN) />
 <cfreturn this />
</cffunction>

...
</cfcomponent>
```

You'd have to make sure that the Init() method of your object (in this example, MyGateway) also extended the Init() method of SPCaller. If you go this route, you could call it like this:

```
<cfset qryTest = callSP("mySP") />
```

When I first started using this component, I did use it in this manner, using it as a base component for all of my Business Objects, so that each of my Business Objects had a callSP() method.

Finally, you can also instantiate the component manually, like this:

```
<cfset application.SPCaller = CreateObject("component","path_to_cfc.SPCaller").Init("MyDatasourceName") />
```

In which case you'd call it like this:

```
<cfset qryTest = application.callSP("mySP") />
```

Arguments

The SPCaller component has one method that you would call, callSP(), which accepts the following arguments

1. SPName - The name of your stored procedure.
2. DataStruct - An optional argument which is a structure of data that should be passed into the SP's parameters. This is optional as often an SP will not have any parameters.
3. DSN - The datasource to be used when calling the SP. This is also optional, as it is only required if you wish to override the DSN that was set via the Init() method.

Usage

For example, to call this SP:

```
CREATE PROCEDURE [dbo].[Test_Update]

 @id int

 ,@colVarChar varchar(50)

AS

SET NOCOUNT ON;



UPDATE tblDataTypes

SET  colVarChar = @colVarChar

WHERE id = @id



SELECT  id, colVarChar

FROM tblDataTypes

WHERE id = @id
```

You could do:

```
<cfset DataStruct = StructNew() />

<cfset DataStruct.id = 1 />

<cfset DataStruct.colVarChar = "New Text" />

<cfset qryTest = SPCaller.callSP("Test_Update",DataStruct) />
```

If you already have all of your data in a struct, for example in the attributes scope in Fusebox, or from Event.getAllValues() in Model-Glue, then you can simply pass that struct into the DataStruct argument, which saves a lot of work.