

founder & ceo

Fuat Kircaali fuat@sys-con.com

group publisher

Jeremy Geelan jeremy@sys-con.com

advertising

senior vp, sales & marketing

Carmen Gonzalez carmen@sys-con.com

vp, sales & marketing

Miles Silverman miles@sys-con.com

advertising sales director

Megan Mussa megan@sys-con.com

associate sales manager

Corinna Melcon corinna@sys-con.com

sys-con events

president, events

Carmen Gonzalez carmen@sys-con.com

events manager

Lauren Orsi lauren@sys-con.com

customer relations

circulation service coordinator

Edna Earle Russell edna@sys-con.com

sys-con.com

vp, information systems

Robert Diamond robert@sys-con.com

web designers

Stephen Kilmurray stephen@sys-con.com

Richard Walter richard@sys-con.com

accounting

financial analyst

Joan LaRose joan@sys-con.com

accounts payable

Betty White betty@sys-con.com

subscriptions

Subscribe@sys-con.com

Call 1-888-303-5282

For subscriptions and requests for bulk orders, please send your letters to Subscription Department

Cover Price: \$8.99/issue

Domestic: \$89.99/yr (12 issues)

Canada/Mexico: \$99.99/yr

All other countries \$129.99/yr

(U.S. Banks or Money Orders)

Back issues: \$12 U.S. \$15 all others

Data Table Gateways

Working on a collection of records



By Jeffrey Houser

In my previous article I wrote about Data

Access Objects. Data

Access Objects, or

DAOs for short, are a

way to separate your insert, select, delete, and update queries from other business logic. This lets you switch from one data storage mechanism to another easily. Whenever people talk about DAOs they also talk about Data Gateways.

I've also heard them called Table Gateways or more commonly gateway objects. The two often go together and are similar in concept. Data Access Objects are designed to work on a single record, whereas Table Gateways are designed to work on a collection of records.

MyFriends' RSSCategories

In the article on DAOs, I took a component from my RSS Aggregator project, MyFriends, and changed its implementation to use a Data Access object. You can download the aggregator code from the software pod on my blog at www.jeffryhouser.com. For this article, I thought I'd take the same data, RSSCategories, and create a gateway component.

When you enter RSS feeds into the system, they can be categorized in any way you like, and the category information is stored in the RSSCategory table. The table has two columns, a primary key, CategoryID, and a

category name column called reasonably enough Category. When creating the DAO, I took an existing component and modified it to use the DAO pattern. Currently, the system doesn't have a gateway component yet, so in this case we'll start from scratch.

Generic Properties & Methods

When creating a Gateway object there are often generic properties and methods that I use. You can encapsulate these into a GenericGateway component. All future gateways will inherit from the gateway.

Here are the generic properties:

- **DSN:** When you're accessing a database from ColdFusion, you need to know the name of the datasource. This property holds that.
- **ColumnList:** The columnlist property will contain the name of the database columns that you want to retrieve from the database. I usually default this to '*'.
- **MaxRows:** How many rows do you want to return? In most cases, you want to return all of the rows, so I default the maxrows property to -1.
- **OrderList:** The orderlist property contains a list of all the fields that you're going to order the query results by. The order list is going to be dependent on the columns in the query. I default this to a blank string.
- **Criteria:** This is usually a set of properties that you use to define the selection criteria from the query. Perhaps you only want users whose names begin with the letter A. Maybe you only want products whose price is less than \$5. I don't implement generic methods in the GenericGateway for these, since they're often specific to the query you want to run.

Here are the generic methods:

- **Getters and Setters:** The getters and setters are inherited from the Base Com-



ponent. I use Hal Helm's generic getter and setter methods, available from his Web site at <http://halhelms.com/webresources/BaseComponent.cfc>.

- **Init:** The init method is one that will define the generic properties of the component such as the DSN, ColumnList, MaxRows, and OrderList.
- **Execute:** The execute method is one that will piece together the query from the various property information, run it, and return the query.
- **Criteria methods:** In some cases, the criteria can be more complex than you would set with a simple getter or setter. Perhaps you want to use a range of numbers in your query. Maybe you want to test for equality and similarity using wildcards. Sometimes these are implemented better with their own criteria setting method from inside the sub-component. In most cases, I don't have any additional criteria setting methods. You can use your judgment.

Next I'll show the implementation of the GenericGateway object and then the implementation of an RSSCategories gateway.

The Generic Gateway Object

You can take a look at the code in the Generic Gateway object in Listing 1. It starts out with the cfcomponent tag. Act surprised. The component extends the BaseComponent. The pseudo-constructor code initializes four generic properties: dsn, columnlist, maxrows, and orderlist. There's a single method named init. The init method accepts the four separate arguments, one for each property. If that property is defined, it overrides the default. This is a pretty generic component. I leave the implementation of the execute method for the specific components that inherit from the generic gateway.

Writing the RSSCategoryGateway Object

The RSSCategoryGateway.cfc can be seen in Listing 2. The component extends the GenericGateway, thus inheriting all its methods and properties. It adds two instance variables to the mix, CategoryID and Category. In this case, I'm not changing any of the default values, or I would override them as part of the pseudo-constructor.

There's one new method in this gateway, the execute method. Of course, this component inherits all the methods from the genericGateway, and its parent is the BaseComponent. The execute method, you'll remember, will piece together the query, run it, and return the results. That's exactly what this one does.

The method vars the query name so it stays local to the method. Then it has a cfquery tag. The query selects the columnlist from the table. Since this isn't intended to be generic, I didn't use the tablename as a variable. Then I enter the where conditions. I don't know if there will be any conditions

or not, so I use an SQL trick, where 0=0; 0 is always equal to 0, so this condition will always be true no matter what data is returned from the query. Using this as the first condition the query lets me use, which remains true of all future conditions, since there will always be a prior condition. The code checks to see if the CategoryID is zero. If it is, do nothing. If it's not, filter the output based on the CategoryID value. I set up CategoryID to test equality, although it could easily do a greater than or less than, or something else completely depending on the data you're trying to retrieve. Next it checks the Category instance variable. If it's an empty string, do nothing. Otherwise, add in the Category check clause. I added a wildcard to the category field.

If there are no criteria, a finished query may be as simple as:

```
Select *
From RSSCategories
Where 0=0
```

If both criteria are used together, the query may turn out something like:

```
Select *
From RSSCategories
Where 0=0
And RSSCategories.CategoryID = 1
And RSSCategories.Category like 'A%'
```

This may seem like a lot of overhead when using a table with just a two fields. But, with larger tables, or more complicated queries, the benefits can be seen more easily. You might use this component when creating a system for editing the categories, but another gateway when creating reports based on categories and RSSFeeds in those categories.

Using the Gateway

I can show you a simple example of how we can use the component RSSCategoryGateway component. First we need to create an instance and run the init method:

```
variables.RSSCategories = CreateObject("component", "#request.Component-
Loc#.RSSCategoryGateway");
variables.RSSCategories.init('*', 'category', -1, request.DSN);
```

This was put inside a CFScript block. The init method just resets the defaults in this case with the exception of the DSN. A blank DSN won't do us any good. This piece of code will run the simple query, with no filters:

```
ResultsNoFilter = variables.RSSCategories.
execute();
```

You can dump ResultsNoFilter to see all



entries in the RSSCategories table. Let's add a filter:

```
variables.RSSCategories.set('category', 'A');
ResultsCategoryFilter = variables.RSSCategories.execute();
```


You can easily dump the results to see all the categories that start with the letter A. This is a simple concept that has a lot of power especially when dealing with complicated queries.

Final Thoughts

As with DAO objects, I feel that Gateways implementations in ColdFusion are severely lacking in documentation. Everyone talks about why to use them; no one talks about how to implement them. I hope this article helped give you a head start on using gateways. It's easy for me to think of variations of this implementation that can achieve the same level of encapsulation and reuse, and still meet the definition of a gateway.

I'm now entering my third year of writing this column. Some-

times it's hard to figure out what to write about in a beginner's column that hasn't been done ad nauseam. I'd love to get some feedback from readers on what they want me to discuss in the coming year. To contact me, just go to my blog at www.jeffryhouser.com and fill out the contact form.

And one last plug, for those who are dying to hear the sound of my voice, I'm the co-host of a Flex-related podcast at www.theflexshow.com. Give a listen if you're interested in Flex! 

About the Author

Jeffry Houser has been working with computers for over 20 years and in Web development for over 8 years. He owns a consulting company and has authored three separate books on ColdFusion, most recently ColdFusion MX: The Complete Reference (McGraw-Hill Osborne Media).

jeff@instantcoldfusion.com

Listing 1

```
<cfcomponent extends="BaseComponent">

<cfscript>
variables.instance.dsn = "";
variables.instance.columnlist = "**";
variables.instance.maxrows = -1
variables.instance.orderlist = "";
</cfscript>

<cffunction name="Init" access="Public" returntype="void" hint="I init
the component">
<cfargument name="columnlist" type="string" required="false">
<cfargument name="orderlist" type="string" required="false">
<cfargument name="maxrows" type="numeric" required="false">
<cfargument name="dsn" type="string" required="false">

<cfif IsDefined("arguments.columnlist")>
<cfset variables.instance.columnlist = arguments.columnlist>
</cfif>
<cfif IsDefined("arguments.orderlist")>
<cfset variables.instance.orderlist = arguments.orderlist>
</cfif>
<cfif IsDefined("arguments.maxrows")>
<cfset variables.instance.maxrows = arguments.maxrows>
</cfif>
<cfif IsDefined("arguments.dsn")>
<cfset variables.instance.dsn = arguments.dsn>
</cfif>

</cffunction>

</cfcomponent>
```

Listing 2

```
<cfcomponent extends="GenericGateway">
```

```
<cfscript>
variables.instance.CategoryID = 0;
variables.instance.Category = "";
</cfscript>

<cffunction name="execute" access="public" returntype="query">

<cfset var getCategories = "">
<cfquery name="getCategories" datasource="#variables.instance.dsn#">
select #variables.instance.columnlist#
from RSSCategories
where 0=0
<cfif variables.instance.CategoryID NEQ 0>and RSSCategories.CategoryID
= #variables.instance.CategoryID# </cfif>
<cfif variables.instance.Category NEQ "">and RSSCategories.Category
like '#variables.instance.Category#%' </cfif>
</cfquery>

<cfreturn getCategories>

</cffunction>

</cfcomponent>
```

Download the Code...
Go to <http://coldfusion.sys-con.com>