DataMgr Components

Posted At : August 3, 2005 12:38 PM | Posted By : Steve
Related Categories: DataMgr

The Challenges

Whenever I do a simple insert or update, I do several of the same simple checks.
Beyond that, the error messaged I get from the database often aren't very informative. Specifically, when the data is too long for a string field, the database generally doesn't indicate which field.

I also have a handful of reusable components and I want them to create any tables they need on any database I might use.

The Solution

Let's look at a sample "CritterMgr" component for handling, say, critters. This will be stored in CritterMgr.cfc in the root of our web site.

```
<cfcomponent><cffunction name="init" access="public" returntype="any" output="no">
    <cfargument name="DataMgr" type="com.sebtools.DataMgr" required="yes">

 <cfset variables.DataMgr = arguments.DataMgr>
 <cfset variables.DataMgr.loadXML(getDbXml(),true)> <cfreturn this>
</cffunction><cffunction name="setCritter" access="public" returntype="void" output="no">
    <cfargument name="CritterName" type="string">
    <cfargument name="BirthDate" type="date">
    <cfargument name="CritterID" type="numeric" required="No"> <cfif StructKeyExists(arguments,"CritterID")>
        <cfset variables.DataMgr.updateRecord('Critters',arguments)>
    <cfelse>
        <cfset variables.DataMgr.insertRecord('Critters',arguments)>
    </cfif>
</cffunction><cffunction name="getDbXml" access="private" returntype="string" output="no">
<cfset var tableXML = "">
<cfsavecontent variable="tableXML">
<tables>
    <table name="Critters">
        <field ColumnName="CritterID" CF_DataType="CF_SQL_INTEGER" PrimaryKey="true" Increment="true" />
        <field ColumnName="CritterName" CF_DataType="CF_SQL_VARCHAR" Length="60" />
        <field ColumnName="BirthDate" CF_DataType="CF_SQL_Date" />
    </table>
</tables>
</cfsavecontent>
<cfreturn tableXML>
</cffunction>
</cfcomponent>
```

We will initialize CritterMgr with the init() method. Since that takes the DataMgr as an argument, we better initialize that first (Application scope is personal preference only):

```
<cfset Application.DataMgr = CreateObject("component","com.sebtools.DataMgr_MSSQL").init("mydsn")>
```

CritterMgr expects a type of "com.sebtools.DataMgr", but we are instantiating com.sebtools.DataMgr_MSSQL. This works because DataMgr_MSSQL extends DataMgr. Note that DataMgr components must be in com.sebtools. This can be in a com/sebtools under your CustomTags directory or under the root of your web site.

Now we can call the CritterMgr:

```
<cfset Application.Critter = CreateObject("component","CritterMgr").init(Application.DataMgr)>
```

CritterMgr's init() method takes the DataMgr and makes it available to the rest of the component in variables scope.

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

DataMgr must know the structure of any table that it works with. If the tables already exist, you have the option to simply pass the table name to the loadTable() method (note that DataMgr_Access does not yet support this).

Our CritterMgr, however, needs to define its own tables, so we use the loadXML() method of DataMgr. CritterMgr passes the XML to DataMgr's loadXML() and indicates true for table creation. In this example, the XML is retrieved from a private getDbXml() method.

```
<cfset variables.DataMgr.loadXML(getDbXml(),true)>
```

This should be a good example of the XML format. The **XML Schema** is also available. Note that the XML is case-sensitive.

In this example, we are creating tables in MS SQL Server (by using DataMgr_MSSQL). If we wanted to install the tables in a MySQL database we would have just called DataMgr using DataMgr_MySQL. DataMgr components exit for Access, MS SQL and MySQL (so far).

Next we need to add and update a records. Let's assume that we have a form with a BirthDate field and a CritterName field. The form will also contain a "CritterID" field if it is for an update. That being the case, here would be our code for an insert or update on the action page.

```
<cfset Application.Critter.setCritter(argumentCollection=form)>
```

Or

```
<cfinvoke component="#Application.Critter#" method="setCritter" argumentcollection="#form#"></cfinvoke>
```

Since the form scope is a structure, we can pass it into the setCritter method using argumentcollection. In this case, the form names match the argument names. Otherwise, we would have to specify each field instead of using argumentcollection.

The setCritter() method then calls insertRecord() or updateRecord() appropriately. It passes in the name of the table on which to perform the action and the structure of data to insert/update.

```
<cfif StructKeyExists(arguments,"CritterID")>
    <cfset variables.DataMgr.updateRecord('Critters',arguments)>
<cfelse>
    <cfset variables.DataMgr.insertRecord('Critters',arguments)>
</cfif>
```

For inserts and updates, DataMgr needs a structure indicating the data to insert or update with a key for each field with the value that you wish to insert/update. Since the arguments scope is a structure, we can pass it directly into our insert or update method. If the argument names didn't match the field names in the database, we would just need to create the appropriate structure in the method itself.

DataMgr will test each value for validity with the data-type of the field. It will throw ane error for any character field receiving more data than it can hold. Note that updates must include the primary key field or the DataMgr will throw an error.

Any fields missing from the insert/update method will not be included in the resulting query. So, if the BirthDate field isn't included in an update then it would keep the value it had before.

Conclusion

Now, inserts and updates are handled easily and with extra error-handling. Our CritterMgr can easily be installed on any database which DataMgr supports.

For more information, check out **my CFCs page**, which has documentation on DataMgr and downloadable files.

Leave me a note and let me know what you think!