MySQL in ColdFusion Example Documentation

This article shows how to use EditLive! as the interface for a database driven press release center. This sample allows users to:

- add new press articles,
- edit existing press articles,
- view the article in the web browser, and
- delete existing press articles.

The complete source code for this example can be found in the *EDITLIVE_INSTALL/webfolder/examples/coldfusion/database* folder, where *EDITLIVE_INSTALL* is the location that the EditLive! for Java SDK has been installed to.

Getting Started

Required Skills

The following skills are required prior to working with this example:

- Basic client-side JavaScript
- Basic ColdFusion
- Basic MySQL

Overview

In this sample EditLive! is embedded into a Web page using ColdFusion and JavaScript. The example sets several variables affecting the appearance and functionality of the applet and loads a basic document into the instance of the applet.

This example demonstrates how to perform the following with EditLive! and ColdFusion:

- Embed an instance of EditLive! in a Web page using JavaScript.
- Invoke methods and set parameters effecting the appearance of EditLive!.
- Load a document into EditLive! from a ColdFusion variable.

Creating the Database

The database used in this sample is a MySQL database named ELContent. Below is a table outlining the fields within the database table articles and a description of the information each field stores.

Column Name	Description
article_id	A unique number used to identify the article record
article_title	The title or headline of the article
article_body	The actual article content
article_styleElementText	The styles used to format the article if content was pasted in to EditLive! for Microsoft Word

To create this table, use the following MySQL query:

```
CREATE TABLE `articles` (
   `article_id` int(11) NOT NULL auto_increment,
   `article_title` varchar(255) collate utf8_unicode_ci NOT NULL default '',
   `article_styleElementText` blob NOT NULL,
   `article_body` blob NOT NULL,
   PRIMARY KEY (`article_id`)
) DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

To create the entire database including sample content, import the database.sql file. Use the following command, which will prompt you for the mysql root password:

```
mysql -u root -p < database.sql
```

Creating the ColdFusion Data Source

This example uses a ColdFusion data source called ELContent. To add a data source:

1. Start the ColdFusion Administrator. For ColdFusion MX 7 on Windows, Select **Start -> Programs -> Macromedia -> ColdFusion MX 7 -> Administrator**.
2. Enter the **Administrator** password if prompted.
3. Choose **Datasources** from the **Data & Services** section on the left.
4. Enter the Data Source Name **ELContent**.
5. Choose the Driver **MySQL**.
6. Click **Add**.
7. Enter the database name **ELContent**.
8. Enter the server, username and password for your MySQL server.
9. Click **Submit**.
10. Locate ELContent in the list and click **Verify**. This will run a test to ensure the data source connection is correct. You will see a message stating that the connection to the data source was verified correctly.

Integrating EditLive!

To use EditLive! with a database, several web pages are required. Each of these pages is explained here and code samples are provided.

Index Page (start.cfm)

The index page of the press release center lists all of the articles currently available in the database. Users are able to:

- create a new article using EditLive!,
- edit an existing article using EditLive!, and
- delete an existing article from the press database.

1. Create a link in the page to the file *add.cfm* to allow users to use EditLive! to create a new article.

```
<P><A href="add.cfm">Create a new article</A></P>
```

2. Connect to the database and retrieve all of the existing articles listed in the articles table.

```
<!---Create a query to retrieve a list of articles from the database--->
<CFQUERY DATASOURCE="ELContent" NAME="article_list">
   SELECT * FROM articles ORDER BY article_id
</CFQUERY>

<!---Check if there are records in the database.--->
<CFIF article_list.RecordCount IS "0">
```

3. Let users know if there are no records in the database.

```
<!---There are no records. Tell the user.--->
   <p>There are no records in the database. Click <STRONG>Add an Article</STRONG> to add a record to the
      database.</p>
```

4. Use embedded ColdFusion to loop through the recordset of articles to create a table which lists all of the existing articles by title. Then, for each article, create three links:

- View page to view the article
- Edit page to edit the article using EditLive!
- Delete page to delete the article from the database

Append the article_id to all of the links so that the relevant article is known.

```
<CFELSE>
   <!--- There are records in the database. Loop through all the records in the query and
         write out a table row for each record. Include links to view the article, edit the
         article, or delete the article.
   --->

   <TABLE cellpadding=3 cellspacing=0 border=0>
      <TR>
         <TH>ID</TH>
         <TH width="250">Title</TH>
         <TH>Available Actions</TH>
      </TR>

      <CFOUTPUT query="article_list">
            <TR>
            <TD>#article_id#</TD>
            <TD>#article_title#</TD>
            <TD><A href="view.cfm?article_id=#article_id#">View</A> |
                <A href="edit.cfm?article_id=#article_id#">Edit</A> |
                <A href="delete.cfm?article_id=#article_id#">Delete</A>
            </TD>
         </TR>
      </CFOUTPUT>
   </TABLE>
</CFIF>
```

Adding a New Article (add.cfm and xt_add.cfm)

The file *add.cfm* is used to create a new article using EditLive!. Once the article is complete, it is then added to the database using the processing page, *xt_add.cfm*.

add.cfm
1. Start with a standard HTML page containing EditLive!. To see the steps involved in creating this page please see EditLive! Basic ColdFusion Integration.

2. Add a heading to the page of "Create a New Article".

```
<BODY> <H1>Create a New Article</H1>
```

3. Create a form to place an instance of EditLive! in. Enter the processing file name, *xt_add.cfm*, in the action attribute of the FORM tag.

```
<!--- This form contains EditLive!, a text area for the article title, a submit button and a cancel button.
--->
<FORM action="xt_add.cfm" method="POST" name="articleForm">
```

4. Add text field at the beginning of the form for the article title.

```
<!--- Article title --->
<P>Title: <INPUT type="text" name="article_title" size="50"></P>
```

5. Create a string with the initial content for the new page.

```
<cfset contents="<p>This is the initial source</p>">
```

6. Modify the setBody() call to use the new content variable.

```
ELJApplet1_js.setBody("#URLEncodedFormat(contents)#");
```

⚠ URLEncodedFormat must be used to output the variable; for more information, see the article on Encoding Content for Use with EditLive!.

7. Add two buttons to the end of the form: a submit button to save the article to the database; and a cancel button to return the browser to the index page without saving.

```
<P>
   <INPUT type="submit" value="Save" name="Add Article">
   <INPUT type="button" value="Cancel" name="Cancel" onclick="javascript:history.back();">
</P>
```

8. Close the form tag.

```
</form>
```

xt_add.cfm
1. Generate the SQL query to insert the new content into the database, ensuring all content is escaped for SQL.

```
<!---
Update the article record values in the database with the values from the form objects. Use <cfqueryparam> to
escape the content for SQL.
--->
<CFQUERY datasource="ELContent" name="add_article">
   INSERT INTO articles (article_title, article_body, article_styleElementText)
   VALUES(<cfqueryparam value="#FORM.article_title#" cfsqltype="cf_sql_varchar">,
          <cfqueryparam value="#FORM.ELJApplet1#" cfsqltype="cf_sql_varchar">,
          <cfqueryparam value="#FORM.ELJApplet1_styles#" cfsqltype="cf_sql_varchar">)
</CFQUERY>
```

2. Redirect to start.cfm.

```
<!--- Go back to the start page ---> <CFLOCATION URL="start.cfm">
```

Editing an Existing Article (edit.cfm and xt_edit.cfm)

The file *edit.cfm* loads an existing article into EditLive! for modification. Once the article is complete, the relevant record in the database is updated using the processing page, *xt_edit.cfm*.

edit.cfm
1. Start with a standard HTML page containing EditLive!. To see the steps involved in creating this page please see EditLive! Basic ColdFusion Example.

2. Use <cfparam> to require that the **article_id** parameter has been specified.

```
<!--- throw an error if article_id was not specified --->
<cfparam name="article_id">
```

3. Add a heading to the page of "Edit Document".

```
<BODY> <H1>Edit Document</H1>
```

4. Retrieve the current article. Ensure the articleID variable is escaped for SQL.

```
<!---
Retrieve the specified article details from the database Use <cfqueryparam> to escape the content for SQL.
--->
<CFQUERY DATASOURCE="ELContent" NAME="article_content">
   SELECT * FROM articles WHERE article_id=<cfqueryparam value="#article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>
```

5. Create a form to place an instance of EditLive! in. Enter the processing file name, xt_edit.cfm, in the action attribute of the FORM tag.

```
<!---
This form contains EditLive!, a text area for the article title, a submit button and a cancel button.
--->
<FORM action="xt_add.cfm" method="POST" name="articleForm">
```

6. Create a hidden form object for the article ID.

```
<cfoutput>
   <!--- Hidden field for identifying the article --->
   <INPUT type="hidden" name="article_id" value="#article_content.article_id#">
```

7. Add a text field at the beginning of the form for the article title.

```
<!--- Article title --->
<P>Title:
   <INPUT type="text" name="article_title" value="#HTMLEditFormat(article_content.article_title)#" size="40">
</P>
```

8. Modify the setBody() call to load the content from the database.

```
ELJApplet1_js.setBody("#URLEncodedFormat(article_content.article_body)#");
```

⚠ URLEncodedFormat must be used to output the variable; for more information see the article on Encoding Content for Use with EditLive!.

9. Add a setStyles() call directly after setBody() to load the styles from the database.

```
ELJApplet1_js .setStyles("#URLEncodedFormat(article_content.article_styleElementText)#");
```

⚠ URLEncodedFormat must be used to output the variable; for more information see the article on Encoding Content for Use with EditLive!.

10. Add two buttons to the end of the form: a submit button to save the article to the database; and a cancel button to return the browser to the index page without saving.

```
<P>
   <INPUT type="submit" value="Save" name="Add Article">
   <INPUT type="button" value="Cancel" name="Cancel" onclick="javascript:history.back();">
</P>
```

11. Close the form tag.

```
</form>
```

xt_edit.cfm
1. Generate the SQL query to update the article in the database, ensuring all content is escaped for SQL.

```
<!---
Update the article record values in the database with the values from the form objects. Use <cfqueryparam> to
escape the content for SQL.
--->

<CFQUERY DATASOURCE="ELContent">
UPDATE articles
SET article_title=<cfqueryparam value="#FORM.article_title#" cfsqltype="cf_sql_varchar">,
    article_body=<cfqueryparam value="#FORM.article_body#" cfsqltype="cf_sql_varchar">,
    article_styleElementText=<cfqueryparam value="#FORM.article_styleElementText#" cfsqltype="cf_sql_varchar">
WHERE article_id=<cfqueryparam value="#FORM.article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>
```

2. Redirect to start.cfm.

```
<!--- Go back to the start page --->
<CFLOCATION URL="start.cfm">
```

Deleting an Article (delete.cfm and xt_delete.cfm)

The file delete.cfm displays the chosen article information to allow the user to confirm that they wish to delete the specified article. Once confirmation is obtained, the relevant record in the database is deleted using the processing page, *xt_delete.cfm*.

delete.cfm
1. Use <cfparam> to require that the article_id parameter has been specified.

```
<!--- throw an error if article_id was not specified --->
<cfparam name="article_id">
```

2. Retrieve the current article information. Ensure the articleID variable is escaped for SQL.

```
<!--- Retrieve the specified article details from the database --->
<CFQUERY DATASOURCE="ELContent" NAME="article_content">
SELECT *
FROM articles WHERE article_id=<cfqueryparam value="#article_id#"
     cfsqltype="cf_sql_integer">
</CFQUERY>
```

3. Create a standard HTML page, adding the heading "Delete a Document" and a message asking the user if they wish to delete the article.

```
<HTML>
   <HEAD>
      <CFOUTPUT>
         <TITLE>Delete Article ID #article_content.article_id#?</TITLE>
      </CFOUTPUT>
   </HEAD>
   <BODY>
      <H1>Delete a Document</H1>
      <P>Are you sure you wish to delete this article?</P>
```

4. Create a table in the HTML page that displays the article id and title using the variable defined in Step 2.

```
<!---
This table contains the article ID and title for deletion confirmation.
--->

<TABLE>
<CFOUTPUT>
  <!--- Article ID --->
  <TR>
    <TD>Article ID:</TD>
    <TD>#article_content.article_id#</TD>
  </TR>

  <!--- Article title --->
  <TR>
    <TD>Title:</TD>
    <TD>#article_content.article_title#</TD>
  </TR>
</TABLE>
```

5. Create a form in the page. Put the *xt_delete.cfm* file name in the action attribute of the FORM tag.

```
<!--- Form for deleting the article --->
<FORM action="xt_delete.cfm" method="POST">
```

6. Add a hidden form object to the form to store the article id for processing.

```
<INPUT type="hidden" name="article_id" value="#article_content.article_id#">
```

7. Add buttons to delete the article or cancel the deletion.

```
<P>
   <INPUT type="submit" value="Delete">
   <INPUT type="button" value="Cancel" onclick="javascript:history.back();">
</P>
```

8. Close the form and document tags.

```
        </FORM>
      </CFOUTPUT>
   </BODY>
</HTML>
```

xt_delete.cfm
1. Generate the SQL query to remove the article from the database, ensuring the article id is escaped for SQL.

```
<!--- Delete the specified article from the database --->
<CFQUERY DATASOURCE="ELContent" name="article_delete">
DELETE FROM articles
WHERE article_id=<cfqueryparam value="#FORM.article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>
```

2. Redirect to start.cfm

```
<!--- Go back to the start page --->
<CFLOCATION URL="start.cfm">
```

Viewing an Article (view.cfm)

The file view.cfm displays the chosen article.

1. Use <cfparam> to require that the **article_id** parameter has been specified.

```
<!--- throw an error if article_id was not specified --->
<cfparam name="article_id">
```

2. Retrieve the current article. Ensure the articleID variable is escaped for SQL.

```
<!--- Retrieve the specified article details from the database --->
<CFQUERY DATASOURCE="ELContent" NAME="article_content">
SELECT *
FROM articles
WHERE article_id=<cfqueryparam value="#article_id#" cfsqltype="cf_sql_integer">
</CFQUERY>
```

3. Use embeddded ColdFusion to set the <STYLE> tag of the page to the article style information retrieved from the database and to display the article title and content.

```
<HTML>
   <HEAD>
      <!--- Display the article content --->
      <CFOUTPUT>
         <TITLE>#article_content.article_title#</TITLE>
         <STYLE> #article_content.article_styleElementText#</STYLE>
   </HEAD>
   <BODY>
      <H1>#article_content.article_title#</H1>
      #article_content.article_body#
      </CFOUTPUT>
   </BODY>
</HTML>
```

Handling Image Uploads

To enable the image upload functionality, a new script must be created and then referenced in the configuration file.

To create and reference an upload script, refer to the ColdFusion HTTP File Upload Handler Script section in the EditLive! for Java SDK.