MySQL in PHP Example Documentation

Introduction

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/php/database* folder where *EDITLIVE_INSTALL* is the location where the EditLive! SDK has been installed.

Getting Started

Required Skills

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

- Basic client-side JavaScript
- Basic PHP
- Basic MySQL

Overview

In this sample, EditLive! is embedded into a Web page using PHP 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 PHP:

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

Integrating EditLive! for Java

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

Database Conntection Page (i_database.php)

This file defines the functions used to interact with the database in order to ensure that the other files are not database-specific. The main functions used are DBConnect(), DBQuery(), DBFetchRow() and DBResult(). It also contains the function escape_string() to escape content strings for use in SQL.

Index Page (start.php)

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. Include i_database.php to connect to the database.

```
<?
   require_once("./i_database.php");
?>
```

2. Create a link in the page to the file add.php to allow users to use EditLive! to create a new article.

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

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

```
<?
   //Attempt to establish a connection with the database
   if (!DBConnect()) {
      print DBError();
   }
   else {
      DBQuery("select * from articles");
      if (DBFetchRow()) {
?>
```

4. Use embedded PHP 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.

```
<TABLE cellpadding=3 cellspacing=0 border=0>
   <TR>
      <TH>ID</TH>
      <TH width="250">Title</TH>
      <TH>Available Actions</TH>
   </TR>
<?
   // There are records. Loop through all the records in the
   // recordset and write out a table row for each record
   do {
?>
   <TR>
      <TD><?=DBResult("article_id")?></TD>
      <TD><?=DBResult("article_title")?></TD>
      <TD><A href="view.php?article_id=<?=DBResult("article_id")?>">View</A> |
          <A href="edit.php?article_id=<?=DBResult("article_id")?>">Edit</A> |
          <A href="delete.php?article_id=<?=DBResult("article_id")?>">Delete </A>
      </TD>
   </TR>
<?
   } while(DBFetchRow())
?>
</TABLE>
```

5. Add an ELSE statement to the IF loop to let users know if there are no records in the database.

```
<?
   } else {
?>
<P>There are no records in the database. Click <STRONG>Create a new article</STRONG> to add a record to the
database.</P>
<?
   } }
?>
```

6. Disconnect from the database.

```
<?
   DBDisconnect();
?>
```

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

The file add.php 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.php*.

add.php
1. Start with a standard HTML page containing EditLive!. To see the steps involved in creating this page please see the EditLive! Basic PHP Example.

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.php*, 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.php" method="POST" name="articleForm">
```

4. Add a 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.

```
<?
   //the initial content to load into ELJ
   $contents = "<p>This is the initial source</p>"
?>
```

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

rawurlencode 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.php
1. Include _i_database.php_ and initialize the redirect flag.

```
// include the database wrapper functions
require_once("./i_database.php");
// redirect flag, used to cancel the redirect if we have errors
$redirect = true;
```

2. Attempt to connect to the database, and cancel the redirect if there is an error.

```
if (!DBConnect()) {
   print DBError();
   $redirect = false;
} else {
```

3. Create 3 strings to store the article title, content, and styles passed via POST, and escape them for SQL.

```
// Get the POSTed data from the form and escape it for SQL using the
// function in i_database
$articleTitle = escape_string($HTTP_POST_VARS["article_title"]);
$articleBody = escape_string($HTTP_POST_VARS["ELJApplet1"]);
$articleStyleElementText = escape_string($HTTP_POST_VARS["ELJApplet1_styles"]);
```

4. Generate the SQL query to insert the new content into the database.

```
// Insert the POSTed data into the database
$query = "INSERT INTO articles ( article_title, article_styleElementText,"
. " article_body ) VALUES ( '$articleTitle', "
. "'$articleStyleElementText', '$articleBody' )";
```

5. Execute the SQL query, and cancel the redirect if there is an error.

```
DBQuery($query);
if (DBError() != ""){
   print DBError(); $redirect = false;
}
```

6. Disconnect from the database.

```
// Disconnect is required when updating DBDisconnect(); }
```

7. If there have been no errors and the redirect flag is still set, redirect to *start.php*.

```
// redirect if no errors if($redirect){ Header("Location: start.php"); } ?>
```

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

The file *edit.php* 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.php*.

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

2. Include i_database.php.

```
<? // include the database wrapper functions
   require_once("./i_database.php");
```

3. Read the HTTP GET variable article_id to find which article the user has requested to edit.

```
// get the article id
   $articleID = $HTTP_GET_VARS["article_id"];
```

4. Attempt to connect to the database; if there is an error print it and cancel the script.

```
// Attempt to establish a connection with the database
   if (!DBConnect()) {
      print DBError();
   }else {
?>
```

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

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

6. Attempt to retrieve the current article, and cancel the script if it cannot be found. Ensure the articleID variable is escaped for SQL.

```
<?
   //SQL query to get required document from the database
   DBQuery("SELECT * FROM articles WHERE article_id = "
      . escape_string($articleID) . ";");
   if (DBFetchRow()) {
?>
```

7. Create a form to place an instance of EditLive! in. Enter the processing file name, *xt_edit.php*, 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_edit.php" method="POST" name="articleForm">
```

8. Create a hidden form object for the article id.

```
<?
   // Hidden field for identifying the article
?>
<INPUT type="hidden" name="article_id" value="<?=htmlspecialchars($article_id)?>">
```

9. 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="<?=htmlspecialchars(DBResult("article_title"))?>"
size="40"></P>
```

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

```
ELJApplet1_js.setBody("<?=rawurlencode(DBResult("article_body"))?>");
```

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

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

```
ELJApplet1_js.setStyles("<?=rawurlencode( DBResult("article_styleElementText"))?>");
```

12. 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>
```

13. Close the form tag.

```
</form>
```

14. If the article cannot be found, an error will need to be displayed.

```
<?
   } else {
      //There are no records matching this article_id //TODO: Handle and display meaningful error
   }
}
```

15. Disconnect from the database.

```
DBDisconnect(); ?> </BODY>
```

xt_edit.php
1. Include _i_database.php_ and initialize the redirect flag.

```
// include the database wrapper functions
require_once("./i_database.php");
// redirect flag, used to cancel the redirect if we have errors
$redirect = true;
```

2. Attempt to connect to the database, and cancel the redirect if there is an error.

```
if (!DBConnect()) {
   print DBError();
   $redirect = false;
} else {
```

3. Create 4 strings to store the article title, content and styles passed via POST, and escape them for SQL.

```
// Get the POSTed data from the form and escape it for SQL using the
// function in i_database
$articleID = escape_string($HTTP_POST_VARS["article_id"]);
$articleTitle = escape_string($HTTP_POST_VARS["article_title"]);
$articleBody = escape_string($HTTP_POST_VARS["ELJApplet1"]);
$articleStyleElementText = escape_string($HTTP_POST_VARS["ELJApplet1_styles"]);
```

4. Generate the SQL query to insert the new content into the database.

```
// Insert the POSTed data into the database
$query = "UPDATE articles SET article_title='$articleTitle',"
. " article_styleElementText='$articleStyleElementText',"
. " article_body='$articleBody' WHERE article_id=$articleID";
```

5. Execute the SQL query, and cancel the redirect if there is an error.

```
DBQuery($query);
if (DBError() != ""){
   print DBError(); $redirect = false;
}
```

6. Disconnect from the database.

```
// Disconnect is required when updating
DBDisconnect(); }
```

7. If there have been no errors and the redirect flag is still set, redirect to *start.php*.

```
// redirect if no errors
if($redirect){ Header("Location: start.php"); } ?>
```

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

The file *delete.php* 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.php*.

delete.php
1. Include i_database.php.

```
<?
   // include the database wrapper functions
   require_once("./i_database.php");
```

2. Read the HTTP GET variable article_id to find which article the user has requested to edit.

```
// get the article id
$articleID = $HTTP_GET_VARS["article_id"];
```

3. Attempt to connect to the database; if there is an error, print it and cancel the script.

```
// Attempt to establish a connection with the database
if (!DBConnect()) {
   print DBError();
} else {
```

4. Attempt to retrieve the current article, and cancel the script if it cannot be found. Ensure the articleID variable is escaped for SQL.

```
 // SQL query to get required document from the database
   DBQuery("SELECT * FROM articles WHERE article_id = ". escape_string($articleID) . ";");
   if (DBFetchRow()) {
?>
```

5. 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>
      <TITLE>Delete Article ID <?=htmlspecialchars($articleID)?>?</TITLE>
   </HEAD>
   <BODY>
      <H1>Delete a Document</H1>
      <P>Are you sure you wish to delete this article?</P>
```

6. Create a table in the HTML page that displays the article id and title using the DBResult function.

```
<TABLE>
   <TR>
      <TD>Article ID:</TD>
      <TD><?=htmlspecialchars($articleID)?></TD>
   </TR>
   <TR>
      <TD>Title:</TD>
      <TD><?=htmlspecialchars(DBResult("article_title"))?></TD>
   </TR>
</TABLE>
```

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

```
<FORM action="xt_delete.php" method="GET">
```

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

```
<INPUT type="hidden" name="article_id" value="<?=htmlspecialchars($articleID)?>">
```

9. 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>
```

10. Close the form and document tags.

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

11. If the article cannot be found, an error will need to be displayed.

```
<?
   } else {
      //There are no records matching this article_id
      //TODO: Handle and display meaningful error
   }
}
```

12. Disconnect from the database.

```
  DBDisconnect();
?>
```

xt_delete.php
1. Include i_database.php and initialize the redirect flag.

```
// include the database wrapper functions
require_once("./i_database.php");
// redirect flag, used to cancel the redirect if we have errors
$redirect = true;
```

2. Attempt to delete the article, and cancel the script if it cannot be found. Ensure the articleID variable is escaped for SQL.

```
if (!DBConnect()){
   print DBError();
   $redirect = false;
} else {
```

3. Attempt to delete the article, and cancel the script if it cannot be found. Ensure the articleID variable is escaped for SQL.

```
DBQuery("DELETE FROM articles WHERE article_id = " . escape_string($articleID) . ";");

if (DBError() != "") {
   print DBError();
   $redirect = false;
}
```

4. Disconnect from the database.

```
// Disconnect is required when updating
DBDisconnect();
}
```

5. If there have been no errors and the redirect flag is still set, redirect to *start.php*.

```
   // redirect if no errors
   if($redirect){
      Header("Location: start.php");
   }
?>
```

Viewing an Article (view.php)

1. Include i_database.php.

```
<?
   // include the database wrapper functions
   require_once("./i_database.php");
```

2. Read the HTTP GET variable article_id to find which article the user has requested to edit.

```
// get the article id
$articleID = $HTTP_GET_VARS["article_id"];
```

3. Attempt to connect to the database. If there is an error, print it and cancel the script.

```
// Attempt to establish a connection with the database
if (!DBConnect()) {
   print DBError();
} else {
```

4. Attempt to retrieve the current article, and cancel the script if it cannot be found. Ensure the articleID variable is escaped for SQL.

```
   //SQL query to get required document from the database
   DBQuery("SELECT * FROM articles WHERE article_id = " . escape_string($articleID) . ";");
   if (DBFetchRow()) {
?>
```

5. Use embedded PHP to both 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>
      <TITLE><?=DBResult("article_title")?></TITLE>
      <STYLE> <?=DBResult("article_styleElementText")?> </STYLE>
   </HEAD>
   <BODY>
      <H1><?=DBResult("article_title")?></H1>
      <?=DBResult("article_body")?>
   </BODY>
</HTML>
```

6. If the article cannot be found, an error will need to be displayed.

```
<?
   } else {
      //There are no records matching this article_id
      //TODO: Handle and display meaningful error
   }
}
```

7. Disconnect from the database.

```
   DBDisconnect();
?>
```

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 PHP HTTP File Upload Handler Script section of the EditLive! SDK.