# Cracking The Vault (part 2)

## By Vikram Vaswani

# Table of Contents

# Filling In The Blanks

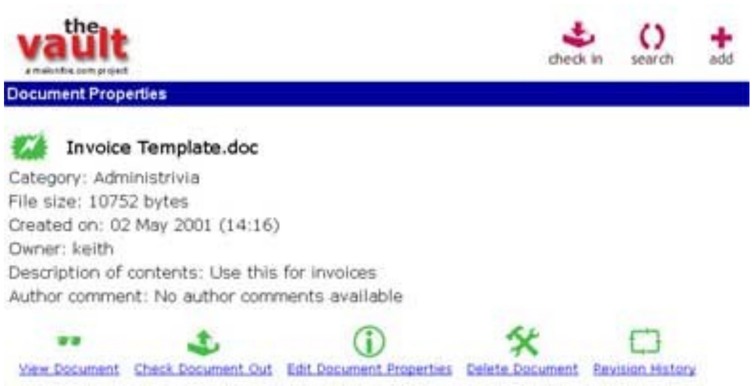
In the first part of this article, I explained my rationale for an application I like to call The Vault, a centralized document management system which uses different types of permissions to protect access to group documents. I then spent some time building a wish list of features that I would like the application to support, followed by an explanation of the rules required to govern the system, and of the document check−in/check−out process I plan to use (modeled on the techniques employed by source−control systems like CVS)

With all the theory out of the way, I then proceeded to design a database schema that supported my feature set and rules, and also wrote a few scripts designed to simplify user interaction with the system. However, I did not write the most important scripts − those that take care of actually checking documents in and out of the system − or discuss the revision history mechanism.

I plan to address both these items, and a few more, over the next few pages. So keep reading.

## Checking It Out

You'll remember from last time's article that an option to check out a file appears on the document information page, provided that the file is not checked out to someone else and the user has "modify" rights.



Document Properties

Invoice Template.doc
Category: Administrivia
File size: 10752 bytes
Created on: 02 May 2001 (14:16)
Owner: keith
Description of contents: Use this for invoices
Author comment: No author comments available

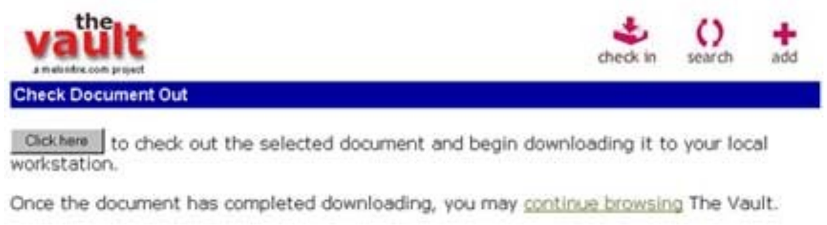
```
<?
// from details.php
$query2 = "SELECT status FROM data, perms WHERE perms.fid =
'$id' AND
perms.uid = '$SESSION_UID' AND perms.rights = '2' AND
data.status = '0' AND
data.id = perms.fid";
$result2 = mysql_db_query($database, $query2, $connection) or
die ("Error
in query: $query2. " . mysql_error());

if(mysql_num_rows($result2) > 0)
{
// if so, display link for checkout
?>
<td align="center"><a href="check-out.php?id=<? echo $id;
?>"><img
src="images/co.jpg" width=40 height=40 alt=""
border="0"><br><font
size="-1">Check Document Out</font></a></td>
<?
}
?>
```

Clicking the link will take the user to "check−out.php", and also pass the script a file ID.

```
<?
// check−out.php − performs checkout and updates database

// check for session and $id

// includes

// verify that user has modify rights
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");
$query = "SELECT id, realname FROM data, perms WHERE id =
'$id' AND
perms.rights = '2' AND perms.uid = '$SESSION_UID' AND
perms.fid = data.id
AND status = '0'";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// error check

// all ok, proceed!

if (!$submit)
{
// form not yet submitted
// display information on how to initiate download
?>
<html>
<head>
<basefont face="Verdana">
</head>

<body bgcolor="white">

<? include("menu.inc");?>

<table width="100%" border="0" cellspacing="0"
cellpadding="3">
<tr>
<td bgcolor="#0000A0">
<b><font face="Arial" color="White">Check Document
Out</font></b>
</td>
</tr>
</table>

<p>

<form action="<? echo $PHP_SELF?>" method="post">
<input type="hidden" name="id" value="<? echo $id; ?>">
<input type="submit" name="submit" value="Click here"> to
check out the
selected document and begin downloading it to your local
workstation.
</form>
Once the document has completed downloading, you may <a
href="out.php">continue browsing</a> The Vault.
</body>
</html>
<?
}
// form submitted − download
else
{
list($id, $realname) = mysql_fetch_row($result);
mysql_free_result($result);

// since this user has checked it out and will modify it
// update db to reflect new status
$query = "UPDATE data SET status = '$SESSION_UID' WHERE id =
'$id'";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// get the filename
$filename = $dataDir . $id . ".dat";

// send headers to browser to initiate file download
header ("Content−Type: application/octet−stream");
header ("Content−Disposition: attachment;
filename=$realname");
readfile($filename);
}

// clean up
mysql_close($connection);
?>
```

After a few basic error checks, the script produces some simple instructions − click a button to initiate file download, or click a link to go back to the main document listing. In this case, the button is actually a form (more on this later), which, once submitted, UPDATEs the database to reflect that the file has now been checked out to the current user, then generates the filename (based on the file ID), and sends HTTP headers to the browser to prepare it for a file download. Note that the filename sent in the "Content−Disposition: " header is the original name of the file, as stored in the database, and not the internal name of the file.

Once the browser receives the headers, it should pop up a "Save As" dialog box, allowing the user to save the file to his or her local workstation, where it can be modified and edited. While the file is checked out to a user, other users will not see the check−out menu option, and the file listing in "out.php" will indicate that the file is checked out to a specific user via a red storm−cloud icon (you may remember this code from the previous article).



to check out the selected document and begin downloading it to your local workstation.

Once the document has completed downloading, you may continue browsing The Vault.

You'll notice that I've used a form to call the script which actually initiates the download. My original stab at this was to simply call the script and pass it the file ID via the URL GET method − for example, "check−out.php?id=13". However, while this technique worked without a problem in Netscape and Lynx browsers, and even in version 5.0 of Internet Explorer, I noticed a problem with Internet Explorer 5.5; the browser chokes if asked to download a script containing GET−type parameters. Consequently, I decided to use a form and pass parameters via the POST method instead.

Some users have also reported another strange problem with Internet Explorer 5.5 − rather than downloading the target file, the browser has a nasty tendency to download the calling script instead. I plan to look into this at some point − if you have any ideas on what this is all about, let me know!

Finally, Internet Explorer may also display an annoying tendency to display the file in the browser, rather than download it, if the file is a recognized format (text, image et al). The workaround here is to change the "Content−Type" header in the script above to something the browser will not recognize − such as

```
<?
// snip
header ("Content−Type: application/octetstream");
?>
```

You can read more about this problem athttp://ppewww.ph.gla.ac.uk/~flavell/www/content−type.html and http://msdn.microsoft.com/workshop/networking/moniker/overview/appendix_a.asp

## Room With A View

You'll remember that users also have the ability to view the most current version of a document, regardless of whether or not they can check it out.

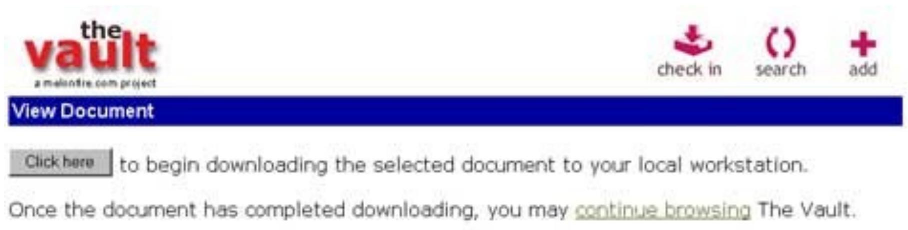
```
<!-- from details.php -->
<td align="center"><a href="view.php?id=<? echo $id; ?> "><img
src="images/view.jpg" width=40 height=40 alt=""
border="0"><br><font
size="-1">View Document</font></a></td>
```

The manner in which this is handled is almost identical to the check−out process, except that this time, I'm calling "view.php" instead of "check−out.php". And "view.php" does the same thing as "check−out.php", initiating an immediate file download. However, since downloading a file for viewing should not render it inaccessible to other users, "view.php" does not UPDATE the database, leaving the file status field as is.

Here's "view.php".

```
<?
// view.php − performs download without updating database

// checks and includes

// verify again that user has view rights
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");
$query = "SELECT id, realname FROM data, perms WHERE id =
'$id' AND
perms.rights = '1' AND perms.uid = '$SESSION_UID' AND
perms.fid = data.id";

// all checks completed

// form not yet submitted
// display information on how to initiate download
if (!$submit)
{
?>
<html>
<head>
<basefont face="Verdana">
</head>

<body bgcolor="white">

<? include("menu.inc");?>

<table width="100%" border="0" cellspacing="0"
cellpadding="3">
<tr>
<td bgcolor="#0000A0">
<b><font face="Arial" color="White">View Document</font></b>
</td>
</tr>
</table>

<p>

<form action="<? echo $PHP_SELF?>" method="post">
<input type="hidden" name="id" value="<? echo $id; ?>">
<input type="submit" name="submit" value="Click here"> to
begin
downloading the selected document to your local workstation.
</form>
Once the document has completed downloading, you may <a
href="out.php">continue browsing</a> The Vault.
</body>
</html>
<?
}
// form submitted − begin download
else
{
list($id, $realname) = mysql_fetch_row($result);
mysql_free_result($result);

// get the filename
$filename = $dataDir . $id . ".dat";

// send headers to browser to initiate file download
header ("Content−Type: application/octet−stream");
header ("Content−Disposition: attachment;
filename=$realname");
readfile($filename);
}
}
// clean up
mysql_close($connection);
```

```
?>
```



to begin downloading the selected document to your local workstation.

Once the document has completed downloading, you may continue browsing The Vault.

## All Revved Up

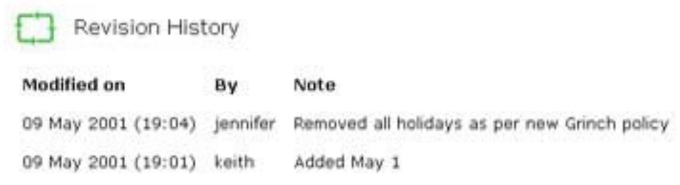
You'll remember that the document information page, "details.php", also contains a link to the document's revision history. That history is generated by the script "history.php", which queries the "log" table to build a list of changes made to the document.

The first part of the script simply provides the same document information seen in "details.php" − the description, size and so on. Once that's over with, a query to the "log" and "user" tables builds a list of changes made to the specific file, sorted by date.

---

```
<table border="0" cellspacing="5" cellpadding="5">
<tr>
<td><font size="−1"><b>Modified on</b></font>
<td><font size="−1"><b>By</b></font>
<td><font size="−1"><b>Note</b></font> </td>
</tr>
<?

// query to obtain a list of modifications
$query = "SELECT user.username, log.modified_on, log.note FROM
log, user
WHERE log.id = '$id' AND user.id = log.modified_by ORDER BY
log.modified_on
DESC";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// iterate through resultset
while(list($modified_by, $modified_on, $note) =
mysql_fetch_row($result))
{
?>
<tr>
<td><font size="−1"><? echo fixDate($modified_on);
?></font></td>
<td><font size="−1"><? echo $modified_by; ?></font></td>
<td><font size="−1"><? echo $note; ?></font></td>
</tr>
<?
}
// clean up
mysql_free_result($result);
?>
</table>
```

Here's a sample screen.



Revision History

| Modified on | By | Note |
|---|---|---|
| 09 May 2001 (19:04) | jennifer | Removed all holidays as per new Grinch policy |
| 09 May 2001 (19:01) | keith | Added May 1 |

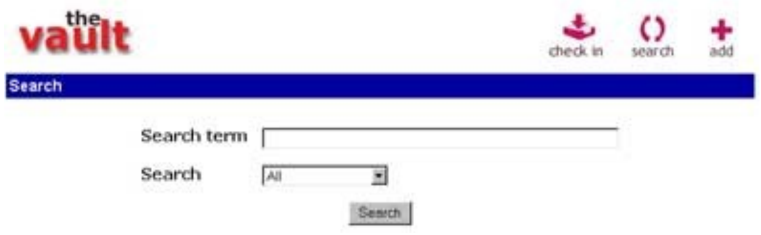
## Looking For Something?

The final feature I'd like to add is a search capability, to enable users to quickly drill down to the documents matching specific criteria. At the moment, the search "engine" is very primitive, allowing users to only query document descriptions, document names, and document comments for specific keywords. If required, this can easily be improved upon; for a baseline release, it will suffice.

The search feature is accessible from the main menu, and links to "search.php", which contains a simple form.

```
<table border="0" cellspacing="5" cellpadding="5">
<form action="out.php" method="POST">

<tr>
<td valign="top"><b>Search term</b></td>
<td><input type="Text" name="keyword" size="50"></td>
</tr>

<tr>
<td valign="top"><b>Search</b></td>
<td><select name="where">
<option value="1">Descriptions only</option>
<option value="2">Filenames only</option>
<option value="3">Comments only</option>
<option value="4" selected>All</option>
</select></td>
</tr>

<tr>
<td colspan="2" align="center"><input type="Submit"
name="submit"
value="Search"></td>
</tr>

</form>
</table>
```

The form variable $keyword contains the search term, while the variable $where contains a number indicating which table column to search against.



You will notice that this form actually submits data to "out.php". How can this be? What does it mean? Am I out of my tiny little gourd?

My original plan was to have "search.php" itself process the search via a SELECT query to the database. I wrote the query I planned to use, and the code looked something like this

```
$query = "SELECT data.id, user.username, data.realname,
data.created,
data.description, data.comment, data.status FROM data, user,
perms WHERE
data.id = perms.fid AND user.id = data.owner AND perms.uid =
'$SESSION_UID'
AND perms.rights = '1'";
```

Depending on the contents of $where, this would be further modified − for example, if

```
$where == true
```

the query would read

```
$query = "SELECT data.id, user.username, data.realname,
data.created,
data.description, data.comment, data.status FROM data, user,
perms WHERE
data.id = perms.fid AND user.id = data.owner AND perms.uid =
'$SESSION_UID'
AND perms.rights = '1' AND data.description LIKE
```

```
'%$keyword%'";
```

and so on.

After a little bit of thought, I realized that the first part of the query was identical to that used in "out.php" to generate an initial document listing...which meant that I could save myself some time by using that script (with some modifications) as my search results page.

Here are the changes I finally made to "out.php".

```
<?

// my original out.php query
// get a list of documents the user has "view" permission for
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");
$query = "SELECT data.id, user.username, data.realname,
data.created,
data.description, data.comment, data.status FROM data, user,
perms WHERE
data.id = perms.fid AND user.id = data.owner AND perms.uid =
'$SESSION_UID'
AND perms.rights = '1'";

// if coming from the search form, $keyword and $where will
exist
// so modify the query with additional constraints
if ($keyword != "" &isset($where))
{

// switch loop
switch ($where)
{
// description search
case 1:
$query .= " AND (data.description LIKE '%$keyword%')";
break;

// filename search
case 2:
$query .= " AND (data.realname LIKE '%$keyword%')";
break;
// comment search
case 3:
$query .= " AND (data.comment LIKE '%$keyword%')";
break;

// search all!
case 4:
$query .= " AND (data.description LIKE '%$keyword%' OR
data.realname LIKE
'%$keyword%' OR data.comment LIKE '%$keyword%')";
break;
}

}

$query .= " ORDER BY created DESC";
```

And now, if "out.php" receives the $keyword and $where variables, it will "know" that a search is being conducted and will modify the query with additional constraints so as to display only documents which match the search criteria. Cool, huh?

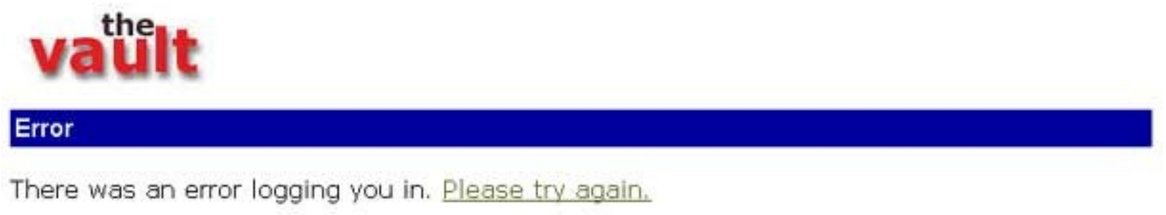
## Oops!

The last script − and the simplest − is the error handler, "error.php". If you look at the source code, you'll notice many links to this script, each one passing it a cryptic error code via the $ec variable. Very simply, "error.php" intercepts the variable and converts it to a human−readable error message, which is then displayed to the user.

---

```
<?
// error.php − displays error messages based on error code $ec

// includes
include("config.php");

switch ($ec)
{
// login failure
case 0:
$message = "There was an error logging you in. <a
href=start.html>Please
try again.</a>";
break;

// session problem
case 1:
$message = "Please <a href=start.html>log in</a> again.";
break;

// malformed variable/failed query
case 2:
$message = "There was an error performing the requested
action. Please <a
href=start.html>log in</a> again.";
break;

// file not uploaded
case 11:
$message = "Please upload a valid document.";
break;

// rights not assigned
case 12:
$message = "You must assign view/modify rights to at least one
user.";
break;
// illegal file type
case 13:
$message = "That file type is not currently
supported.<p>Please upload a
document conforming to any of the following file types:<br><ul
align=left>";

foreach($allowedFileTypes as $this)
{
$message .= "<li>$this";
}
$message .= "</ul>";
break;

default:
$message = "There was an error performing the requested
action. Please <a
href=start.html>log in</a> again.";
break;

}

?>
<html>
<head>
<basefont face="Verdana">
</head>

<body bgcolor="White">

<p>
<? echo $message; ?>
</body>
</html>
```

Here's what it looks like.



There was an error logging you in. Please try again.

Simple and elegant − not to mention flexible. Found a new error? No problem − assign it an error code and let "error.php" know.

## Endgame

That just about concludes this little tour of the various scripts that make up The Vault. Throughout this development effort, I have made widespread use of PHP's session management capabilities, file and string functions, HTTP headers, and mySQL database queries. If you are new to PHP, I hope that the effort has been instructive, and that it has helped you gain a greater understanding of these powerful open−source tools.

It should be noted at this point that this project is by no means complete. Since The Vault was introduced for internal use a few weeks back, a number of minor bugs have been reported, and some additional capabilities requested. Among the features requested: the ability to quickly display documents in specific categories; to sort listings by name, size, date and owner; to search by revision log comments; to limit the number of files displayed per page; to change the default colours; and to display the name of the currently logged−in user.

Additionally, once I have confirmation from users that the system, as designed, meets their needs, I would like to review the design once again, with particular emphasis on further modifying both the database schema and the SQL queries. This process should take place in conjunction with the development plan for new features, so that the addition of new features does not add to overhead and cause performance degradation.

I plan to continue adding features and optimizing code, as and when time permits − if you'd like to give me a hand, or have ideas on how to improve the techniques discussed here, drop me a line and let me know. Ciao!