



Time is MONEY (Part One)

By The Disenchanted Developer

This article copyright [Melonfire](#) 2000–2002. All rights reserved.

Table of Contents

<u>Act One, Scene One</u>	1
<u>Up A Creek</u>	2
<u>Bills, Bills, Bills</u>	3
<u>So Many Tables, So Little Time</u>	4
<u>Open Sesame</u>	9
<u>The Lazy Programmer Strikes Again</u>	12
<u>Today's Menu</u>	14
<u>Too Much Information</u>	20
<u>Time For Bed</u>	24

Act One, Scene One

So there we all are, the Boss, the Customer, a couple of the Customer's minions and yours truly, all seated around a long table in the Customer's high-tech conference room. The Customer is talking around an evil-smelling cigar and waving his hands around energetically, the Boss is listening patiently, and the minions are grinning idiotically at each other. I have long since tuned out of the conversation, and am currently attempting to figure out which movie to rent over the weekend.

Outside, it is a sunny April morning, the kind to make you think of picnics in shady gardens and poolside barbecues in the long evenings. Inside, all hell is about to break loose.

"...and both HRD and I would be very interested in seeing those numbers," says the Customer, pausing for a minute to puff some smoke and then jabbing his cigar at the Boss. "But the question is, can you guys do it for us, and how soon can we see a prototype?"

The Boss turns to me and asks politely, "Well?"

What on earth does the man want? No time to figure it out, better stall...

"Sure," I mumble, trying to sound as though I'm giving the matter great thought, "but perhaps we could backtrack a bit and go over the requirements in detail..."

"No, no", says the Boss impatiently, "I think the requirements are quite clear. They need a simple timesheet application so that they can analyze employee hours and bill clients appropriately. It's not a big deal, but they need it in...three days, right?," this to the Customer, who's now gone back to blowing smoke rings at the ceiling.

"Well," says the Customer with a smirk, "as soon as possible, actually."

"So, will we be able to deliver in that time frame?," says the Boss, giving me his look, the one which reads "say yes or else..."

"Of course," I say, just like a good soldier. "Absolutely no problem," I assure the room at large, smiling at the Customer's cigar. "What have I done?," I ask myself silently.

Up A Creek

For those of you not too good at reading between the lines, let me tell you what just happened.

The Customer, who happens to be a very successful lawyer, and the head of one of the city's biggest legal firms, wants to generate invoices based on hours worked by his employees. The Customer's HRD department wants to monitor and analyze employee work hours, for reasons best known to themselves. And they're looking to us to build the application to do it.

The Boss just volunteered me to develop the application for the Customer. Since the Customer happens to be very rich and an association with him would probably prove to be lucrative (and, I suspect, the cause of many ulcers), the Boss is very keen to make a good first impression...which is probably why he agreed to that ridiculous three-day deadline as well.

"Look," he says to me when I bring up the topic in the cab, "you're one of my best developers. We need to do this, because those guys are looking for a new software contractor and this is our foot in the door. You keep telling me about that RAD thing you like so much – HPP, PPH," (he means PHP, the moron), "whatever you call it. Why can't you use that, put it together, make it look pretty and send it across in a couple days? Trust me," (oh no, I'm thinking, here it comes, the Boss's favourite maxim) "I'm sure it isn't as difficult as you're making it out to be."

How on earth did this guy get to be CEO of a software development company?

Anyhow, it looks like I'm up the proverbial creek without the equally-proverbial paddle. I need to develop the requirements for this application, design an appropriate database schema, put together the code, package it in a pretty interface, test it and deliver it...all within the next seventy-two hours.

With the help of powerful open-source tools like PHP and mySQL, the process can be simplified considerably. And over the course of this article, I'm going to demonstrate how, by building a PHP/mySQL-based timesheet application suitable for small businesses or independent contractors.

The goal here is two-fold: to introduce novice and intermediate programmers to the process of designing and implementing a Web-based application; and to offer HR managers, accountants, corporate efficiency experts and other interested folk a possible solution to their woes.

Let's get started!

Bills, Bills, Bills

Before getting into the nitty-gritty of syntax and structure, it's important to understand the problems this application will be addressing, so that the functional requirements of the solution become clear. As the Customer explained it, his HRD and Billing people had the following two problems:

1. Most law firms and consultancies bill on an hourly basis. If employees log their hours on paper worksheets and submit them to Billing at the end of every day, it falls to the poor guys in Billing to add up the hours worked on specific projects and generate appropriate invoices. An electronic system, available via the local intranet to every employee, would help tremendously, as the tasks of classification and summation could be handled by the system.
2. Human Resources would also like a peek at the data, in order to locate and resolve staffing and resource allocation issues. It would be helpful to know, for example, the projects (and the tasks within projects) which generate the most activity, so that resources can be allocated to those projects and tasks (or new employees hired to cover the shortfall, if any). It would also be helpful to view activity by user, in order to identify which users are the most productive, which users make the most contributions to specific projects, and which users spend their day playing Tetris.

Having understood the problems, it becomes easier to decide on the requirements of the solution. An analysis of the problems above reveals that most of them would be resolved by a system which:

1. maintained a list of active projects;
2. broke up activity on a project into standard tasks, and maintained a list of these standard tasks;
3. allowed employees to log work hours by project and task, on a daily basis;
4. provided reports on the hours worked by different users on a specific project, together with summary totals (to simplify billing, and to understand employee contributions to different projects);
5. provided reports on the hours spent by users on the different components of a specific project, together with summary totals (to assist in monitoring employee efficiency and resource allocation);
6. provided summary reports on resource allocation across different projects for a specific period of time (to allow managers to develop a big-picture view of employee activity and make corrections, if required)

This, therefore, constitutes the initial feature set for the application, and serves as the guideline for any future development.

It should be noted as this point that these requirements (and consequently, this application) are not limited only to law firms or software consultancies. Information like this would be useful to any company, regardless of size, simply because it would help in analyzing and monitoring employee activity, tracking resource allocation and usage, and resolving possible flaws in internal business processes and systems.

So Many Tables, So Little Time

Having written down the requirements, it becomes much easier to begin designing the architecture of the system. The first (and most important) part of this design process is database design, in which I will be designing the tables to hold application data.

This is a good time for you to download the source code, so that you can refer to it throughout this article (you will need a Web server capable of running PHP and a mySQL database).

[time.zip](#)

First, I need a table to hold the list of active projects – let's call that the "projects" table:

```
#
# Table structure for table 'projects'
#

DROP TABLE IF EXISTS projects;
CREATE TABLE projects (
  pid smallint(5) unsigned NOT NULL auto_increment,
  pname varchar(40) NOT NULL,
  pdesc text NOT NULL,
  PRIMARY KEY (pid)
);

#
# pid - unique project ID
# pname - project name
# pdesc - project description
#
```

Here are some dummy entries:

```
#
# Dumping data for table 'projects'
#

INSERT INTO projects (pid, pname, pdesc) VALUES ( '1',
'XTech.com',
'Interface design and development of the XTech corporate
site');

INSERT INTO projects (pid, pname, pdesc) VALUES ( '2',
'Melonfire.com',
'Weekly updates of the Melonfire content catalog');
```



Time is Money (part 1)

```
INSERT INTO projects (pid, pname, pdesc) VALUES ( '3',
'VideoMoz (Linux)',
'Software development of the VideoMoz animation and graphics
library for
the Linux platform');
```

```
INSERT INTO projects (pid, pname, pdesc) VALUES ( '4',
'VideoMoz
(Windows)', 'Software development of the VideoMoz.dll
animation library for
Windows 98, 2000 and XP');
```

```
INSERT INTO projects (pid, pname, pdesc) VALUES ( '5',
'NamelessCorp
AddBook', 'Design and development of address book application
for the
NamelessCorp intranet');
```

```
INSERT INTO projects (pid, pname, pdesc) VALUES ( '6',
'NamelessCorp
invDB', 'Design of invoicing database for NamelessCorp
Accounting
department');
```

I also need a table to hold the list of standard, company-defined, billable tasks – the "tasks" table:

```
#
# Table structure for table 'tasks'
#

DROP TABLE IF EXISTS tasks;
CREATE TABLE tasks (
tid tinyint(3) unsigned NOT NULL auto_increment,
tname varchar(40) NOT NULL,
tdesc text NOT NULL,
PRIMARY KEY (tid)
);

#
# tid - unique task ID
# tname - task name
# tdesc - task description
#
```



Time is Money (part 1)

As you can see, it's almost identical to the "projects" table – except, obviously, for the data it contains:

```
#
# Dumping data for table 'tasks'
#

INSERT INTO tasks (tid, tname, tdesc) VALUES ( '1', 'Design',
'Interface
design, software architecture design, database schema
design');

INSERT INTO tasks (tid, tname, tdesc) VALUES ( '2',
'Development',
'Development of software code, standard libraries and
functions');

INSERT INTO tasks (tid, tname, tdesc) VALUES ( '3', 'System
Test',
'Verifying software functionality, creating test cases,
writing bug
reports');

INSERT INTO tasks (tid, tname, tdesc) VALUES ( '4', 'Technical
Support',
'Supporting customers (phone/fax/email/online/site), assisting
in software
(un)installation, answering user questions');

INSERT INTO tasks (tid, tname, tdesc) VALUES ( '5',
'Documentation',
'Creating technical manuals, software data specifications,
product catalogs
and marketing literature');

INSERT INTO tasks (tid, tname, tdesc) VALUES ( '6',
'Training', 'Executing
training programs for customers (online/site)');

INSERT INTO tasks (tid, tname, tdesc) VALUES ( '7',
'Research',
'Understanding new technology, experimenting with new
applications and
tools');
```

Next, we need a table to hold the list of users allowed to use the applications, together with their passwords. Each user is assigned a unique ID, which will be used throughout the application



Time is Money (part 1)

```
#
# Table structure for table 'users'
#

DROP TABLE IF EXISTS users;
CREATE TABLE users (
  uid tinyint(3) unsigned NOT NULL auto_increment,
  uname varchar(255) NOT NULL,
  upass varchar(255) NOT NULL,
  uperms tinyint(4) DEFAULT '0' NOT NULL,
  PRIMARY KEY (uid),
  UNIQUE uname (uname)
);

#
# uid - unique user ID
# uname - user's log-in name
# upass - user's password
# uperms - user's permission level (user or admin)
#
```

You'll notice the last column in this table contains a permission level for each user. I need this in order to categorize users into two types – regular users or administrators. Only administrators should have the ability to generate summary reports for user activity.

Here is some seed data for this table, which identifies users "joe" and "sherry" to be administrators (in case you're wondering, the passwords in this dummy data are the same as the corresponding username – they've just been encrypted with mySQL's password() function)

```
#
# Dumping data for table 'users'
#

INSERT INTO users (uid, uname, upass, uperms) VALUES ( '1',
'john',
'2ca0ede551581d29', '0');

INSERT INTO users (uid, uname, upass, uperms) VALUES ( '2',
'joe',
'7b57f28428847751', '1');

INSERT INTO users (uid, uname, upass, uperms) VALUES ( '3',
'vanessa',
'24b841bb4fef7fda', '0');

INSERT INTO users (uid, uname, upass, uperms) VALUES ( '4',
```



Time is Money (part 1)

```
'sherry',  
'12ee5cff47618c7a', '1');
```

It should be noted that these three tables would typically need to be controlled by an administrator, who would be in charge of adding (and removing) new users and categories to the system.

In case you're wondering why I've split these items into separate tables, rather than including them all in a single table, or even hard-coding them into the application, the reason is very simple: I want to make it easier for an administrator to add and edit these values.

By breaking them into separate tables, an administrator who wants to customize the application (for example, add new projects, or edit the various tasks) can do so without having to mess about with the program code. This is part of a process known as "normalization", and it's very important when designing a database with two or more tables (links to some good articles on normalization appear at the end of this article)

Finally, we need a table to hold the work hours entered by individual employees, and map these hours to a specific project and task – the "log" table:

```
#  
# Table structure for table 'log'  
#  
  
DROP TABLE IF EXISTS log;  
CREATE TABLE log (  
  lid tinyint(3) unsigned NOT NULL auto_increment,  
  pid tinyint(3) unsigned DEFAULT '0' NOT NULL,  
  tid tinyint(3) unsigned DEFAULT '0' NOT NULL,  
  uid tinyint(3) unsigned DEFAULT '0' NOT NULL,  
  hours float unsigned DEFAULT '0' NOT NULL,  
  date date DEFAULT '0000-00-00' NOT NULL,  
  PRIMARY KEY (lid)  
);  
  
#  
# lid - unique record ID  
# pid - project ID (foreign key to "projects" table)  
# tid - task ID (foreign key to "tasks" table)  
# uid - user ID (foreign key to "users" table)  
# hours - hours worked  
# date - date on which hours worked  
#
```

Entries to this table will be made by individual users through the application, and the data in this table will eventually be used to generate summary reports.



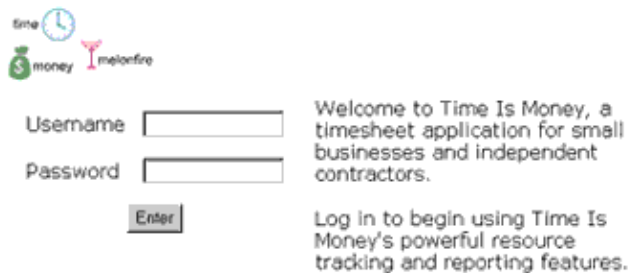
Open Sesame

With the database designed and out of the way, it's time to actually start writing some code. First up, the user login process, and the scripts which verify the user's password and grant him access to the system.

Here's the initial login form, "index.html".

```
<table border="0" cellspacing="5" cellpadding="5">
<form action="login.php" method="post">
<tr>
<td>Username</td>
<td><input type="Text" name="frmuser" size="15"></td>
</tr>
<tr>
<td>Password</td>
<td><input type="password" name="frmpass" size="15"></td>
</tr>
<tr>
<td colspan="2" align="CENTER"><input type="Submit"
name="submit"
value="Enter"></td>
</tr>
</form>
</table>
```

Here's what it looks like:



The screenshot shows a login form for 'Time Is Money'. At the top left, there are three small icons: a clock labeled 'time', a money bag labeled 'money', and a person labeled 'melonfire'. Below these icons are two input fields: 'Username' and 'Password'. To the right of the 'Password' field is a button labeled 'Enter'. To the right of the input fields, there is a welcome message: 'Welcome to Time Is Money, a timesheet application for small businesses and independent contractors.' Below this message is another line of text: 'Log in to begin using Time Is Money's powerful resource tracking and reporting features.'

Once the form is submitted, the data is processed by "login.php", which connects to the database to verify the username and password against the "users" table.

```
<?
// login.php - verifies user login

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

Time is Money (part 1)

```
// check login and password
// connect and execute query
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");
$query = "SELECT uid, uperms from users WHERE uname =
'$firmuser' AND upass
= PASSWORD('$firmpass)";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// if row exists - login/pass is correct
if (mysql_num_rows($result) == 1)
{
// initiate a session
session_start();

// register the user's ID and permission level
session_register("SESSION_UID");
session_register("SESSION_UPERMS");
list($uid, $uperms) = mysql_fetch_row($result);
$SESSION_UID = $uid;
$SESSION_UPERMS = $uperms;

// redirect to main menu page
header("Location:menu.php");
mysql_free_result ($result);

// close connection
mysql_close($connection);
}
else
// login/pass check failed
{
mysql_free_result ($result);
mysql_close($connection);
// redirect to error page
header("Location: error.php?ec=0");
exit;
}
?>
```

Assuming the username and password is correct, the script initiates a session, and registers two session variables, `$SESSION_UID` (which contains the user's ID) and `$SESSION_UPERMS` (which contains the user's permission level). These variables will remain available throughout the session, and will be used in many of the subsequent scripts. The script then redirects the browser to "menu.php", which sets up the main



Time is Money (part 1)

menu for the system, via an HTTP header.

A login failure will redirect the browser to the generic error handler, "error.php", with an error code indicating the type of error. I'll be using this error handler extensively, to handle the different types of errors possible.

It is important to note that calls to header() and session_start() must take place before *any* output is sent to the browser. Even something as minor as whitespace or a carriage return outside the PHP tags can cause these calls to barf all over your script.

Finally, the include()d files, "config.php" and "functions.php", contain variables and functions which will be used throughout the application. The most important of these are the database name, user name and password, which are stored in "config.php" – take a look:

```
<?
// config.php - useful variables/functions

// database parameters
// alter this as per your configuration
$database="timesheet";
$user = "time_agent";
$pass = "gs645kaf";
$hostname = "localhost";
?>
```

The Lazy Programmer Strikes Again

Once the user is successfully logged in, "menu.php" takes over and generates a menu of functions available to the user.

The first thing "menu.php" (and every other script) does is to verify the existence of a valid session – this is necessary to prevent unauthorized users from viewing the pages. If a session doesn't exist, the browser is immediately redirected to the error page.

```
<?
// check for valid user session
session_start();
if(!session_is_registered("SESSION_UID"))
{
header("Location: error.php?ec=1");
exit;
}
?>
```

Assuming the session check does not fail, a basic HTML page is built.

```
<html>
<head>
</head>

<body bgcolor="white">

<?
// display page header
$title = "Main Menu";
include("header.inc.php");
?>

<?
// code to build main menu goes here
?>

<? include("footer.inc.php"); ?>

</body>
</html>
```

Before we get into the nitty-gritty of how "menu.php" works, I want to draw your attention to the manner in which each page within this application is built.



Time is Money (part 1)

Each page generated through this application has a particular layout – a logo in the top left corner and a blue bar below it containing a page title. The bottom of every page has a copyright notice and a disclaimer. Since these elements will remain constant, through the application, I've placed the corresponding HTML code in separate header and footer files, and simply include()d them on each page.

Again, by separating common interface elements into separate files, I've made it easier to customize the look of the application; simply alter these files, and the changes will be reflected on all the pages.

The variable \$title stores the title for each page, and is used by "header.inc.php" – as you can see.

```
<!-- header.inc.php -->
<table width="100%" border="0" cellspacing="0"
cellpadding="3">
<tr>
<td><a href="logout.php"></a></td>
</tr>

<tr>
<td bgcolor="#3098C3"><font color="white">&nbsp;<b><? echo
$title;
?></b></font></td>
</tr>
</table>
<p>
```



Today's Menu

The page header and footer enclose the code necessary to perform the particular script's function. In the specific case of "menu.php", this code involves setting up a main menu for the user to:

view a list of current projects, with descriptions;

view a list of standard tasks, with definitions;

view or add timesheet data for any particular day;

generate reports (if administrator);

log out of the system;

Here's the code to display these options:

```
<?
// display page header
$title = "Main Menu";
include("header.inc.php");
?>

Please select from the following options:

<ul>
<li>
<!-- view projects option -->
<a href="projects.php">View project descriptions</a>
<p>
<li>
<!-- view tasks option -->
<a href="tasks.php">View task descriptions</a>
<p>
<li>
<!-- view timesheet option -->
<form name="view" action="view.php" method="post">
<a href="javascript:submitForm(0)">View timesheet</a> for
<? generateDateSelector(); ?>
</form>
<p>
<li>
<!-- log out option -->
<a href="logout.php">Log out of the system </a>
</ul>

<? include("footer.inc.php"); ?>
```


Here's what it looks like:



This is nothing but an unordered list of links, with each link pointing to a different script. Notice that the middle link requires the user to select a date, and so I've constructed it as a separate form containing a series of date selection boxes. I've also done away with the standard submit buttons in this form, preferring instead to use a simple JavaScript to submit it when its corresponding link is clicked.

In case you're wondering about the call to the `generateDateSelector()` function, let me explain what it does. Since I will be using the same series of drop-down boxes for date selection in numerous places, I decided to save myself some time by writing a simple PHP function to generate these boxes for me on demand. This function is stored in the `include()` file "functions.php", and looks like this:

```
<?
// generate three list boxes for d-m-y selection
function generateDateSelector($prefix="")
{
// month array
$monthArray = array("", "January", "February", "March",
"April", "May",
"June", "July", "August", "September", "October", "November",
"December");

// get current year, month and date
$arr = getdate(mktime());
$currYear = $arr["year"];
$currMonth = $arr["mon"];
$currDay = $arr["mday"];

// generate date drop-down
echo "<select name=" . $prefix . "d>";
for ($x=1; $x<=31; $x++)
{
$str = "<option value=" . sprintf("%02d", $x) . " ";
if ($x == $currDay)
```

Time is Money (part 1)

```
{
$str .= " selected";
}
$str .= ">" . sprintf("%02d", $x) . "</option>";
echo $str;
}
echo "</select>";

// generate month drop-down
echo "<select name=" . $prefix . "m>";
for ($x=1; $x<=12; $x++)
{
$str = "<option value=" . sprintf("%02d", $x) . " ";
if ($x == $currMonth)
{
$str .= " selected";
}
$str .= ">" . $monthArray[$x] . "</option>";
echo $str;
}
echo "</select>";

// generate year drop-down
echo "<select name=" . $prefix . "y>";
for ($x=$currYear; $x<($currYear+5); $x++)
{
$str = "<option value=$x";
if ($x == $currYear)
{
$str .= " selected";
}
$str .= ">" . sprintf("%04d", $x) . "</option>";
echo $str;
}
echo "</select>";
}
?>
```

As you can see, the function includes code to automatically pre-select the current date, month and year, together with an optional \$prefix argument to customize the variable names for the three drop-down boxes.

The output of this function would look something like this

```
<select name=d>
<option value=01>01</option><option
value=02>02</option><option
value=03>03</option><option value=04>04</option><option
```

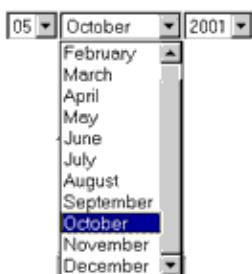
Time is Money (part 1)

```
value=05>05</option><option value=06>06</option><option
value=07>07</option><option value=08>08</option><option
value=09>09</option><option value=10>10</option><option
value=11>11</option><option value=12>12</option><option
value=13>13</option><option value=14>14</option><option
value=15>15</option><option value=16>16</option><option
value=17>17</option><option value=18>18</option><option
value=19>19</option><option value=20>20</option><option
value=21>21</option><option value=22>22</option><option
value=23
selected>23</option><option value=24>24</option><option
value=25>25</option><option value=26>26</option><option
value=27>27</option><option value=28>28</option><option
value=29>29</option><option value=30>30</option><option
value=31>31</option>
</select>

<select name=m>
<option value=01>January</option><option
value=02>February</option><option
value=03>March</option><option value=04>April</option><option
value=05>May</option><option value=06>June</option><option
value=07>July</option><option value=08
selected>August</option><option
value=09>September</option><option
value=10>October</option><option
value=11>November</option><option value=12>December</option>
</select>

<select name=y>
<option value=2001 selected>2001</option><option
value=2002>2002</option><option
value=2003>2003</option><option
value=2004>2004</option><option value=2005>2005</option>
</select>
```

or this:



There's only one thing missing from this menu – the link for administrators to use when generating reports.

Time is Money (part 1)

However, I need to check that the user currently logged-in is, in fact, an administrator before displaying this menu option. Let's take care of that next:

```
<!-- view timesheet option - snip -->

<?
// if administrator logged-in
// display report option
if ($SESSION_UPERMS == 1)
{
?>
<li>
<!-- generate report option -->
<form name="report" action="report.php" method="post">
<a href="javascript:submitForm(1)">Generate activity
reports</a> between
<? generateDateSelector("s"); ?> and <?
generateDateSelector("e"); ?> for
<select name="pid">
<option value="0">&lt;all projects&gt;</option>
<?
// open connection to database
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

// get project list
$query = "SELECT pid, pname from projects";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());
while (list($pid, $pname) = mysql_fetch_row($result))
{
echo "<option value=$pid>$pname</option>";
}
mysql_free_result($result);
?>
</select>
</form>
<p>
<?
}
?>

<!-- log out option - snip -->
```

Time is Money (part 1)

Since I would like administrators to have the ability to view reports for a specific project, I've included a drop-down box containing a list of all current projects in the menu item above. This list is generated via a query to the "projects" table.

And here's what the finished product looks like:



Main Menu

Please select from the following options:

- [View project descriptions](#)
- [View task descriptions](#)
- [View timesheet](#) for [05] [October] [2001]
- [Generate activity reports](#) between [05] [October] [2001] and [05] [October] [2001] for [all projects]
- [Log out of the system](#)

Everything here is © [Majorda](#), 2001. All rights reserved.
All source code provided without warranty - use at your own risk.

Too Much Information

With the main menu out of the way, let's now begin putting together the scripts corresponding to the different menu options. The first of these is the script "projects.php", which allows users to view a list of active projects.

This is a very simple script – all it needs to do is connect to the database, retrieve a list of project names and associated descriptions, and print them in a neat list.

```
<?
// projects.php - display project list

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

// check for valid user session
session_start();
if(!session_is_registered("SESSION_UID"))
{
header("Location: error.php?ec=1");
exit;
}
?>
<html>
<head>
<basefont face="Verdana">
<style type="text/css">
TD {font-family: Verdana; font-size: smaller}
</style>
</head>

<body bgcolor="white">

<?
// display page header
$title = "<a style=color:white href=menu.php>Main Menu</a> >
View Project
Descriptions";
include("header.inc.php");
?>

<!-- main table -->
<table width="100%" border="0" cellspacing="0"
cellpadding="0">
<tr>
<td valign="top" align="left" width="60%"><b><font
```

Time is Money (part 1)

```
color="#3098C3">Project
Descriptions</font></b></td>
</tr>

<tr>
<td>&nbsp;</td>
</tr>

<?
// open connection to database
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

// get project names and descriptions
$query = "SELECT pname, pdesc FROM projects";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// iterate through resultset and display
while (list($pname, $pdesc) = mysql_fetch_row($result))
{
echo "<tr><td><ul><b><li>$pname</b><br>$pdesc</td>";
echo "<tr><td>&nbsp;</td>";
}

// close connection
mysql_close($connection);
?>

</table>

<? include("footer.inc.php"); ?>

</body>
</html>
```

It looks simple, and it is. Here's what the output looks like:

Time is Money (part 1)



[Main Menu](#) > [View Project Descriptions](#)

Project Descriptions

- **XTech.com**
Interface design and development of the XTech corporate site
- **Melonfire.com**
Weekly updates of the Melonfire content catalog
- **VideoMoz (Linux)**
Software development of the VideoMoz animation and graphics library for the Linux platform
- **VideoMoz (Windows)**
Software development of the VideoMoz.dll animation library for Windows 98, 2000 and XP
- **NamelessCorp AddressBook**
Design and development of address book application for the NamelessCorp intranet

Almost identical in function is the script "tasks.php", which displays a list of standard tasks, together with definitions for each. The only difference lies in the query that is used – as you can see:

```
<?
// tasks.php - display task list

// includes

// check for valid user session

// display page header

// open connection to database
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

// get project names and descriptions
$query = "SELECT tname, tdesc FROM tasks";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

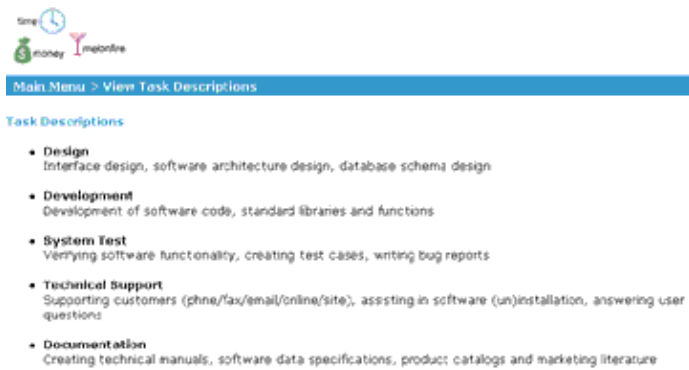
// iterate through resultset and display
while (list($tname, $tdesc) = mysql_fetch_row($result))
{
echo "<tr><td><ul><b><li>$tname</b><br>$tdesc</td>";
echo "<tr><td>&nbsp;</td>";
}

// close connection
mysql_close($connection);

// include page footer
?>
```

Time is Money (part 1)

And here's what this baby looks like:



The screenshot shows a web application interface. At the top left, there are three small icons: a clock labeled 'time', a dollar sign labeled 'money', and a person labeled 'me/online'. Below these icons is a blue navigation bar with the text 'Main Menu > View Task Descriptions'. Underneath the navigation bar, the heading 'Task Descriptions' is followed by a bulleted list of software development tasks.

- **Design**
Interface design, software architecture design, database schema design
- **Development**
Development of software code, standard libraries and functions
- **System Test**
Verifying software functionality, creating test cases, writing bug reports
- **Technical Support**
Supporting customers (phone/fax/email/online/site), assisting in software (un)installation, answering user questions
- **Documentation**
Creating technical manuals, software data specifications, product catalogs and marketing literature

Time For Bed

Of the five options available on the main menu, I've just taken care of two – or, if you want to get technical about it, 40% of the project is now complete. I'll do one more, just to put me on the right side of the halfway mark, and then shut shop for the day.

Since I'm asleep on my feet, I think I'll skip the data entry and report generation code for the moment – this is bound to be complex, and I need all my wits about me to tackle them. Instead, I think I'll handle the last item on the menu next – the "logout.php" script.

```
<?
// logout.php - destroy user session

// destroy session data
session_start();
session_destroy();

// redirect to index page
header("Location:index.html");
?>
```

I don't even think this qualifies to be called a script – it's just three lines of code. All it does is destroy the current session and redirect the browser back to the index page to await a new login.

And that's about it for the moment. We've accomplished a fair amount of work so far – we've got the requirements down, designed a database schema, put together scripts to handle user login and logout, and written all code necessary to display project and task information.

In the concluding part of this article, I will be discussing the scripts which handle timesheet display, record addition and deletion, and report generation. These scripts, especially the ones that handle report generation, will be substantially more complex than the ones you've just seen, so I need to give them some thought before sitting down to write the code. Why don't you do the same, and we'll get together soon to compare notes?

Note: All examples in this article have been tested on Linux/i586 with Apache 1.3.12, MySQL 3.23 and PHP 4.06. Examples are illustrative only, and are not meant for a production environment. YMMV!