SYSTEM MANUAL

Contents

1. Overview of the System

The purpose of this systems manual is to provide information as to the operation and maintenance of the code for future developers of the system, or myself if revisiting the system after a long hiatus and needing to re-familiarise myself with the inner workings of the system.

The system is written in PHP, and uses 5 tables in a MySQL database, to store information. The purpose of the system is to allow users to add content such as videos and images, and allow other users to comment and rate on that content. The system is also designed to make it easy to connect an administration system, by leaving hooks in place for common features such as banning and thread removal, as well as trouble tickets that users can submit for moderator and administrator consideration.

2. DESCRIPTION OF INDIVIDUAL UNITS AND THEIR PURPOSE

2.1. changeuser.php

This takes the adjustments to a user's details from changeusermenu.php, and applies the appropriate sanitization and validation, before updating the database with them.

Procedures of changeuser.php

GenerateEncryptionKey($UID,$Name)

```
function GenerateEncryptionKey($UID,$Name)
{
 $key = "";
 $constant = "ABRAXAS";
 $tuid = $UID;
 $tName = convert_uuencode($Name);
 $i = 0;

 str_pad($key,strlen(tName));
 for ($i=0; $i<=strlen($tName);$i++)
  {

   $key[$i] = chr( (ord($tName[$i])*ord($constant[$i % 7])+ord($UID)-
48+$i) % 256 );
  }
 return implode($key);
}
```

convert_uuencode(): this converts a string into entirely printable characters, ensuring that there will be an ordinal value for tName when creating the key
str_pad(): this pads a string to a new length, in this case the length of the uuencoded username, tName. This is to ensure that the key is long enough to hold the final encryption key.
implode(): This converts an array of characters into a string, thus returning the final encryption key.

The for loop serves to go through every position in the newly-passed $key, returning a character equal to the ASCII value of the ordinal value of the equivalent positioned character in tName multiplied by the ordinal value of our constant’s relative postion in tName, modulo the length of the constant. This value is then added to the ordinal value of the user’s UID, subtracting 48 (which is the number of characters before the ASCII alphabet begins and adding the presently accessed character’s numeric position.

2.2. changeusermenu.php

This menu allows the user to change their personal settings, filling in previous defaults where possible, which is then submitted and passed to changeuser.php

2.3. checklogin.php

This receives the form data from loginmenu.php, and compares it with the user database to return the information of the user that just logged in (starting the session) or returning a login failed message if the login details were incorrect. The password from loginmenu.php is AES encrypted (using an encryption key based on the user's UID and their Username, detailed in procedure GenerateEncryptionKey($UID,$Name)) and compared with the already AES encrypted binary blob stored in the user's password field.

Procedures of checklogin.php

GenerateEncryptionKey($UID,$Name)

```
function GenerateEncryptionKey($UID,$Name)
{
 $key = "";
 $constant = "ABRAXAS";
 $tuid = $UID;
 $tName = convert_uuencode($Name);
 $i = 0;

 str_pad($key,strlen(tName));
 for ($i=0; $i<=strlen($tName);$i++)
  {

   $key[$i] = chr( (ord($tName[$i])*ord($constant[$i % 7])+ord($UID)-
48+$i) % 256 );
  }
 return implode($key);
}
```

convert_uuencode(): this converts a string into entirely printable characters, ensuring that there will be an ordinal value for tName when creating the key
str_pad(): this pads a string to a new length, in this case the length of the uuencoded username, tName. This is to ensure that the key is long enough to hold the final encryption key.
implode(): This converts an array of characters into a string, thus returning the final encryption key.

The for loop serves to go through every position in the newly-passed $key, returning a character equal to the ASCII value of the ordinal value of the equivalent positioned character in tName multiplied by the ordinal value of our constant's relative postion in tName, modulo the length of the constant. This value is then added to the ordinal value of the user's UID, subtracting 48 (which is the number of characters before the ASCII alphabet begins and adding the presently accessed character's numeric position.

2.4. checkregister.php

This receives the form data from registermenu.php, and creates a new user account based on the data supplied, encrypting the given password using AES encryption (using an encryption key based on the user's UID and their Username, detailed in function GenerateEncryptionKey($UID,$Name)), and storing it as a binary blob.

Procedures of checkregister.php

GenerateEncryptionKey($UID,$Name)

```
function GenerateEncryptionKey($UID,$Name)
{
 $key = "";
 $constant = "ABRAXAS";
 $tuid = $UID;
 $tName = convert_uuencode($Name);
 $i = 0;

 str_pad($key,strlen(tName));
 for ($i=0; $i<=strlen($tName);$i++)
  {

   $key[$i] = chr( (ord($tName[$i])*ord($constant[$i % 7])+ord($UID)-
48+$i) % 256 );
  }
 return implode($key);
}
```

convert_uuencode(): this converts a string into entirely printable characters, ensuring that there will be an ordinal value for tName when creating the key
str_pad(): this pads a string to a new length, in this case the length of the uuencoded username, tName. This is to ensure that the key is long enough to hold the final encryption key.
implode(): This converts an array of characters into a string, thus returning the final encryption key.

The for loop serves to go through every position in the newly-passed $key, returning a character equal to the ASCII value of the ordinal value of the equivalent positioned character in tName multiplied by the ordinal value of our constant's relative postion in tName, modulo the length of the constant. This value is then added to the ordinal value of the user's UID, subtracting 48 (which is the number of characters before the ASCII alphabet begins and adding the presently accessed character's numeric position.

2.5. conf.cfg

conf.cfg is a configuration file used for setting up rules to judge whether a file may be uploaded to the web server or not. This is parsed by configreader.php and sends the data to whichever unit included the file.

2.6. configreader.php

This reads the file config.cfg, used for validating uploaded files. This is done via three passes over the config file.

Pass 1: This finds the location of the beginning and end of the config, as well as the beginning and end of the list of available upload locations, and the list of truncated file types.

Pass 2: This reads the lines that are between the beginning and end of the config file, but not between the the beginning and end of the available upload locations or truncated file type list, and stores the contents as parameters.

Parameters stored consist of allowdropdown (defaults to TRUE) which indicates whether or not to show the multiple uploadable folders menu, (img/vid)maxsize (defaults to 2048 kilobytes) which indicates the cap of the uploaded file's size, in kilobytes, (img/vid)truncsize, which indicates whether or not to use the file size cap, and trunctype, which indicates whether or not to use the file type truncation.

Pass 3: This reads the contents of the area between the available upload locations and the truncated file type list, and stores them in separate arrays.

Procedures of configreader.php

stripcomments($outputstring)

```
function stripcomments($outputstring)
 {
 $strcount=0;
 while($strcount<strlen($outputstring))
  {
   if($outputstring[$strcount]=="#")
   {
    return substr($outputstring,0,$strcount);
   }
   $strcount++;
  }
  return $outputstring;
 }
```

strlen(): Returns the length of a given string, in this case used for the while statement that goes through the string looking for the # character, used for commenting lines out.
substr(): Returns a string based on another string and a cutoff point, in this context used to remove the contents after a comment is found.

stripcomments($outputstring) is used to take the present line and strip any commented out information before the line is parsed.

2.7. createthreads.php

This receives the information needed to create a thread from createthreadsmenu.php, and after sanitizing all inputs, examines to see if a file was uploaded along with a comment. If an image was uploaded, then it is moved to the appropriate location and the file path and extension noted, and if a video was uploaded, it converts it to FLV format, so that it can be embedded in pages, moving that converted file to the appropriate location and noting the path and flv extension, finally adding the thread to the database.

2.8. createthreadsmenu.php

This is the input form for the creation of threads. It takes a topic title, a text comment, and either an image or a video file, and submits it to be added to the site.

2.9. dbconnect.php

This manages all database input and output, including the reading of the database configuration file. It is required in every file that uses the database, and allows for quick database connection as well as containing the procedures that create the tables or database originally.

Procedures of dbconnect.php

DBInfo()

```
function DBInfo()
  {
    $config=fopen("servdetails.cfg","r");
    $linecount=0;
    while (!feof($config))
    {
      $outputstring = fgetss($config);
      $linecount++;
      $arg = explode("=",$outputstring);
      $arg[0] = trim(rtrim($arg[0]));
      $arg[1] = trim(rtrim($arg[1]));
      switch($arg[0])
      {
        case "servername":
          $dbservername = $arg[1];
          break;
        case "dbusername":
          $dbusername = $arg[1];
          break;
        case "dbpassword":
          if(isset($arg[1]))
            {
              $dbpassword = $arg[1];
            }
          else $dbpassword = "";
          break;
        case "tablename":
          $tablename = $arg[1];
          break;
      }
    }
    $DBInfo = array();
    $DBInfo[0] = $dbservername;
    $DBInfo[1] = $dbusername;
    $DBInfo[2] = $dbpassword;
    $DBInfo[3] = $tablename;

    return $DBInfo;
```

```
  }
```

DBInfo returns all the needed information to connect to the database, as outlined and documented at setup time, via initialsetup.php and setupmenu.php.

Procedures of dbconnect.php

SQLCreate($dbname)

```
function SQLCreate($dbname)
  {
  $DBInfo = DBInfo();
  $dbservername = $DBInfo[0];
  $dbusername = $DBInfo[1];
  $dbpassword = $DBInfo[2];
  $tablename = $DBInfo[3];
  $con = mysql_connect("$dbservername","$dbusername","$dbpassword");
  if (!$con)
    {
    die('Could not connect: ' . mysql_error());
    }
  if (mysql_query("CREATE DATABASE ".$dbname,$con))
    {
    echo "Database created";
    }
  else
    {
    echo "Error creating database: " . mysql_error();
    }
  mysql_close($con);
  }
```

SQLCreate creates a database, this is used near the beginning of setup, if the database is not already existing.

DBConnect($dbname)

```
function DBConnect($dbname)
  {
  $con = mysql_connect("$dbservername","$dbusername","$dbpassword");
  if (!$con)
    {
    die('Could not connect: ' . mysql_error());
    }
  mysql_select_db("$dbname", $con);
  }
```

DBConnect is a simple and quick way to connect to another database on the server.

SQLConnect()

```
function SQLConnect()
  {
  $con = mysql_connect("$dbservername","$dbusername","$dbpassword");
  if (!$con)
    {
    die('Could not connect: ' . mysql_error());
    }
  }
```

SQLConnect() does not connect to another database, only connecting to the server, and as such can be more useful than DBConnect in some places.

2.10. defstyle.css

This is my default css file for styling parts of the site, although it is a shame I was unable to do this to all areas due to time constraints.

2.11. examinethreads.php

This reads through the threads table, pulling 10 threads that are not replies, but rather the “heads” of threads. If there are more than 10 threads, then the “next page” hyperlink becomes available, allowing the user to view the next 10 older threads. Clicking on the thread's topic title links you to the topic, and clicking on the author links you to his profile.

2.12. ffmpeg.exe

FFMpeg is my chosen software solution for converting video formats to FLV, due to its minimalistic, console-driven approach. FFmpeg is licensed under the GNU Lesser General Public License (LGPL) version 2.1 or later. However, FFmpeg incorporates several optional parts and optimizations that are covered by the GNU General Public License (GPL) version 2 or later. If those parts get used the GPL applies to all of FFmpeg. It can be found at http://www.ffmpeg.org/

2.13. index.php

This is simply a plain page with the navbar to the rest of the content.

2.14. initialsetup.php

This handles all the installation of the system, creating (if needed) and populating the database, as well as creating the administrative account, which is the first user of the system. While there are no procedures defined within it, there is an impressive amount of SQL which is worth looking over. It also writes the server access data to a plain text file, as without it the server is inaccessible.

2.15. loginmenu.php

This asks for the user's username and password, passing the information to checklogin.php for validation.

2.16. permissions.php

This handles most of the permission-based issues in the system, as far as can be handled without the external administration module as provided by the end-user.

Procedures of permissions.php

ResolvePermissions($PermissionLevel)

```
function ResolvePermissions($PermissionLevel) //Return the permissions
configuration of a user.
{
$PermissionLevel = (integer)$PermissionLevel;
$permsql = "SELECT
Name,CanBan,CanDelete,CanReadComplaints,CanRead,CanWrite FROM permissions
WHERE UID = '$PermissionLevel'";
 if (!mysql_query($permsql))
  {
   die('Error:'.mysql_error());
  }
 $permquery = mysql_query($permsql);
```

```
 $PermissionValues = mysql_fetch_array($permquery);
 return $PermissionValues;
}
```

ResolvePermissions() Takes a UID of a permission level and returns the associated permissions configuration.

CanRead($PermissionLevel)

```
function CanRead($PermissionLevel)//Returns a boolean on if the user has read access.
{
 $Perms = ResolvePermissions($PermissionLevel);
 if($Perms["CanRead"] = 1)
 {
  return TRUE;
 }
 else
 {
  return FALSE;
 }
}
```

This returns a boolean value in regards to whether or not the user has read access, used primarily for if statements.

CanWrite($PermissionLevel)

```
function CanWrite($PermissionLevel)//Returns a boolean on if the user has write access.
{
 $Perms = ResolvePermissions($PermissionLevel);
 if($Perms["CanWrite"] = 1)
 {
  return TRUE;
 }
 else
 {
  return FALSE;
 }
}
```

Similar to the above, this returns a boolean value in regards to whether or not the user has write access, used primarily for if statements.

Get_User_PermissionLevel()

```
function Get_User_PermissionLevel()//Uses session data to return a
permission level.
{
 $sql_userlevel = "SELECT PermissionLevel FROM user WHERE UID =
$_SESSION[LoggedUID]";
 if (!mysql_query($sql_userlevel))
  {
   die('Error-userlevel: ' . mysql_error());
  }
 $userlevel = mysql_fetch_row(mysql_query($sql_userlevel));
 return $userlevel[0];
}
```

As the comment says, this uses the logged in user's session data, returning his permission level.

2.17. player.swf

This is a flash object used for playing FLV files, it can be found at http://www.longtailvideo.com/ and is free.

2.18. purenav.php

This is the navbar at the top of most pages, it is never navigated to manually, but is included in each php page, just after sessions are started.

2.19. rating.php

This file adds or subtracts from a thread's rating as well as simultaneously doing the same to a user's rating, and then updates the database with the new data.

2.20. registermenu.php

This page allows a user to create a site account, requesting username, password and email address. This information is then passed to checkregister.php for processing before the user is added to the user database.

2.21. servdetails.cfg

This configuration file contains the information needed to access the server.

2.22. setupmenu.php

This is the installation menu, requesting server and initial user registration information, to be passed onto initialsetup.php

2.23. swfobject.js

This is embedded flash player code, used to put player.swf into pages. It is not my own and is released under the MIT license (http://www.opensource.org/licenses/mit-license.php)

2.24. viewtopic.php

Viewtopic views a given topic, as well as any direct replies to that topic, and any replies to those topics, and so on and so forth, **recursively** until all topics in relation to the head topic are replied to.

Procedures of viewtopic.php

Replies($InitialThreadUID,$RecursionLevel)

```
function Replies($InitialThreadUID,$RecursionLevel)
{
 //grab all threads that match ThreadID to  InitialThreadUID
 //for every thread in that list, run this procedure with their UID as
the argument.

 $i = 0;
 $sql = "SELECT UID,Name,AuthorID,Type,ContentName,Comment,NetRating
FROM thread WHERE ThreadID = '$InitialThreadUID'";
 if (!mysql_query($sql))
  {
   die('Error:'.mysql_error());
  }
 $query = mysql_query($sql);
```

```
 $q = mysql_affected_rows();
 for($i = 0; $i < $q; $i++) //This is the start of the dumping loop.
  {
   $subthreads = mysql_fetch_array($query);
   $sql2 = "SELECT Name,Avatar,NetRating,PermissionLevel FROM user WHERE
UID = $subthreads[AuthorID]";
   if (!mysql_query($sql2))
    {
     die('Error2: ' . mysql_error());
    }
   $name = mysql_fetch_row(mysql_query($sql2));
   echo "<table>";
   echo "<tr class='titlesbar'>";
   echo "<td class='titlebar' colspan = '2'>$subthreads[Name]</td>";
   echo  "<td  class='ratingbar'>Rating: $subthreads[NetRating]<a href =
'rating.php?rate=up&thread=$subthreads[UID]'>+</a>/<a href =
'rating.php?rate=down&thread=$subthreads[UID]'>-</a></td>";
   echo "</tr>";
   echo "<tr class='contentbar'>";

   echo  "<td class='userbar' ><a class = 'username' href =
'viewuser.php?id=$subthreads[AuthorID]'>$name[0]</a><br/>";
   if (!$name[1] == NULL)
    {
     echo  "<img src = 'avatars/$name[1]'/><br/>";
    }
   $userpermit = ResolvePermissions($name[3]);
   echo  "$userpermit[Name]<br/>";
   echo  "Reputation: $name[2]";

   echo "</td>";
   echo  "<td class='contentsbar' colspan = '2'>";

   if ($subthreads["Type"] == 1)
   {

      echo "<img src = '$subthreads[ContentName]'/><br/>";
   }
   if ($subthreads["Type"] == 2)
   {
      echo"<script type='text/javascript' src='swfobject.js'></script>";

      echo"<div id='mediaspace$subthreads[UID]'>It appears you do not
have Adobe Flash Player, or you have JavaScript disabled.</div>";

      echo"<script type='text/javascript'>";
      echo" var so$subthreads[UID] = new
SWFObject('player.swf','ply','470','320','9','#ffffff');";
      echo" so$subthreads[UID].addParam('allowfullscreen','true');";
      echo" so$subthreads[UID].addParam('allowscriptaccess','always');";
      echo" so$subthreads[UID].addParam('wmode','opaque');";
      echo"
so$subthreads[UID].addVariable('file','$subthreads[ContentName]');";
      echo" so$subthreads[UID].write('mediaspace$subthreads[UID]');";
      echo"</script>";
   }
   echo nl2br($subthreads[Comment])."<br/><br/>";
   if(CanWrite(Get_User_PermissionLevel()))
   {
    echo "<br/><br/><div class = 'replytexts'><a class = 'replytext'
href =
'createthreadsmenu.php?reptarg=$subthreads[UID]'>Reply</a></div>";
   }
   echo "</td></tr>";
   echo  "<td class='repliesbar' colspan = '3'>";
   Replies($subthreads["UID"],($RecursionLevel+1));
   echo "</td></tr>";
   echo "</table>";
```

```
  }
}
```

ResolvePermissions(): Is a procedure from permissions.php, taking a permission UID and returning the properties of that permission state.
Get_User_PermissionLevel(): Procedure written in permissions.php, returning the currently logged in user's permission level from their session data.
CanWrite(): Procedure from permissions.php, returning a boolean regarding the write access privileges of that permission state.

Replies is a recursive function that returns all threads that were replies that link to the head of the thread started with, formatting them to look like the other threads. The reason for the table being used is because of the visual effect of the slight increase in the padding each time due to the table borders, making it clear which reply replies to what.

2.25. viewuser.php

This shows the public data about the user, specifically their Username, Permission Level, Email Address, Location, Avatar, Post Count and Reputation.

3. Description of Complicated Algorithms

3.1. examinethreads.php, the algorithm for splitting database output over multiple pages.(Pagination)

```
if(CanRead(Get_User_PermissionLevel()))
{
  $start = 0;
  if(isset($_GET['start']))
  {
   $start = $_GET['start'];
  }
  $sql = "SELECT UID,Name,AuthorID,NetRating FROM thread WHERE ThreadID =
-1  ORDER BY UID DESC";
  if (!mysql_query($sql))
    {
    die('Error1: ' . mysql_error());
    }
   $result = mysql_query($sql);
  for ($i = 1; $i<=10+($start); $i++)
   {
    $uidarray = mysql_fetch_row($result);
    if ($i > $start)
     {
      $threads[$i-$start] = $uidarray;
     }
   }
   echo "<ul class = 'threadlist'>";
   for($i = 1; $i <= 10; $i++)
   {
   $UID = $threads[$i][2];
   if (!isset($UID))
    {
     break;
    }
   $sql = "SELECT Name FROM user WHERE UID = $UID";
   if (!mysql_query($sql))
    {
    die('Error1: ' . mysql_error());
    }
   $result = mysql_query($sql);
   $name = mysql_fetch_row($result);
   echo "<li class = 'threadlist'>";
   echo "<span class = 'topic'><a href =
'viewtopic.php?id=".$threads[$i][0]."'>".$threads[$i][1]."</a></span>";
   echo "<span class = 'author'><a href =
'viewuser.php?id=$UID'>$name[0]</a></span>";
   echo "<span class = 'rating'>Thread Rating:
".$threads[$i][3]."</span>";
   echo "</li>";
   }
  echo "</ul>";
  if ($start > 0)
   {
    echo "<a href = 'examinethreads.php?start=".($start-10)."'>Previous
Page</a>";
   }
  echo "<br/>";
  if (isset($UID))
    {
     echo "<a href = 'examinethreads.php?start=".($start+10)."'>Next
```

```
Page</a>";
    }
```
Basically, the algorithm fetches “thread head” rows until it reaches the amount specified by the variable $start, after which it stores the next 10 in an array $threads. After $threads has ten threads stored, it ouputs them all in order from newest to oldest, and if there are still threads to output, shows the “Next Page” hyperlink, and if $start is greater than zero, shows the “Previous Page” hyperlink.

3.2. viewtopic.php - Recursive Replies function

```
function Replies($InitialThreadUID,$RecursionLevel)
{
 //grab all threads that match ThreadID to  InitialThreadUID
 //for every thread in that list, run this procedure with their UID as the
argument.
 $i = 0;
 $sql = "SELECT UID,Name,AuthorID,Type,ContentName,Comment,NetRating FROM
thread WHERE ThreadID = '$InitialThreadUID'";
 if (!mysql_query($sql))
  {
   die('Error:'.mysql_error());
  }
 $query = mysql_query($sql);
 $q = mysql_affected_rows();
 for($i = 0; $i < $q; $i++) //This is the start of the dumping loop.
  {
   $subthreads = mysql_fetch_array($query);
   $sql2 = "SELECT Name,Avatar,NetRating,PermissionLevel FROM user WHERE
UID = $subthreads[AuthorID]";
   if (!mysql_query($sql2))
    {
     die('Error2: ' . mysql_error());
    }
   $name = mysql_fetch_row(mysql_query($sql2));
   echo "<table>";
   echo "<tr class='titlesbar'>";
   echo "<td class='titlebar' colspan = '2'>$subthreads[Name]</td>";
   echo  "<td  class='ratingbar'>Rating: $subthreads[NetRating]<a href =
'rating.php?rate=up&thread=$subthreads[UID]'>+</a>/<a href =
'rating.php?rate=down&thread=$subthreads[UID]'>-</a></td>";
   echo "</tr>";
   echo "<tr class='contentbar'>";
   echo  "<td class='userbar' ><a class = 'username' href =
'viewuser.php?id=$subthreads[AuthorID]'>$name[0]</a><br/>";
   if (!$name[1] == NULL)
```

```
    {
     echo  "<img src = 'avatars/$name[1]'/><br/>";
    }
   $userpermit = ResolvePermissions($name[3]);
   echo  "$userpermit[Name]<br/>";
   echo  "Reputation: $name[2]";
   echo "</td>";
   echo  "<td class='contentsbar' colspan = '2'>";
   if ($subthreads["Type"] == 1)
   {
      echo "<img src = '$subthreads[ContentName]'/><br/>";
   }
   if ($subthreads["Type"] == 2)
   {
      echo"<script type='text/javascript' src='swfobject.js'></script>";
      echo"<div id='mediaspace$subthreads[UID]'>It appears you do not have
Adobe Flash Player, or you have JavaScript disabled.</div>";
      echo"<script type='text/javascript'>";
      echo" var so$subthreads[UID] = new
SWFObject('player.swf','ply','470','320','9','#ffffff');";
      echo" so$subthreads[UID].addParam('allowfullscreen','true');";
      echo" so$subthreads[UID].addParam('allowscriptaccess','always');";
      echo" so$subthreads[UID].addParam('wmode','opaque');";
      echo"
so$subthreads[UID].addVariable('file','$subthreads[ContentName]');";
      echo" so$subthreads[UID].write('mediaspace$subthreads[UID]');";
      echo"</script>";
   }
   echo nl2br($subthreads['Comment'])."<br/><br/>";
   if(CanWrite(Get_User_PermissionLevel()))
   {
    echo "<br/><br/><div class = 'replytexts'><a class = 'replytext' href
= 'createthreadsmenu.php?reptarg=$subthreads[UID]'>Reply</a></div>";
   }
   echo "</td></tr>";
   echo  "<td class='repliesbar' colspan = '3'>";
   Replies($subthreads["UID"],($RecursionLevel+1));
   echo "</td></tr>";
   echo "</table>";
  }
}
```

This algorithm takes the first thread that it is called with, and searches for all threads that replied to it. It then outputs the first entry in that list, and then calls itself with the UID of that thread as the argument, incrementing the $RecursionLevel variable. Once all replies are found and have had this procedure called against it, it drops down to the previous level of recursion, and so on and so forth until all replies are found, at which point the function ends. The usage of tables makes the degrees of recursion quite obvious as the borders stack against each other.

4. Settings for the system

4.1. Development of the system

The system was developed using XAMMP, a portable program collection including Apache Web Server, PHP 5, and MySQL, as well as several other services I did not need. The default settings for Apache were fine, although the settings for PHP were adjusted slightly, as the default limit on PHP execution time meant that sometimes FFMpeg did not have sufficient time to convert videos when it otherwise should have been able. I increased the time to five minutes, which was plenty of time for videos to convert, and recommend this for any other servers this will be installed upon.

4.2. Distribution of the System

The system will be put onto the college's internal web server, using IIS (Internet Information Services) and PHP 5 with MySQL, as well as eNom Inc's servers, which I have previously discussed the specifications of. The system should be fairly portable, although in future development the server password should be stored encrypted or in a server-side PHP file so that clients cannot stumble across it. The installation procedure is simple and quick, and as such should work fine across a wide range of systems, so long as they have PHP and MySQL, and meet the specifications mentioned in the design.