```
<?php
/*
 * Xibo - Digital Signage - http://www.xibo.org.uk
 * Copyright (C) 2006-2014 Daniel Garner
 *
 * This file is part of Xibo.
 *
 * Xibo is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Affero General Public License as
published by
 * the Free Software Foundation, either version 3 of the License, or
 * any later version.
 *
 * Xibo is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public
License
 * along with Xibo.  If not, see <http://www.gnu.org/licenses/>.
 */
 defined('XIBO') or die("Sorry, you are not allowed to directly
access this page.<br /> Please press the back button in your
browser.");

class User {
    private $db;

    public $userid;
    public $usertypeid;
    public $userName;
    public $homePage;

    private $_myGroups = null;

    /** @var array Cached User Name lookup */
    private $_cacheUsername = array();

    public function __construct(database $db = NULL)
    {
        $this->db           =& $db;
        $this->userid   = Kit::GetParam('userid', _SESSION, _INT);
        $this->usertypeid   = Kit::GetParam('usertype', _SESSION,
_INT);

        // We havent authed yet
        $this->authedDisplayGroupIDs = false;
    }

    /**
     * Validate the User is Logged In
     * @param $ajax Object[optional] Indicates if this request came
from an AJAX call or otherwise
```

```
     */
    function attempt_login($ajax = false)
    {
        $db =& $this->db;
        $userid = Kit::GetParam('userid', _SESSION, _INT);

        // Referring Page is anything after the ?
        $requestUri = rawurlencode(Kit::GetCurrentPage());

        if (!$this->checkforUserid())
        {
            // Log out the user
            if ($userid != 0)
                $db->query(sprintf("UPDATE user SET loggedin = 0
WHERE userid = %d ", $userid));

            // AJAX calls that fail the login test cause a page
redirect
            if ($ajax)
            {
                //create the AJAX request object
                $response = new ResponseManager();

                $response->Login();
                $response->Respond();
            }
            else
            {
                Theme::Set('form_meta', '<input type="hidden"
name="token" value="' . CreateFormToken() . '" />');
                Theme::Set('form_action', 'index.php?
q=login&referingPage=' . $requestUri);
                Theme::Set('about_url', 'index.php?
p=index&q=About');
                Theme::Set('source_url', Theme::SourceLink());

                // Message (either from the URL or the session)
                $message = Kit::GetParam('message', _GET, _STRING,
Kit::GetParam('message', _SESSION, _STRING, ''));
                Theme::Set('login_message', $message);
                Theme::Render('login_page');

                // Clear the session message
                $_SESSION['message'] = '';
                exit;
            }

            return false;
        }
        else
        {
            //write out to the db that the logged in user has
accessed the page still
            $SQL = sprintf("UPDATE user SET lastaccessed = '" .
```

```
date("Y-m-d H:i:s") . "', loggedin = 1 WHERE userid = %d ",
$userid);

            $results = $db->query($SQL) or trigger_error("Can not
write last accessed info.", E_USER_ERROR);

            // Load the information about this user
            $this->LoginServices($userid);

            return true;
        }
    }

    /**
     * Login a user
     * @return
     * @param $username Object
     * @param $password Object
     */
    function login($username, $password)
    {
        $db =& $this->db;

        Kit::ClassLoader('userdata');

        if (Config::Version('DBVersion') < 62) {

            // We can't do CSPRNG because the field doesn't exist,
so we need to do standard user login
            // This can ONLY happen during an upgrade.
            $dbh = PDOConnect::init();
            $sth = $dbh->prepare('SELECT UserID, UserName,
UserPassword, UserTypeID FROM `user` WHERE UserName = :userName');
            $sth->execute(array('userName' => $username));

            $rows = $sth->fetchAll();

            if (count($rows) != 1) {
                setMessage(__('Username or Password incorrect'));
                return false;
            }

            $userInfo = $rows[0];

            // Check the password using a MD5
            if ($userInfo['UserPassword'] != md5($password)) {
                setMessage(__('Username or Password incorrect'));
                return false;
            }

        }
        else {
            // Get the SALT for this username
            if (!$userInfo = $db->GetSingleRow(sprintf("SELECT
```

```
UserID, UserName, UserPassword, UserTypeID, CSPRNG FROM `user` WHERE
UserName = '%s'", $db->escape_string($username)))) {
                setMessage(__('Username or Password incorrect'));
                return false;
            }

            // User Data Object to check the password
            $userData = new Userdata($db);

            // Is SALT empty
            if ($userInfo['CSPRNG'] == 0) {

                // Check the password using a MD5
                if ($userInfo['UserPassword'] != md5($password)) {
                    setMessage(__('Username or Password
incorrect'));
                    return false;
                }

                // Now that we are validated, generate a new SALT
and set the users password.
                $userData-
>ChangePassword(Kit::ValidateParam($userInfo['UserID'], _INT), null,
$password, $password, true /* Force Change */);
            } else {

                // Check the users password using the random SALTED
password
                if ($userData->validate_password($password,
$userInfo['UserPassword']) === false) {
                    setMessage(__('Username or Password
incorrect'));
                    return false;
                }
            }
        }

        // there is a result so we store the userID in the session
variable
        $_SESSION['userid'] =
Kit::ValidateParam($userInfo['UserID'], _INT);
        $_SESSION['username'] =
Kit::ValidateParam($userInfo['UserName'], _USERNAME);
        $_SESSION['usertype'] =
Kit::ValidateParam($userInfo['UserTypeID'], _INT);

        // Set the User Object
        $this->usertypeid = $_SESSION['usertype'];
        $this->userid = $_SESSION['userid'];

        // update the db
        // write out to the db that the logged in user has accessed
the page
        $SQL = sprintf("UPDATE user SET lastaccessed = '" . date("Y-
```

```
m-d H:i:s") . "', loggedin = 1 WHERE userid = %d",
$_SESSION['userid']);

        $db->query($SQL) or trigger_error(__('Can not write last
accessed info.'), E_USER_ERROR);

        // Switch Session ID's
        global $session;
        $session->setIsExpired(0);
        $session->RegenerateSessionID(session_id());

        return true;
    }

        /**
         * Logs in a specific user
         * @param int $userId
         * @return bool
         */
        function LoginServices($userId)
        {
            try {
                $dbh = PDOConnect::init();
                $sth = $dbh->prepare('SELECT UserName, usertypeid,
homepage FROM user WHERE userID = :userId AND Retired = 0');
                $sth->execute(array('userId' => $userId));

                if (!$results = $sth->fetch())
                    return false;

                $this->userid = $userId;
                $this->userName =
Kit::ValidateParam($results['UserName'], _USERNAME);
                $this->usertypeid =
Kit::ValidateParam($results['usertypeid'], _INT);
                $this->homePage =
Kit::ValidateParam($results['homepage'], _WORD);

                return true;
            }
            catch (Exception $e) {
                Debug::Audit($e->getMessage());
                return false;
            }
        }

    /**
     * Logout the user associated with this user object
     * @return
     */
    function logout()
    {
        $db         =& $this->db;
        global $session;
```

```
        $userid = Kit::GetParam('userid', _SESSION, _INT);

        //write out to the db that the logged in user has accessed
the page still
        $SQL = sprintf("UPDATE user SET loggedin = 0 WHERE userid =
%d", $userid);
        if(!$results = $db->query($SQL)) trigger_error("Can not
write last accessed info.", E_USER_ERROR);

        //to log out a user we need only to clear out some session
vars
        unset($_SESSION['userid']);
        unset($_SESSION['username']);
        unset($_SESSION['password']);

        $session->setIsExpired(1);

        return true;
    }

    //Check to see if a user id is in the session information
    function checkforUserid()
    {
        $db         =& $this->db;
        global $session;

        $userid = Kit::GetParam('userid', _SESSION, _INT, 0);

        // Checks for a user ID in the session variable
        if($userid == 0)
        {
            return false;
        }
        else
        {
            if(!is_numeric($_SESSION['userid']))
            {
                unset($_SESSION['userid']);
                return false;
            }
            elseif ($session->isExpired == 1)
            {
                unset($_SESSION['userid']);
                return false;
            }
            else
            {
                // check to see that the ID is still valid
                $SQL = sprintf("SELECT UserID FROM user WHERE
loggedin = 1 AND userid = %d", $userid);

                $result = $db->query($SQL) or trigger_error($db-
>error(), E_USER_ERROR);
```

```
                if($db->num_rows($result)==0)
                {
                    unset($_SESSION['userid']);
                    return false;
                }
                return true;
            }
        }
    }

    /**
     * Get user name from User Id
     * @param int $id
     * @return string
     */
    public function getNameFromID($id)
    {
        $db =& $this->db;

        if (array_key_exists($id, $this->_cacheUsername))
            return $this->_cacheUsername[$id];

        $SQL = sprintf("SELECT username FROM `user` WHERE userid =
%d", $id);

        if(!$results = $db->query($SQL))
            trigger_error("Unknown user id in the system",
E_USER_NOTICE);

        // if no user is returned
        if ($db->num_rows($results) == 0)
        {
            // assume that is the xibo_admin
            return "None";
        }

        $row = $db->get_row($results);

        $this->_cacheUsername[$id] = $row[0];

        return $row[0];
    }

    /**
     * Get My Groups
     * @return array[int]
     */
    public function myGroups()
    {
        if ($this->_myGroups === null) {
            $this->_myGroups = $this->GetUserGroups($this->userid,
true);
        }
```

```
        return $this->_myGroups;
    }

        /**
         * Get an array of user groups for the given user id
         * @param int $id User ID
         * @param bool $returnID Whether to return ID's or Names
         * @return <array>
         */
        public function GetUserGroups($id, $returnID = false)
        {
            $db =& $this->db;

            $groupIDs = array();
            $groups = array();

            $SQL  = "";
            $SQL .= "SELECT group.group, ";
            $SQL .= "       group.groupID ";
            $SQL .= "FROM   `user` ";
            $SQL .= "       INNER JOIN lkusergroup ";
            $SQL .= "       ON     lkusergroup.UserID = user.UserID
";
            $SQL .= "       INNER JOIN `group` ";
            $SQL .= "       ON     group.groupID       =
lkusergroup.GroupID ";
            $SQL .= sprintf("WHERE  `user`.userid
= %d ", $id);

            if(!$results = $db->query($SQL))
            {
                trigger_error($db->error());
                trigger_error("Error looking up user information
(group)", E_USER_ERROR);
            }

            if ($db->num_rows($results) == 0)
            {
                // Every user should have a group?
                // Add one in!
                Kit::ClassLoader('usergroup');

                $userGroupObject = new UserGroup($db);
                if (!$groupID = $userGroupObject->Add($this-
>getNameFromID($id), 1))
                {
                    // Error
                    trigger_error(__('User does not have a group and
Xibo is unable to add one.'), E_USER_ERROR);
                }

                // Link the two
                $userGroupObject->Link($groupID, $id);
```

```
                if ($returnID)
                    return array($groupID);

                return array('Unknown');
            }

            // Build an array of the groups to return
            while($row = $db->get_assoc_row($results))
            {
                $groupIDs[] = Kit::ValidateParam($row['groupID'],
_INT);
                $groups[] = Kit::ValidateParam($row['group'],
_STRING);
            }

            if ($returnID)
                return $groupIDs;


            return $groups;
    }

    function getGroupFromID($id, $returnID = false)
    {
            $db =& $this->db;

            $SQL  = "";
            $SQL .= "SELECT group.group, ";
            $SQL .= "       group.groupID ";
            $SQL .= "FROM   `user` ";
            $SQL .= "       INNER JOIN lkusergroup ";
            $SQL .= "       ON     lkusergroup.UserID = user.UserID
";
            $SQL .= "       INNER JOIN `group` ";
            $SQL .= "       ON     group.groupID       =
lkusergroup.GroupID ";
            $SQL .= sprintf("WHERE  `user`.userid
= %d ", $id);
            $SQL .= "AND    `group`.IsUserSpecific = 1";

            if(!$results = $db->query($SQL))
            {
                trigger_error($db->error());
                trigger_error("Error looking up user information
(group)", E_USER_ERROR);
            }

            if ($db->num_rows($results) == 0)
            {
                // Every user should have a group?
                // Add one in!
                Kit::ClassLoader('usergroup');
```

```
                $userGroupObject = new UserGroup($db);
                if (!$groupID = $userGroupObject->Add($this->getNameFromID($id), 1))
                {
                    // Error
                    trigger_error(__('User does not have a group and we are unable to add one.'), E_USER_ERROR);
                }

                // Link the two
                $userGroupObject->Link($groupID, $id);

                if ($returnID)
                    return $groupID;

                return 'Unknown';
            }

            $row = $db->get_row($results);

            if ($returnID)
            {
                return $row[1];
            }
            return $row[0];
    }

    function getUserTypeFromID($id, $returnID = false)
    {
        $db         =& $this->db;

        $SQL = sprintf("SELECT usertype.usertype, usertype.usertypeid FROM user INNER JOIN usertype ON usertype.usertypeid = user.usertypeid WHERE userid = %d", $id);

        if(!$results = $db->query($SQL))
        {
            trigger_error("Error looking up user information (usertype)");
            trigger_error($db->error());
        }

        if ($db->num_rows($results)==0)
        {
            if ($returnID)
            {
                return "3";
            }
            return "User";
        }

        $row = $db->get_row($results);

        if ($returnID)
```

```
        {
            return $row[1];
        }
        return $row[0];
    }

    function getEmailFromID($id)
    {
        $db         =& $this->db;

        $SQL = sprintf("SELECT email FROM user WHERE userid = %d",
$id);

        if(!$results = $db->query($SQL)) trigger_error("Unknown user
id in the system", E_USER_NOTICE);

        if ($db->num_rows($results)==0)
        {
            $SQL = "SELECT email FROM user WHERE userid = 1";

            if(!$results = $db->query($SQL))
            {
                trigger_error("Unknown user id in the system
[$id]");
            }
        }

        $row = $db->get_row($results);
        return $row[1];
    }

    /**
         * Gets the homepage for the given userid
         * @param <type> $userId
         * @return <type>
         */
    function GetHomePage($userId)
    {
            $db =& $this->db;

            $SQL = sprintf("SELECT homepage FROM `user` WHERE userid
= %d", $userId);

            if (!$homepage = $db->GetSingleValue($SQL, 'homepage',
_WORD))
                trigger_error(__('Unknown User'));

            return $homepage;
    }

    /**
     * Authenticates the page given against the user credentials
held.
     * TODO: Would like to improve performance here by making these
```

```
credentials cached
     * @return
     * @param $page Object
     */
    public function PageAuth($page)
    {
        $db         =& $this->db;
        $userid     =& $this->userid;
        $usertype   =& $this->usertypeid;

        // Check the page exists
        $dbh = PDOConnect::init();

        $sth = $dbh->prepare('SELECT pageID FROM `pages` WHERE name
= :name');
        $sth->execute(array('name' => $page));

        $pageId = $sth->fetchColumn();

        if ($pageId == '') {
            Debug::LogEntry('audit', 'Blocked assess to unrecognised
page: ' . $page . '.', 'index', 'PageAuth');
            throw new Exception(__('Requested page does not
exist'));
        }

        // Check the security
        if ($usertype == 1)
            return true;

        // We have access to only the pages assigned to this group
        try {
            $dbh = PDOConnect::init();

            $SQL  = "SELECT pageid ";
            $SQL .= " FROM `lkpagegroup` ";
            $SQL .= "    INNER JOIN `lkusergroup` ";
            $SQL .= "    ON lkpagegroup.groupID =
lkusergroup.GroupID ";
            $SQL .= " WHERE lkusergroup.UserID = :userid AND pageid
= :pageid";

            $sth = $dbh->prepare($SQL);
            $sth->execute(array(
                    'userid' => $userid,
                    'pageid' => $pageId
                ));

            $results = $sth->fetchAll();

            return (count($results) > 0);
        }
        catch (Exception $e) {
```

```
            Debug::LogEntry('error', $e->getMessage());

            return false;
        }
    }

    /**
     * Return a Menu for this user
     * TODO: Would like to cache this menu array for future requests
     * @return
     * @param $menu Object
     */
    public function MenuAuth($menu)
    {
        $db         =& $this->db;
        $userid     =& $this->userid;
        $usertypeid     =& $this->usertypeid;

        //Debug::LogEntry('audit', sprintf('Authing the menu for
usertypeid [%d]', $usertypeid));

        // Get some information about this menu
        // I.e. get the Menu Items this user has access to
        $SQL  = "";
        $SQL .= "SELECT DISTINCT pages.name     , ";
        $SQL .= "         menuitem.Args , ";
        $SQL .= "         menuitem.Text , ";
        $SQL .= "         menuitem.Class, ";
        $SQL .= "         menuitem.Img, ";
        $SQL .= "         menuitem.External, ";
        $SQL .= "         menuitem.Sequence ";
        $SQL .= "FROM     menuitem ";
        $SQL .= "         INNER JOIN menu ";
        $SQL .= "         ON       menuitem.MenuID = menu.MenuID ";
        $SQL .= "         INNER JOIN pages ";
        $SQL .= "         ON       pages.pageID = menuitem.PageID ";
        if ($usertypeid != 1)
        {
            $SQL .= "       INNER JOIN lkmenuitemgroup ";
            $SQL .= "       ON       lkmenuitemgroup.MenuItemID =
menuitem.MenuItemID ";
            $SQL .= "       INNER JOIN `group` ";
            $SQL .= "       ON       lkmenuitemgroup.GroupID =
group.GroupID ";
                    $SQL .= "       INNER JOIN lkusergroup ";
                    $SQL .= "       ON     group.groupID       =
lkusergroup.GroupID ";
        }
        $SQL .= sprintf("WHERE    menu.Menu             = '%s' ",
$db->escape_string($menu));
        if ($usertypeid != 1)
        {
            $SQL .= sprintf(" AND lkusergroup.UserID = %d",
$userid);
```

```
        }
        $SQL .= " ORDER BY menuitem.Sequence";


        if (!$result = $db->query($SQL))
        {
            trigger_error($db->error());

            return false;
        }

        // No permissions to see any of it
        if ($db->num_rows($result) == 0)
        {
            return false;
        }

        $theMenu = array();

        // Load the results into a menu array
        while ($row = $db->get_assoc_row($result))
        {
            $theMenu[] = $row;
        }

        return $theMenu;
    }

    /**
     * Authenticates this user against the given module
     * or if none provided returns an array of optional modules
     * @return Array
     * @param [Optional] $module String
     */
    public function ModuleAuth($regionSpecific, $module = '',
$assignable = -1)
    {
        $userid =& $this->userid;

        try {
            $dbh = PDOConnect::init();

            // Check that the module is enabled
            $params = array();
            $SQL  = "SELECT * FROM module WHERE Enabled = 1 ";

            if ($regionSpecific != -1) {
                $SQL .= " AND RegionSpecific = :regionspecific ";
                $params['regionspecific'] = $regionSpecific;
            }

            if ($assignable != -1) {
                $SQL .= " AND assignable = :assignable ";
                $params['assignable'] = $assignable;
```

```
            }

            if ($module != '') {
                $SQL .= " AND Module = :module ";
                $params['module'] = $module;
            }

            $SQL .= "  ORDER BY Name ";

            $sth = $dbh->prepare($SQL);
            $sth->execute($params);

            $modules = $sth->fetchAll();

            if (count($modules) == 0) {
                return false;
            }

            // Return this array
            return $modules;
        }
        catch (Exception $e) {

            Debug::LogEntry('error', $e->getMessage());

            return false;
        }
    }

    public function ModuleList($sort_order = array('Name'),
$filter_by = array()) {
        try {
            $dbh = PDOConnect::init();

            $params = array();

            $SQL = '';
            $SQL .= 'SELECT ModuleID, ';
            $SQL .= '   Name, ';
            $SQL .= '   Enabled, ';
            $SQL .= '   Description, ';
            $SQL .= '   RegionSpecific, ';
            $SQL .= '   ValidExtensions, ';
            $SQL .= '   ImageUri, ';
            $SQL .= '   PreviewEnabled, ';
            $SQL .= '   assignable ';
            $SQL .= '  FROM `module` ';
            $SQL .= ' WHERE 1 = 1 ';

            if (Kit::GetParam('id', $filter_by, _INT, 0) != 0) {
                $params['id'] = Kit::GetParam('id', $filter_by,
_INT);
                $SQL .= ' AND ModuleID = :id ';
            }
```

```
            if (Kit::GetParam('name', $filter_by, _STRING) != '') {
                $params['id'] = Kit::GetParam('name', $filter_by,
_STRING);
                $SQL .= ' AND name = :name ';
            }

            // Sorting?
            if (is_array($sort_order))
                $SQL .= 'ORDER BY ' . implode(',', $sort_order);

            //Debug::LogEntry('audit', 'SQL: ' . $SQL . '. Params:
' . var_export($params, true), get_class(), __FUNCTION__);

            $sth = $dbh->prepare($SQL);
            $sth->execute($params);

            return $sth->fetchAll();
        }
        catch (Exception $e) {

            Debug::LogEntry('error', $e->getMessage());

            return false;
        }
    }

    /**
     * Returns the usertypeid for this user object.
     * @return
     */
    public function GetUserTypeID()
    {
        return $this->usertypeid;
    }

    /**
     * Authenticates a user against a fileId
     * @param <type> $fileId
     * @return <bool> true on granted
     */
    public function FileAuth($fileId)
    {
        // Need to check this user has permission to upload this
file (i.e. is it theirs)
        if (!$userId = $this->db->GetSingleValue(sprintf("SELECT
UserID FROM file WHERE FileID = %d", $fileId), 'UserID', _INT))
        {
            trigger_error($this->db->error_text);
            trigger_error($this->db->error());

            return false;
        }
```

```
        return ($userId == $this->userid);
    }

    /**
     * Authorizes a user against a media ID
     * @param <int> $mediaID
     */
    public function MediaAuth($mediaId, $fullObject = false,
$ownerId = 0)
    {
        $auth = new PermissionManager($this);

        if ($ownerId == 0) {
            $SQL  = '';
            $SQL .= 'SELECT UserID ';
            $SQL .= '  FROM media ';
            $SQL .= ' WHERE media.MediaID = %d ';

            if (!$ownerId = $this->db->GetSingleValue(sprintf($SQL,
$mediaId), 'UserID', _INT))
                return $auth;
        }

        // If we are the owner, or a super admin then give full
permissions
        if ($this->usertypeid == 1)
        {
            $auth->FullAccess();
            return $auth;
        }


        // If we are the owner
        if ($ownerId == $this->userid)
        {
            $auth->HalfAccess();
            return $auth;
        }


        // If we are a group admin of the owners groups
        if ($this->usertypeid == 2) {
            // Are we in the same group as the owner.
            $theirGroups = $this->GetUserGroups($ownerId, true);

            if (count(array_intersect($this->myGroups(),
$theirGroups))) {
                $auth->FullAccess();
                return $auth;
            }
        }

        // Permissions for groups the user is assigned to, and
Everyone
```

```
        $SQL  = '';
        $SQL .= 'SELECT UserID, MAX(IFNULL(View, 0)) AS View,
MAX(IFNULL(Edit, 0)) AS Edit, MAX(IFNULL(Del, 0)) AS Del ';
        $SQL .= '  FROM media ';
        $SQL .= '   INNER JOIN lkmediagroup ';
        $SQL .= '   ON lkmediagroup.MediaID = media.MediaID ';
        $SQL .= '   INNER JOIN `group` ';
        $SQL .= '   ON `group`.GroupID = lkmediagroup.GroupID ';
        $SQL .= ' WHERE media.MediaID = %d ';
        $SQL .= '   AND (`group`.IsEveryone = 1 OR `group`.GroupID
IN (%s)) ';
        $SQL .= 'GROUP BY media.UserID ';

        $SQL = sprintf($SQL, $mediaId, implode(',', $this-
>myGroups()));
        //Debug::LogEntry('audit', $SQL);

        if (!$row = $this->db->GetSingleRow($SQL))
            return $auth;

        // There are permissions to evaluate
        $auth->Evaluate($row['UserID'], $row['View'], $row['Edit'],
$row['Del']);

        if ($fullObject)
            return $auth;

        return $auth->edit;
    }

    /**
     * Authorizes a user against a media ID assigned to a layout
     * @param <int> $mediaID
     */
    public function MediaAssignmentAuth($ownerId, $layoutId,
$regionId, $mediaId, $fullObject = false)
    {
        $auth = new PermissionManager($this);

        // If we are the owner, or a super admin then give full
permissions
        if ($this->usertypeid == 1)
        {
            $auth->FullAccess();
            return $auth;
        }


        // If we are the owner
        if ($ownerId == $this->userid)
        {
            $auth->HalfAccess();
            return $auth;
        }
```

```
        // If we are a group admin of the owners groups
        if ($this->usertypeid == 2) {
            // Are we in the same group as the owner.
            $theirGroups = $this->GetUserGroups($ownerId, true);

            if (count(array_intersect($this->myGroups(),
$theirGroups))) {
                $auth->FullAccess();
                return $auth;
            }
        }

        // Permissions for groups the user is assigned to, and
Everyone
        $SQL  = '';
        $SQL .= 'SELECT MAX(IFNULL(View, 0)) AS View,
MAX(IFNULL(Edit, 0)) AS Edit, MAX(IFNULL(Del, 0)) AS Del ';
        $SQL .= '  FROM lklayoutmediagroup ';
        $SQL .= '   INNER JOIN `group` ';
        $SQL .= '   ON `group`.GroupID = lklayoutmediagroup.GroupID
';
        $SQL .= " WHERE lklayoutmediagroup.MediaID = '%s' AND
lklayoutmediagroup.RegionID = '%s' AND lklayoutmediagroup.LayoutID =
%d ";
        $SQL .= '   AND (`group`.IsEveryone = 1 OR `group`.GroupID
IN (%s)) ';

        $SQL = sprintf($SQL, $this->db->escape_string($mediaId),
$this->db->escape_string($regionId), $layoutId, implode(',', $this-
>myGroups()));
        //Debug::LogEntry('audit', $SQL);

        if (!$row = $this->db->GetSingleRow($SQL))
            return $auth;

        // There are permissions to evaluate
        $auth->Evaluate($ownerId, $row['View'], $row['Edit'],
$row['Del']);

        if ($fullObject)
            return $auth;

        return $auth->edit;
    }

    public function RegionAssignmentAuth($ownerId, $layoutId,
$regionId, $fullObject = false)
    {
        $auth = new PermissionManager($this);

        // If we are the owner, or a super admin then give full
permissions
```

```
        if ($this->usertypeid == 1)
        {
            $auth->FullAccess();
            return $auth;
        }


        // If we are the owner
        if ($ownerId == $this->userid)
        {
            $auth->HalfAccess();
            return $auth;
        }


        // If we are a group admin of the owners groups
        if ($this->usertypeid == 2) {
            // Are we in the same group as the owner.
            $theirGroups = $this->GetUserGroups($ownerId, true);

            if (count(array_intersect($this->myGroups(),
$theirGroups))) {
                $auth->FullAccess();
                return $auth;
            }
        }

        // Permissions for groups the user is assigned to, and
Everyone
        $SQL  = '';
        $SQL .= 'SELECT MAX(IFNULL(View, 0)) AS View,
MAX(IFNULL(Edit, 0)) AS Edit, MAX(IFNULL(Del, 0)) AS Del ';
        $SQL .= '  FROM lklayoutregiongroup ';
        $SQL .= '   INNER JOIN `group` ';
        $SQL .= '   ON `group`.GroupID = lklayoutregiongroup.GroupID
';
        $SQL .= " WHERE lklayoutregiongroup.RegionID = '%s' AND
lklayoutregiongroup.LayoutID = %d ";
        $SQL .= '   AND (`group`.IsEveryone = 1 OR `group`.GroupID
IN (%s)) ';

        $SQL = sprintf($SQL, $this->db->escape_string($regionId),
$layoutId, implode(',', $this->myGroups()));
        //Debug::LogEntry('audit', $SQL);

        if (!$row = $this->db->GetSingleRow($SQL))
            return $auth;

        // There are permissions to evaluate
        $auth->Evaluate($ownerId, $row['View'], $row['Edit'],
$row['Del']);

        if ($fullObject)
            return $auth;
```

```
        return $auth->edit;
    }

    /**
     * Returns an array of Media the current user has access to
     */
    public function MediaList($sort_order = array('name'),
$filter_by = array())
    {
        try {
            $media = Media::Entries($sort_order, $filter_by);
            $parsedMedia = array();

            foreach ($media as $row) {
                $mediaItem = array();

                // Validate each param and add it to the array.
                $mediaItem['mediaid'] = $row->mediaId;
                $mediaItem['media'] = $row->name;
                $mediaItem['mediatype'] = $row->mediaType;
                $mediaItem['duration'] = $row->duration;
                $mediaItem['ownerid'] = $row->ownerId;
                $mediaItem['filesize'] = $row->fileSize;
                $mediaItem['parentid'] = $row->parentId;
                $mediaItem['filename'] = $row->fileName;
                $mediaItem['tags'] = $row->tags;
                $mediaItem['storedas'] = $row->storedAs;
                $mediaItem['groups'] = $row->groups;

                $auth = $this->MediaAuth($row->mediaId, true, $row-
>ownerId);

                if ($auth->view) {
                    $mediaItem['view'] = (int)$auth->view;
                    $mediaItem['edit'] = (int)$auth->edit;
                    $mediaItem['del'] = (int)$auth->del;
                    $mediaItem['modifyPermissions'] = (int)$auth-
>modifyPermissions;

                    $parsedMedia[] = $mediaItem;
                }
            }

            return $parsedMedia;
        }
        catch (Exception $e) {
            Debug::LogEntry('error', $e->getMessage(), get_class(),
__FUNCTION__);
            return false;
        }
    }

    /**
```

```
     * Authorises a user against a layoutid
     * @param <type> $layoutId
     * @return <type>
     */
    public function LayoutAuth($layoutId, $fullObject = false)
    {
        $auth = new PermissionManager($this);

        // Get the Campaign ID
        $SQL  = "SELECT campaign.CampaignID ";
        $SQL .= "  FROM `lkcampaignlayout` ";
        $SQL .= "   INNER JOIN `campaign` ";
        $SQL .= "   ON lkcampaignlayout.CampaignID =
campaign.CampaignID ";
        $SQL .= " WHERE lkcampaignlayout.LayoutID = %d ";
        $SQL .= "   AND campaign.IsLayoutSpecific = 1";

        if (!$campaignId = $this->db->GetSingleValue(sprintf($SQL,
$layoutId), 'CampaignID', _INT))
        {
            trigger_error(__('Layout has no associated Campaign,
corrupted Layout'));

            // New auth object is no permissions
            if ($fullObject)
                return $auth;

            return false;
        }

        $auth = $this->CampaignAuth($campaignId, true);

        if ($fullObject)
            return $auth;

        return $auth->edit;
    }

    /**
     *Authorises a user against a template Id
     * @param <type> $templateId
     * @return <type>
     */
    public function TemplateAuth($templateId, $fullObject = false)
    {
        $auth = $this->LayoutAuth($templateId, true);

        if ($fullObject)
            return $auth;

        return $auth->edit;
    }

    /**
```

```
     * Returns an array of layouts that this user has access to
     */
    public function LayoutList($sort_order = array('layout'),
$filter_by = array()) {

        $layouts = Layout::Entries($sort_order, $filter_by);
        $parsedLayouts = array();

        foreach ($layouts as $row) {
            $layoutItem = array();

            // Validate each param and add it to the array.
            $layoutItem['layoutid'] = $row->layoutId;
            $layoutItem['layout']   = $row->layout;
            $layoutItem['description'] = $row->description;
            $layoutItem['tags'] = $row->tags;
            $layoutItem['ownerid']  = $row->ownerId;
            $layoutItem['xml']  = $row->xml;
            $layoutItem['campaignid'] = $row->campaignId;
            $layoutItem['retired'] = $row->retired;
            $layoutItem['status'] = $row->status;
            $layoutItem['backgroundImageId'] = $row-
>backgroundImageId;
            $layoutItem['mediaownerid'] = $row->mediaOwnerId;

            // Details for media assignment
            $layoutItem['regionid'] = $row->regionId;
            $layoutItem['lklayoutmediaid'] = $row->lkLayoutMediaId;

            $layoutItem['groups'] = $row->groups;

            // Authenticate the assignment (if not null already)
            if ($layoutItem['lklayoutmediaid'] != 0) {
                $assignmentAuth = $this-
>MediaAssignmentAuth($layoutItem['mediaownerid'],
$layoutItem['layoutid'], $layoutItem['regionid'],
Kit::GetParam('mediaId', $filter_by, _INT, 0), true);

                // If we get here and the user does not have assess
to this region assignment, don't add this row
                if (!$assignmentAuth->del)
                    continue;
            }

            $auth = $this->CampaignAuth($layoutItem['campaignid'],
true);

            if ($auth->view) {
                $layoutItem['view'] = (int) $auth->view;
                $layoutItem['edit'] = (int) $auth->edit;
                $layoutItem['del'] = (int) $auth->del;
                $layoutItem['modifyPermissions'] = (int) $auth-
>modifyPermissions;
```

```
                $parsedLayouts[] = $layoutItem;
            }
        }

        return $parsedLayouts;
    }

    /**
     * A List of Templates the User has access to
     * @param string $template [description]
     * @param string $tags     [description]
     * @param string $isSystem [description]
     */
    public function TemplateList($sort_order = array('layout'),
$filter_by = array())
    {
        $filter_by['excludeTemplates'] = 0;

        return $this->LayoutList($sort_order, $filter_by);
    }

    public function ResolutionList($sort_order =
array('resolution'), $filter_by = array()) {
        try {
            $dbh = PDOConnect::init();

            $params = array();
            $SQL = 'SELECT * FROM resolution WHERE 1 = 1 ';

            // Include the current?
            if (Kit::GetParam('withCurrent', $filter_by, _INT, 0) !=
0) {
                $SQL .= ' OR resolutionid = :resolutionid ';
                $params['resolutionid'] =
Kit::GetParam('withCurrent', $filter_by, _INT);
            }

            if (Kit::GetParam('enabled', $filter_by, _INT, 1) != -1)
{
                $SQL .= ' AND enabled = :enabled ';
                $params['enabled'] = Kit::GetParam('enabled',
$filter_by, _INT, 1);
            }

            // Sorting?
            if (is_array($sort_order))
                $SQL .= 'ORDER BY ' . implode(',', $sort_order);

            Debug::sql($SQL, $params);

            $sth = $dbh->prepare($SQL);
            $sth->execute($params);

            $results = $sth->fetchAll();
```

```
            $resolutions = array();

            foreach ($results as $row) {
                $res = array();

                $res['resolutionid'] =
Kit::ValidateParam($row['resolutionID'], _INT);
                $res['resolution'] =
Kit::ValidateParam($row['resolution'], _STRING);
                $res['width'] = Kit::ValidateParam($row['width'],
_INT);
                $res['height'] = Kit::ValidateParam($row['height'],
_INT);
                $res['intended_width'] =
Kit::ValidateParam($row['intended_width'], _INT);
                $res['intended_height'] =
Kit::ValidateParam($row['intended_height'], _INT);
                $res['version'] =
Kit::ValidateParam($row['version'], _INT);
                $res['enabled'] =
Kit::ValidateParam($row['enabled'], _INT);

                $resolutions[] = $res;
            }

            return $resolutions;
        }
        catch (Exception $e) {

            Debug::LogEntry('error', $e->getMessage());

            return false;
        }
    }

    /**
     * Authorises a user against a dataSetId
     * @param <type> $dataSetId
     * @return <type>
     */
    public function DataSetAuth($dataSetId, $fullObject = false)
    {
        $auth = new PermissionManager($this);

        $SQL  = '';
        $SQL .= 'SELECT UserID ';
        $SQL .= '  FROM dataset ';
        $SQL .= ' WHERE dataset.DataSetID = %d ';

        if (!$ownerId = $this->db->GetSingleValue(sprintf($SQL,
$dataSetId), 'UserID', _INT))
            return $auth;

        // If we are the owner, or a super admin then give full
```

```
permissions
        if ($this->usertypeid == 1)
        {
            $auth->FullAccess();
            return $auth;
        }


        // If we are the owner
        if ($ownerId == $this->userid)
        {
            $auth->HalfAccess();
            return $auth;
        }


        // If we are a group admin of the owners groups
        if ($this->usertypeid == 2) {
            // Are we in the same group as the owner.
            $theirGroups = $this->GetUserGroups($ownerId, true);

            if (count(array_intersect($this->myGroups(),
$theirGroups))) {
                $auth->FullAccess();
                return $auth;
            }
        }

        // Permissions for groups the user is assigned to, and
Everyone
        $SQL  = '';
        $SQL .= 'SELECT UserID, MAX(IFNULL(View, 0)) AS View,
MAX(IFNULL(Edit, 0)) AS Edit, MAX(IFNULL(Del, 0)) AS Del ';
        $SQL .= '  FROM dataset ';
        $SQL .= '   INNER JOIN lkdatasetgroup ';
        $SQL .= '   ON lkdatasetgroup.DataSetID = dataset.DataSetID
';
        $SQL .= '   INNER JOIN `group` ';
        $SQL .= '   ON `group`.GroupID = lkdatasetgroup.GroupID ';
        $SQL .= ' WHERE dataset.DataSetID = %d ';
        $SQL .= '   AND (`group`.IsEveryone = 1 OR `group`.GroupID
IN (%s)) ';
        $SQL .= 'GROUP BY dataset.UserID ';

        $SQL = sprintf($SQL, $dataSetId, implode(',', $this-
>myGroups()));
        //Debug::LogEntry('audit', $SQL);

        if (!$row = $this->db->GetSingleRow($SQL))
            return $auth;

        // There are permissions to evaluate
        $auth->Evaluate($row['UserID'], $row['View'], $row['Edit'],
$row['Del']);
```

```
        if ($fullObject)
            return $auth;

        return $auth->edit;
    }

    /**
     * Returns an array of layouts that this user has access to
     */
    public function DataSetList()
    {
        $SQL  = "";
        $SQL .= "SELECT DataSetID, ";
        $SQL .= "       DataSet, ";
        $SQL .= "       Description, ";
        $SQL .= "       UserID ";
        $SQL .= "  FROM dataset ";
        $SQL .= " ORDER BY DataSet ";

        //Debug::LogEntry('audit', sprintf('Retreiving list of
layouts for %s with SQL: %s', $this->userName, $SQL));

        if (!$result = $this->db->query($SQL))
        {
            trigger_error($this->db->error());
            return false;
        }

        $dataSets = array();

        while ($row = $this->db->get_assoc_row($result))
        {
            $dataSetItem = array();

            // Validate each param and add it to the array.
            $dataSetItem['datasetid'] =
Kit::ValidateParam($row['DataSetID'], _INT);
            $dataSetItem['dataset']   =
Kit::ValidateParam($row['DataSet'], _STRING);
            $dataSetItem['description'] =
Kit::ValidateParam($row['Description'], _STRING);
            $dataSetItem['ownerid']  =
Kit::ValidateParam($row['UserID'], _INT);

            $auth = $this->DataSetAuth($dataSetItem['datasetid'],
true);

            if ($auth->view)
            {
                $dataSetItem['view'] = (int) $auth->view;
                $dataSetItem['edit'] = (int) $auth->edit;
                $dataSetItem['del'] = (int) $auth->del;
                $dataSetItem['modifyPermissions'] = (int) $auth-
```

```
>modifyPermissions;

                $dataSets[] = $dataSetItem;
            }
        }

        return $dataSets;
    }

    /**
     * Authorises a user against a DisplayGroupId
     * @param <int> $displayGroupId
     * @return <type>
     */
    public function DisplayGroupAuth($displayGroupId, $fullObject =
false)
    {
        $auth = new PermissionManager($this);
        $noOwnerId = 0;

        // If we are the owner, or a super admin then give full
permissions
        if ($this->usertypeid == 1)
        {
            $auth->FullAccess();

            if ($fullObject)
                return $auth;

            return true;
        }

        // Permissions for groups the user is assigned to, and
Everyone
        $SQL  = '';
        $SQL .= 'SELECT MAX(IFNULL(View, 0)) AS View,
MAX(IFNULL(Edit, 0)) AS Edit, MAX(IFNULL(Del, 0)) AS Del ';
        $SQL .= '  FROM displaygroup ';
        $SQL .= '   INNER JOIN lkdisplaygroupgroup ';
        $SQL .= '   ON lkdisplaygroupgroup.DisplayGroupID =
displaygroup.DisplayGroupID ';
        $SQL .= '   INNER JOIN `group` ';
        $SQL .= '   ON `group`.GroupID = lkdisplaygroupgroup.GroupID
';
        $SQL .= ' WHERE displaygroup.DisplayGroupID = %d ';
        $SQL .= '   AND (`group`.IsEveryone = 1 OR `group`.GroupID
IN (%s)) ';

        $SQL = sprintf($SQL, $displayGroupId, implode(',', $this-
>myGroups()));
        //Debug::LogEntry('audit', $SQL);

        if (!$row = $this->db->GetSingleRow($SQL))
            return $auth;
```

```
        // There are permissions to evaluate
        $auth->Evaluate($noOwnerId, $row['View'], $row['Edit'],
$row['Del']);

        if ($fullObject)
            return $auth;

        return $auth->edit;
    }

    /**
     * Authenticates the current user and returns an array of
display groups this user is authenticated on
     * @return
     */
    public function DisplayGroupList($isDisplaySpecific = 0, $name =
'')
    {
        $db         =& $this->db;
        $userid     =& $this->userid;

        $SQL  = "SELECT displaygroup.DisplayGroupID,
displaygroup.DisplayGroup, displaygroup.IsDisplaySpecific,
displaygroup.Description ";
        if ($isDisplaySpecific == 1)
            $SQL .= " , lkdisplaydg.DisplayID ";

        $SQL .= "  FROM displaygroup ";

        // If we are only interested in displays, then return the
display
        if ($isDisplaySpecific == 1)
        {
            $SQL .= "   INNER JOIN lkdisplaydg ";
            $SQL .= "   ON lkdisplaydg.DisplayGroupID =
displaygroup.DisplayGroupID ";
        }

        $SQL .= " WHERE 1 = 1 ";

        if ($name != '')
        {
            // convert into a space delimited array
            $names = explode(' ', $name);

            foreach($names as $searchName)
            {
                // Not like, or like?
                if (substr($searchName, 0, 1) == '-')
                    $SQL.= " AND  (displaygroup.DisplayGroup NOT
LIKE '%" . sprintf('%s', ltrim($db->escape_string($searchName),
'-')) . "%') ";
                else
```

```
                    $SQL.= " AND  (displaygroup.DisplayGroup LIKE
'%" . sprintf('%s', $db->escape_string($searchName)) . "%') ";
            }
        }

        if ($isDisplaySpecific != -1)
            $SQL .= sprintf(" AND displaygroup.IsDisplaySpecific =
%d ", $isDisplaySpecific);

        $SQL .= " ORDER BY displaygroup.DisplayGroup ";

        Debug::LogEntry('audit', sprintf('Retreiving list of
displaygroups for %s with SQL: %s', $this->userName, $SQL));

        if (!$result = $this->db->query($SQL))
        {
            trigger_error($this->db->error());
            return false;
        }

        $displayGroups = array();

        while ($row = $this->db->get_assoc_row($result))
        {
            $displayGroupItem = array();

            // Validate each param and add it to the array.
            $displayGroupItem['displaygroupid'] =
Kit::ValidateParam($row['DisplayGroupID'], _INT);
            $displayGroupItem['displaygroup']   =
Kit::ValidateParam($row['DisplayGroup'], _STRING);
            $displayGroupItem['description']   =
Kit::ValidateParam($row['Description'], _STRING);
            $displayGroupItem['isdisplayspecific'] =
Kit::ValidateParam($row['IsDisplaySpecific'], _STRING);
            $displayGroupItem['displayid'] = (($isDisplaySpecific ==
1) ? Kit::ValidateParam($row['DisplayID'], _INT) : 0);

            $auth = $this-
>DisplayGroupAuth($displayGroupItem['displaygroupid'], true);

            if ($auth->view)
            {
                $displayGroupItem['view'] = (int) $auth->view;
                $displayGroupItem['edit'] = (int) $auth->edit;
                $displayGroupItem['del'] = (int) $auth->del;
                $displayGroupItem['modifypermissions'] = (int)
$auth->modifyPermissions;

                $displayGroups[] = $displayGroupItem;
            }
        }

        return $displayGroups;
```

```
    }

    /**
     * List of Displays this user has access to view
     */
    public function DisplayList($sort_order = array('displayid'),
$filter_by = array(), $auth_level = 'view') {

        $SQL  = 'SELECT display.displayid, ';
        $SQL .= '    display.display, ';
        $SQL .= '    displaygroup.description, ';
        $SQL .= '    layout.layout, ';
        $SQL .= '    display.loggedin, ';
        $SQL .= '    IFNULL(display.lastaccessed, 0) AS
lastaccessed, ';
        $SQL .= '    display.inc_schedule, ';
        $SQL .= '    display.licensed, ';
        $SQL .= '    display.email_alert, ';
        $SQL .= '    displaygroup.DisplayGroupID, ';
        $SQL .= '    display.ClientAddress, ';
        $SQL .= '    display.MediaInventoryStatus, ';
        $SQL .= '    display.MacAddress, ';
        $SQL .= '    display.client_type, ';
        $SQL .= '    display.client_version, ';
        $SQL .= '    display.client_code, ';
        $SQL .= '    display.screenShotRequested, ';
        $SQL .= '    display.storageAvailableSpace, ';
        $SQL .= '    display.storageTotalSpace, ';
        $SQL .= '    currentLayout.layout AS currentLayout, ';
        $SQL .= '    currentLayout.layoutId AS currentLayoutId ';
        $SQL .= '  FROM display ';
        $SQL .= '    INNER JOIN lkdisplaydg ON lkdisplaydg.DisplayID
= display.DisplayID ';
        $SQL .= '    INNER JOIN displaygroup ON
displaygroup.DisplayGroupID = lkdisplaydg.DisplayGroupID ';
        $SQL .= '    LEFT OUTER JOIN layout ON layout.layoutid =
display.defaultlayoutid ';
        $SQL .= '    LEFT OUTER JOIN layout currentLayout ON
currentLayout.layoutId = display.currentLayoutId';

        if (Kit::GetParam('displaygroupid', $filter_by, _INT) != 0)
{
            // Restrict to a specific display group
            $SQL .= sprintf(' WHERE displaygroup.displaygroupid = %d
', Kit::GetParam('displaygroupid', $filter_by, _INT));
        }
        else {
            // Restrict to display specific groups
            $SQL .= ' WHERE displaygroup.IsDisplaySpecific = 1 ';
        }

        // Filter by Display ID?
        if (Kit::GetParam('displayid', $filter_by, _INT) != 0) {
            $SQL .= sprintf(' AND display.displayid = %d ',
```

```
Kit::GetParam('displayid', $filter_by, _INT));
        }

        // Filter by Display Name?
        if (Kit::GetParam('display', $filter_by, _STRING) != '') {
            // convert into a space delimited array
            $names = explode(' ', Kit::GetParam('display',
$filter_by, _STRING));

            foreach($names as $searchName)
            {
                // Not like, or like?
                if (substr($searchName, 0, 1) == '-')
                    $SQL.= " AND  (display.display NOT LIKE '%" .
sprintf('%s', ltrim($this->db->escape_string($searchName), '-')) .
"%') ";
                else
                    $SQL.= " AND  (display.display LIKE '%" .
sprintf('%s', $this->db->escape_string($searchName)) . "%') ";
            }
        }

        if (Kit::GetParam('macAddress', $filter_by, _STRING) != '')
{
            $SQL .= sprintf(' AND display.macaddress LIKE \'%s\' ',
'%' . $this->db->escape_string(Kit::GetParam('macAddress',
$filter_by, _STRING)) . '%');
        }

        // Exclude a group?
        if (Kit::GetParam('exclude_displaygroupid', $filter_by,
_INT) != 0) {
            $SQL .= " AND display.DisplayID NOT IN ";
            $SQL .= "       (SELECT display.DisplayID ";
            $SQL .= "       FROM    display ";
            $SQL .= "               INNER JOIN lkdisplaydg ";
            $SQL .= "               ON      lkdisplaydg.DisplayID =
display.DisplayID ";
            $SQL .= sprintf("   WHERE  lkdisplaydg.DisplayGroupID
= %d ", Kit::GetParam('exclude_displaygroupid', $filter_by, _INT));
            $SQL .= "       )";
        }

        // Filter by client version
        if (Kit::GetParam('clientVersion', $filter_by, _STRING) !=
'') {
            $clientVersion = '%' . $this->db-
>escape_string(Kit::GetParam('clientVersion', $filter_by,
_STRING)) . '%';
            $SQL .= sprintf(" AND (display.client_version LIKE '%s'
OR display.client_type LIKE '%s') ", $clientVersion,
$clientVersion);
        }
```

```
        // Sorting?
        if (is_array($sort_order))
            $SQL .= 'ORDER BY ' . implode(',', $sort_order);

        Debug::sql($SQL, $filter_by);

        if (!$result = $this->db->query($SQL))
        {
            trigger_error($this->db->error());
            return false;
        }

        $displays = array();

        while ($row = $this->db->get_assoc_row($result))
        {
            $displayItem = array();

            // Validate each param and add it to the array.
            $displayItem['displayid'] =
Kit::ValidateParam($row['displayid'], _INT);
            $displayItem['display'] =
Kit::ValidateParam($row['display'], _STRING);
            $displayItem['description'] =
Kit::ValidateParam($row['description'], _STRING);
            $displayItem['layout'] =
Kit::ValidateParam($row['layout'], _STRING);
            $displayItem['loggedin'] =
Kit::ValidateParam($row['loggedin'], _INT);
            $displayItem['lastaccessed'] =
Kit::ValidateParam($row['lastaccessed'], _STRING);
            $displayItem['inc_schedule'] =
Kit::ValidateParam($row['inc_schedule'], _INT);
            $displayItem['licensed'] =
Kit::ValidateParam($row['licensed'], _INT);
            $displayItem['email_alert'] =
Kit::ValidateParam($row['email_alert'], _INT);
            $displayItem['displaygroupid'] =
Kit::ValidateParam($row['DisplayGroupID'], _INT);
            $displayItem['clientaddress'] =
Kit::ValidateParam($row['ClientAddress'], _STRING);
            $displayItem['mediainventorystatus'] =
Kit::ValidateParam($row['MediaInventoryStatus'], _INT);
            $displayItem['macaddress'] =
Kit::ValidateParam($row['MacAddress'], _STRING);
            $displayItem['client_type'] =
Kit::ValidateParam($row['client_type'], _STRING);
            $displayItem['client_version'] =
Kit::ValidateParam($row['client_version'], _STRING);
            $displayItem['client_code'] =
Kit::ValidateParam($row['client_code'], _STRING);
            $displayItem['screenShotRequested'] =
Kit::ValidateParam($row['screenShotRequested'], _INT);
            $displayItem['storageAvailableSpace'] =
```

```
Kit::ValidateParam($row['storageAvailableSpace'], _INT);
            $displayItem['storageTotalSpace'] =
Kit::ValidateParam($row['storageTotalSpace'], _INT);
            $displayItem['currentLayoutId'] =
Kit::ValidateParam($row['currentLayoutId'], _INT);
            $displayItem['currentLayout'] =
Kit::ValidateParam($row['currentLayout'], _STRING);

            $auth = $this-
>DisplayGroupAuth($displayItem['displaygroupid'], true);

            if ($auth->view)
            {
                // If auth level = edit and we don't have edit, then
leave them off
                if ($auth_level == 'edit' && !$auth->edit)
                    continue;

                $displayItem['view'] = (int) $auth->view;
                $displayItem['edit'] = (int) $auth->edit;
                $displayItem['del'] = (int) $auth->del;
                $displayItem['modifypermissions'] = (int) $auth-
>modifyPermissions;

                $displays[] = $displayItem;
            }
        }

        return $displays;

    }

    /**
     * Authorises a user against a campaign
     * @param <type> $campaignId
     * @return <type>
     */
    public function CampaignAuth($campaignId, $fullObject = false)
    {
        $auth = new PermissionManager($this);

        $SQL  = '';
        $SQL .= 'SELECT UserID ';
        $SQL .= '  FROM `campaign` ';
        $SQL .= ' WHERE campaign.CampaignID = %d ';

        if (!$ownerId = $this->db->GetSingleValue(sprintf($SQL,
$campaignId), 'UserID', _INT))
            return $auth;

        // If we are the owner, or a super admin then give full
permissions
        if ($this->usertypeid == 1)
        {
```

```
            $auth->FullAccess();
            return $auth;
        }


        // If we are the owner
        if ($ownerId == $this->userid)
        {
            $auth->HalfAccess();
            return $auth;
        }


        // If we are a group admin of the owners groups
        if ($this->usertypeid == 2) {
            // Are we in the same group as the owner.
            $theirGroups = $this->GetUserGroups($ownerId, true);

            if (count(array_intersect($this->myGroups(),
$theirGroups))) {
                $auth->FullAccess();
                return $auth;
            }
        }

        // Permissions for groups the user is assigned to, and
Everyone
        $SQL  = '';
        $SQL .= 'SELECT UserID, MAX(IFNULL(View, 0)) AS View,
MAX(IFNULL(Edit, 0)) AS Edit, MAX(IFNULL(Del, 0)) AS Del ';
        $SQL .= '  FROM `campaign` ';
        $SQL .= '   INNER JOIN lkcampaigngroup ';
        $SQL .= '   ON lkcampaigngroup.CampaignID =
campaign.CampaignID ';
        $SQL .= '   INNER JOIN `group` ';
        $SQL .= '   ON `group`.GroupID = lkcampaigngroup.GroupID ';
        $SQL .= ' WHERE campaign.CampaignID = %d ';
        $SQL .= '   AND (`group`.IsEveryone = 1 OR `group`.GroupID
IN (%s)) ';
        $SQL .= 'GROUP BY campaign.UserID ';

        $SQL = sprintf($SQL, $campaignId, implode(',', $this-
>myGroups()));
        //Debug::LogEntry('audit', $SQL);

        if (!$row = $this->db->GetSingleRow($SQL))
            return $auth;

        // There are permissions to evaluate
        $auth->Evaluate($row['UserID'], $row['View'], $row['Edit'],
$row['Del']);

        if ($fullObject)
            return $auth;
```

```
        return $auth->edit;
    }

    /**
     * Authenticates the current user and returns an array
ofcampaigns this user is authenticated on
     * @return
     */
    public function CampaignList($name = '', $isRetired = false,
$showEmpty = true)
    {
        $db         =& $this->db;
        $userid     =& $this->userid;

        $layoutJoin = ($showEmpty) ? 'LEFT OUTER JOIN' : 'INNER
JOIN';

        $SQL  = "SELECT campaign.CampaignID, Campaign,
IsLayoutSpecific, COUNT(lkcampaignlayout.LayoutID) AS NumLayouts,
MIN(layout.retired) AS Retired ";
        $SQL .= "  FROM `campaign` ";
        $SQL .= "   $layoutJoin `lkcampaignlayout` ";
        $SQL .= "   ON lkcampaignlayout.CampaignID =
campaign.CampaignID ";
        $SQL .= "   $layoutJoin `layout` ";
        $SQL .= "   ON lkcampaignlayout.LayoutID = layout.LayoutID
";
        $SQL .= "       AND layout.layoutID NOT IN (SELECT layoutId
FROM lktaglayout WHERE tagId = 1) ";
        $SQL .= " WHERE 1 = 1 ";

        if ($name != '')
        {
            // convert into a space delimited array
            $names = explode(' ', $name);

            foreach($names as $searchName)
            {
                // Not like, or like?
                if (substr($searchName, 0, 1) == '-')
                    $SQL.= " AND  (campaign.Campaign NOT LIKE '%" .
sprintf('%s', ltrim($db->escape_string($searchName), '-')) . "%') ";
                else
                    $SQL.= " AND  (campaign.Campaign LIKE '%" .
sprintf('%s', $db->escape_string($searchName)) . "%') ";
            }
        }

        $SQL .= "GROUP BY campaign.CampaignID, Campaign,
IsLayoutSpecific ";
        $SQL .= "ORDER BY Campaign";

        if (!$result = $this->db->query($SQL))
```

```
        {
            trigger_error($this->db->error());
            return false;
        }

        $campaigns = array();

        while ($row = $this->db->get_assoc_row($result))
        {
            $campaignItem = array();

            // Validate each param and add it to the array.
            $campaignItem['campaignid'] =
Kit::ValidateParam($row['CampaignID'], _INT);
            $campaignItem['campaign'] =
Kit::ValidateParam($row['Campaign'], _STRING);
            $campaignItem['numlayouts'] =
Kit::ValidateParam($row['NumLayouts'], _INT);
            $campaignItem['islayoutspecific'] =
Kit::ValidateParam($row['IsLayoutSpecific'], _INT);
            $campaignItem['retired'] =
Kit::ValidateParam($row['Retired'], _BOOL);

            // return retired layouts?
            if (!$isRetired && $campaignItem['retired'])
                continue;

            $auth = $this->CampaignAuth($campaignItem['campaignid'],
true);

            if ($auth->view)
            {
                $campaignItem['view'] = (int) $auth->view;
                $campaignItem['edit'] = (int) $auth->edit;
                $campaignItem['del'] = (int) $auth->del;
                $campaignItem['modifypermissions'] = (int) $auth-
>modifyPermissions;

                $campaigns[] = $campaignItem;
            }
        }

        return $campaigns;
    }

    /**
     * Get a list of transitions
     * @param string in/out
     * @param string transition code
     * @return boolean
     */
    public function TransitionAuth($type = '', $code = '')
    {
        // Return a list of in/out transitions (or both)
```

```
        $SQL = 'SELECT TransitionID, ';
        $SQL .= '   Transition, ';
        $SQL .= '   Code, ';
        $SQL .= '   HasDuration, ';
        $SQL .= '   HasDirection, ';
        $SQL .= '   AvailableAsIn, ';
        $SQL .= '   AvailableAsOut ';
        $SQL .= '  FROM `transition` ';
        $SQL .= ' WHERE 1 = 1 ';

        if ($type != '')
        {
            // Filter on type
            if ($type == 'in')
                $SQL .= '  AND AvailableAsIn = 1 ';

            if ($type == 'out')
                $SQL .= '  AND AvailableAsOut = 1 ';
        }

        if ($code != '')
        {
            // Filter on code
            $SQL .= sprintf("AND Code = '%s' ", $this->db-
>escape_string($code));
        }

        $SQL .= ' ORDER BY Transition ';

        $rows = $this->db->GetArray($SQL);

        if (!is_array($rows)) {
            trigger_error($this->db->error());
            return false;
        }

        $transitions = array();

        foreach ($rows as $transition)
        {
            $transitionItem = array();

            $transitionItem['transitionid'] =
Kit::ValidateParam($transition['TransitionID'], _INT);
            $transitionItem['transition'] =
Kit::ValidateParam($transition['Transition'], _STRING);
            $transitionItem['code'] =
Kit::ValidateParam($transition['Code'], _WORD);
            $transitionItem['hasduration'] =
Kit::ValidateParam($transition['HasDuration'], _INT);
            $transitionItem['hasdirection'] =
Kit::ValidateParam($transition['HasDirection'], _INT);
            $transitionItem['enabledforin'] =
Kit::ValidateParam($transition['AvailableAsIn'], _INT);
```

```
            $transitionItem['enabledforout'] =
Kit::ValidateParam($transition['AvailableAsOut'], _INT);
            $transitionItem['class'] =
(($transitionItem['hasduration'] == 1) ? 'hasDuration' : '') . ' ' .
(($transitionItem['hasdirection'] == 1) ? 'hasDirection' : '');

            $transitions[] = $transitionItem;
        }

        return $transitions;
    }

    /**
     * List of Displays this user has access to view
     */
    public function DisplayProfileList($sort_order = array('name'),
$filter_by = array()) {

        try {
            $dbh = PDOConnect::init();

            $params = array();
            $SQL  = 'SELECT displayprofileid, name, type, config,
isdefault, userid FROM displayprofile ';

            $type = Kit::GetParam('type', $filter_by, _WORD);
            if (!empty($type)) {
                $SQL .= ' WHERE type = :type ';
                $params['type'] = $type;
            }

            // Sorting?
            if (is_array($sort_order))
                $SQL .= 'ORDER BY ' . implode(',', $sort_order);

            $sth = $dbh->prepare($SQL);
            $sth->execute($params);

            $profiles = array();

            while ($row = $sth->fetch()) {
                $displayItem = array();

                // Validate each param and add it to the array.
                $displayItem['displayprofileid'] =
Kit::ValidateParam($row['displayprofileid'], _INT);
                $displayItem['name'] =
Kit::ValidateParam($row['name'], _STRING);
                $displayItem['type'] =
Kit::ValidateParam($row['type'], _STRING);
                $displayItem['config'] =
Kit::ValidateParam($row['config'], _STRING);
                $displayItem['isdefault'] =
Kit::ValidateParam($row['isdefault'], _INT);
```

```
                $displayItem['userid'] =
Kit::ValidateParam($row['userid'], _INT);

                $auth = new PermissionManager($this);

                // If we are the owner, or a super admin then give
full permissions
                if ($this->usertypeid != 1 && $this->userid !=
$displayItem['userid'])
                    continue;

                $displayItem['view'] = 1;
                $displayItem['edit'] = 1;
                $displayItem['del'] = 1;
                $displayItem['modifypermissions'] = 1;

                $profiles[] = $displayItem;
            }

            return $profiles;
        }
        catch (Exception $e) {

            Debug::LogEntry('error', $e->getMessage(), get_class(),
__FUNCTION__);

            return false;
        }
    }

    public function userList($sortOrder = array('username'),
$filterBy = array())
    {
        // Normal users can only see themselves
        if ($this->usertypeid == 3) {
            $filterBy['userId'] = $this->userid;
        }
        // Group admins can only see users from their groups.
        else if ($this->usertypeid == 2) {
            $groups = $this->myGroups();
            $filterBy['groupIds'] = (isset($filterBy['groupIds'])) ?
array_merge($filterBy['groupIds'], $groups) : $groups;
        }

        try {
            $user = Userdata::entries($sortOrder, $filterBy);
            $parsedUser = array();

            foreach ($user as $row) {
                $userItem = array();

                // Validate each param and add it to the array.
                $userItem['userid'] = $row->userId;
                $userItem['username'] = $row->userName;
```

```
                $userItem['usertypeid'] = $row->userTypeId;
                $userItem['homepage'] = $row->homePage;
                $userItem['email'] = $row->email;
                $userItem['newuserwizard'] = $row->newUserWizard;
                $userItem['lastaccessed'] = $row->lastAccessed;
                $userItem['loggedin'] = $row->loggedIn;
                $userItem['retired'] = $row->retired;
                $userItem['object'] = $row;

                // Add to the collection
                $parsedUser[] = $userItem;
            }

            return $parsedUser;
        }
        catch (Exception $e) {
            Debug::LogEntry('error', $e->getMessage(), get_class(),
__FUNCTION__);
            return false;
        }
    }

    public function GetPref($key, $default = NULL) {
        $storedValue = Session::Get($key);

        return ($storedValue == NULL) ? $default : $storedValue;
    }

    public function SetPref($key, $value) {
        Session::Set($key, $value);
    }
}
?>
```