Source Code

Requested by: Dr. Eric Breimer
Web Master & Assistant Professor
Department of Computer Science
Siena College

Dr. Tim Lederman
Professor
Department of Computer Science
Siena College

Alumni Spotlight Web System (ASWS)

Initrode Solutions

Prepared by: Anthony Angelucci
Charles Feltes
Elise Hearn
Christopher McConnell

Monday, April 30th 2007

```
<?php
// $Id: form.inc,v 1.174.2.3 2007/01/29 21:51:53 drumm Exp $

/**
 * @defgroup form Form generation
 * @{
 * Functions to enable the processing and display of HTML forms.
 *
 * Drupal uses these functions to achieve consistency in its form processing
and
 * presentation, while simplifying code and reducing the amount of HTML that
 * must be explicitly generated by modules.
 *
 * The drupal_get_form() function handles retrieving, processing, and
 * displaying a rendered HTML form for modules automatically. For example:
 *
 * // Display the user registration form.
 * $output = drupal_get_form('user_register');
 *
 * Forms can also be built and submitted programmatically without any user
input
 * using the drupal_execute() function.
 *
 *
 * For information on the format of the structured arrays used to define forms,
 * and more detailed explanations of the Form API workflow, see the reference
at
 *
http://api.drupal.org/api/HEAD/file/developer/topics/forms_api_reference.html
 * and the quickstart guide at
 * http://api.drupal.org/api/HEAD/file/developer/topics/forms_api.html
 */

/**
 * Retrieves a form from a builder function, passes it on for
 * processing, and renders the form or redirects to its destination
 * as appropriate. In multi-step form scenarios, it handles properly
 * processing the values using the previous step's form definition,
 * then rendering the requested step for display.
 *
 * @param $form_id
 *   The unique string identifying the desired form. If a function
 *   with that name exists, it is called to build the form array.
 *   Modules that need to generate the same form (or very similar forms)
 *   using different $form_ids can implement hook_forms(), which maps
 *   different $form_id values to the proper form building function. Examples
 *   may be found in node_forms(), search_forms(), and user_forms().
 * @param ...
 *   Any additional arguments needed by the form building function.
 * @return
 *   The rendered form.
 */
function drupal_get_form($form_id) {
  // In multi-step form scenarios, the incoming $_POST values are not
  // necessarily intended for the current form. We need to build
  // a copy of the previously built form for validation and processing,
  // then go on to the one that was requested if everything works.

  $form_build_id = md5(mt_rand());
  if (isset($_POST['form_build_id']) &&
isset($_SESSION['form'][$_POST['form_build_id']]['args']) && $_POST['form_id']
== $form_id) {
    // There's a previously stored multi-step form. We should handle
```

```
    // IT first.
    $stored = TRUE;
    $args = $_SESSION['form'][$_POST['form_build_id']]['args'];
    $form = call_user_func_array('drupal_retrieve_form', $args);
    $form['#build_id'] = $_POST['form_build_id'];
  }
  else {
    // We're coming in fresh; build things as they would be. If the
    // form's #multistep flag is set, store the build parameters so
    // the same form can be reconstituted for validation.
    $args = func_get_args();
    $form = call_user_func_array('drupal_retrieve_form', $args);
    if (isset($form['#multistep']) && $form['#multistep']) {
      // Clean up old multistep form session data.
      _drupal_clean_form_sessions();
      $_SESSION['form'][$form_build_id] = array('timestamp' => time(), 'args'
=> $args);
      $form['#build_id'] = $form_build_id;
    }
    $stored = FALSE;
  }

  // Process the form, submit it, and store any errors if necessary.
  drupal_process_form($args[0], $form);

  if ($stored && !form_get_errors()) {
    // If it's a stored form and there were no errors, we processed the
    // stored form successfully. Now we need to build the form that was
    // actually requested. We always pass in the current $_POST values
    // to the builder function, as values from one stage of a multistep
    // form can determine how subsequent steps are displayed.
    $args = func_get_args();
    $args[] = $_POST;
    $form = call_user_func_array('drupal_retrieve_form', $args);
    unset($_SESSION['form'][$_POST['form_build_id']]);
    if (isset($form['#multistep']) && $form['#multistep']) {
      $_SESSION['form'][$form_build_id] = array('timestamp' => time(), 'args'
=> $args);
      $form['#build_id'] = $form_build_id;
    }
    drupal_prepare_form($args[0], $form);
  }

  return drupal_render_form($args[0], $form);
}


/**
 * Remove form information that's at least a day old from the
 * $_SESSION['form'] array.
 */
function _drupal_clean_form_sessions() {
  if (isset($_SESSION['form'])) {
    foreach ($_SESSION['form'] as $build_id => $data) {
      if ($data['timestamp'] < (time() - 84600)) {
        unset($_SESSION['form'][$build_id]);
      }
    }
  }
}


/**
```

```
 * Retrieves a form using a form_id, populates it with $form_values,
 * processes it, and returns any validation errors encountered. This
 * function is the programmatic counterpart to drupal_get_form().
 *
 * @param $form_id
 *   The unique string identifying the desired form. If a function
 *   with that name exists, it is called to build the form array.
 *   Modules that need to generate the same form (or very similar forms)
 *   using different $form_ids can implement hook_forms(), which maps
 *   different $form_id values to the proper form building function. Examples
 *   may be found in node_forms(), search_forms(), and user_forms().
 * @param $form_values
 *   An array of values mirroring the values returned by a given form
 *   when it is submitted by a user.
 * @param ...
 *   Any additional arguments needed by the form building function.
 * @return
 *   Any form validation errors encountered.
 *
 * For example:
 *
 * // register a new user
 * $values['name'] = 'robo-user';
 * $values['mail'] = 'robouser@example.com';
 * $values['pass'] = 'password';
 * drupal_execute('user_register', $values);
 *
 * // Create a new node
 * $node = array('type' => 'story');
 * $values['title'] = 'My node';
 * $values['body'] = 'This is the body text!';
 * $values['name'] = 'robo-user';
 * drupal_execute('story_node_form', $values, $node);
 */
function drupal_execute($form_id, $form_values) {
  $args = func_get_args();

  $form_id = array_shift($args);
  $form_values = array_shift($args);
  array_unshift($args, $form_id);

  if (isset($form_values)) {
    $form = call_user_func_array('drupal_retrieve_form', $args);
    $form['#post'] = $form_values;
    return drupal_process_form($form_id, $form);
  }
}

/**
 * Retrieves the structured array that defines a given form.
 *
 * @param $form_id
 *   The unique string identifying the desired form. If a function
 *   with that name exists, it is called to build the form array.
 *   Modules that need to generate the same form (or very similar forms)
 *   using different $form_ids can implement hook_forms(), which maps
 *   different $form_id values to the proper form building function.
 * @param ...
 *   Any additional arguments needed by the form building function.
 */
function drupal_retrieve_form($form_id) {
  static $forms;
```

```
  // We save two copies of the incoming arguments: one for modules to use
  // when mapping form ids to builder functions, and another to pass to
  // the builder function itself. We shift out the first argument -- the
  // $form_id itself -- from the list to pass into the builder function,
  // since it's already known.
  $args = func_get_args();
  $saved_args = $args;
  array_shift($args);

  // We first check to see if there's a function named after the $form_id.
  // If there is, we simply pass the arguments on to it to get the form.
  if (!function_exists($form_id)) {
    // In cases where many form_ids need to share a central builder function,
    // such as the node editing form, modules can implement hook_forms(). It
    // maps one or more form_ids to the correct builder functions.
    //
    // We cache the results of that hook to save time, but that only works
    // for modules that know all their form_ids in advance. (A module that
    // adds a small 'rate this comment' form to each comment in a list
    // would need a unique form_id for each one, for example.)
    //
    // So, we call the hook if $forms isn't yet populated, OR if it doesn't
    // yet have an entry for the requested form_id.
    if (!isset($forms) || !isset($forms[$form_id])) {
      $forms = module_invoke_all('forms', $saved_args);
    }
    $form_definition = $forms[$form_id];
    if (isset($form_definition['callback arguments'])) {
      $args = array_merge($form_definition['callback arguments'], $args);
    }
    if (isset($form_definition['callback'])) {
      $callback = $form_definition['callback'];
    }
  }
  // If $callback was returned by a hook_forms() implementation, call it.
  // Otherwise, call the function named after the form id.
  $form = call_user_func_array(isset($callback) ? $callback : $form_id, $args);

  // We store the original function arguments, rather than the final $arg
  // value, so that form_alter functions can see what was originally
  // passed to drupal_retrieve_form(). This allows the contents of #parameters
  // to be saved and passed in at a later date to recreate the form.
  $form['#parameters'] = $saved_args;
  return $form;
}

/**
 * This function is the heart of form API. The form gets built, validated and
in
 * appropriate cases, submitted.
 *
 * @param $form_id
 *   The unique string identifying the current form.
 * @param $form
 *   An associative array containing the structure of the form.
 * @return
 *   The path to redirect the user to upon completion.
 */
function drupal_process_form($form_id, &$form) {
  global $form_values, $form_submitted, $user, $form_button_counter;
  static $saved_globals = array();
  // In some scenarios, this function can be called recursively. Pushing any
pre-existing
```

```
  // $form_values and form submission data lets us start fresh without
clobbering work done
  // in earlier recursive calls.
  array_push($saved_globals, array($form_values, $form_submitted,
$form_button_counter));

  $form_values = array();
  $form_submitted = FALSE;
  $form_button_counter = array(0, 0);

  drupal_prepare_form($form_id, $form);
  if (($form['#programmed']) || (!empty($_POST) && (($_POST['form_id'] ==
$form_id) || ($_POST['form_id'] == $form['#base'])))) {
    drupal_validate_form($form_id, $form);
    // IE does not send a button value when there is only one submit button
(and no non-submit buttons)
    // and you submit by pressing enter.
    // In that case we accept a submission without button values.
    if ((($form['#programmed']) || $form_submitted || (!$form_button_counter[0]
&& $form_button_counter[1])) && !form_get_errors()) {
      $redirect = drupal_submit_form($form_id, $form);
      if (!$form['#programmed']) {
        drupal_redirect_form($form, $redirect);
      }
    }
  }

  // We've finished calling functions that alter the global values, so we can
  // restore the ones that were there before this function was called.
  list($form_values, $form_submitted, $form_button_counter) =
array_pop($saved_globals);
  return $redirect;
}

/**
 * Prepares a structured form array by adding required elements,
 * executing any hook_form_alter functions, and optionally inserting
 * a validation token to prevent tampering.
 *
 * @param $form_id
 *   A unique string identifying the form for validation, submission,
 *   theming, and hook_form_alter functions.
 * @param $form
 *   An associative array containing the structure of the form.
 */
function drupal_prepare_form($form_id, &$form) {
  global $user;

  $form['#type'] = 'form';

  if (!isset($form['#post'])) {
    $form['#post'] = $_POST;
    $form['#programmed'] = FALSE;
  }
  else {
    $form['#programmed'] = TRUE;
  }

  // In multi-step form scenarios, this id is used to identify
  // a unique instance of a particular form for retrieval.
  if (isset($form['#build_id'])) {
    $form['form_build_id'] = array(
      '#type' => 'hidden',
```

```
      '#value' => $form['#build_id'],
      '#id' => $form['#build_id'],
      '#name' => 'form_build_id',
    );
  }

  // If $base is set, it is used in place of $form_id when constructing
validation,
  // submission, and theming functions. Useful for mapping many similar or
duplicate
  // forms with different $form_ids to the same processing functions.
  if (isset($form['#base'])) {
    $base = $form['#base'];
  }

  // Add a token, based on either #token or form_id, to any form displayed to
  // authenticated users. This ensures that any submitted form was actually
  // requested previously by the user and protects against cross site request
  // forgeries.
  if (isset($form['#token'])) {
    if ($form['#token'] === FALSE || $user->uid == 0 || $form['#programmed']) {
      unset($form['#token']);
    }
    else {
      $form['form_token'] = array('#type' => 'token', '#default_value' =>
drupal_get_token($form['#token']));
    }
  }
  else if ($user->uid && !$form['#programmed']) {
    $form['#token'] = $form_id;
    $form['form_token'] = array(
      '#id' => form_clean_id('edit-'. $form_id .'-form-token'),
      '#type' => 'token',
      '#default_value' => drupal_get_token($form['#token']),
    );
  }


  if (isset($form_id)) {
    $form['form_id'] = array('#type' => 'hidden', '#value' => $form_id, '#id'
=> form_clean_id("edit-$form_id"));
  }
  if (!isset($form['#id'])) {
    $form['#id'] = form_clean_id($form_id);
  }

  $form += _element_info('form');

  if (!isset($form['#validate'])) {
    if (function_exists($form_id .'_validate')) {
      $form['#validate'] = array($form_id .'_validate' => array());
    }
    elseif (function_exists($base .'_validate')) {
      $form['#validate'] = array($base .'_validate' => array());
    }
  }

  if (!isset($form['#submit'])) {
    if (function_exists($form_id .'_submit')) {
      // We set submit here so that it can be altered.
      $form['#submit'] = array($form_id .'_submit' => array());
    }
    elseif (function_exists($base .'_submit')) {
```

```
      $form['#submit'] = array($base .'_submit' => array());
    }
  }

  foreach (module_implements('form_alter') as $module) {
    $function = $module .'_form_alter';
    $function($form_id, $form);
  }

  $form = form_builder($form_id, $form);
}


/**
 * Validates user-submitted form data from a global variable using
 * the validate functions defined in a structured form array.
 *
 * @param $form_id
 *   A unique string identifying the form for validation, submission,
 *   theming, and hook_form_alter functions.
 * @param $form
 *   An associative array containing the structure of the form.
 *
 */
function drupal_validate_form($form_id, $form) {
  global $form_values;
  static $validated_forms = array();

  if (isset($validated_forms[$form_id])) {
    return;
  }

  // If the session token was set by drupal_prepare_form(), ensure that it
  // matches the current user's session.
  if (isset($form['#token'])) {
    if (!drupal_valid_token($form_values['form_token'], $form['#token'])) {
      // Setting this error will cause the form to fail validation.
      form_set_error('form_token', t('Validation error, please try again. If
this error persists, please contact the site administrator.'));
    }
  }

  _form_validate($form, $form_id);
  $validated_forms[$form_id] = TRUE;
}

/**
 * Processes user-submitted form data from a global variable using
 * the submit functions defined in a structured form array.
 *
 * @param $form_id
 *   A unique string identifying the form for validation, submission,
 *   theming, and hook_form_alter functions.
 * @param $form
 *   An associative array containing the structure of the form.
 * @return
 *   A string containing the path of the page to display when processing
 *   is complete.
 *
 */
function drupal_submit_form($form_id, $form) {
  global $form_values;
  $default_args = array($form_id, &$form_values);
```

```
  if (isset($form['#submit'])) {
    foreach ($form['#submit'] as $function => $args) {
      if (function_exists($function)) {
        $args = array_merge($default_args, (array) $args);
        // Since we can only redirect to one page, only the last redirect
        // will work.
        $redirect = call_user_func_array($function, $args);
        if (isset($redirect)) {
          $goto = $redirect;
        }
      }
    }
  }
  return $goto;
}

/**
 * Renders a structured form array into themed HTML.
 *
 * @param $form_id
 *   A unique string identifying the form for validation, submission,
 *   theming, and hook_form_alter functions.
 * @param $form
 *   An associative array containing the structure of the form.
 * @return
 *   A string containing the path of the page to display when processing
 *   is complete.
 *
 */
function drupal_render_form($form_id, &$form) {
  // Don't override #theme if someone already set it.
  if (isset($form['#base'])) {
    $base = $form['#base'];
  }

  if (!isset($form['#theme'])) {
    if (theme_get_function($form_id)) {
      $form['#theme'] = $form_id;
    }
    elseif (theme_get_function($base)) {
      $form['#theme'] = $base;
    }
  }

  if (isset($form['#pre_render'])) {
    foreach ($form['#pre_render'] as $function) {
      if (function_exists($function)) {
        $function($form_id, $form);
      }
    }
  }

  $output = drupal_render($form);
  return $output;
}

/**
 * Redirect the user to a URL after a form has been processed.
 *
 * @param $form
 *   An associative array containing the structure of the form.
 * @param $redirect
```

```
 *   An optional string containing the destination path to redirect
 *   to if none is specified by the form.
 *
 */
function drupal_redirect_form($form, $redirect = NULL) {
  if (isset($redirect)) {
    $goto = $redirect;
  }
  if (isset($form['#redirect'])) {
    $goto = $form['#redirect'];
  }
  if ($goto !== FALSE) {
    if (is_array($goto)) {
      call_user_func_array('drupal_goto', $goto);
    }
    elseif (!isset($goto)) {
      drupal_goto($_GET['q']);
    }
    else {
      drupal_goto($goto);
    }
  }
}

/**
 * Performs validation on form elements. First ensures required fields are
 * completed, #maxlength is not exceeded, and selected options were in the
 * list of options given to the user. Then calls user-defined validators.
 *
 * @param $elements
 *   An associative array containing the structure of the form.
 * @param $form_id
 *   A unique string identifying the form for validation, submission,
 *   theming, and hook_form_alter functions.
 */
function _form_validate($elements, $form_id = NULL) {
  // Recurse through all children.
  foreach (element_children($elements) as $key) {
    if (isset($elements[$key]) && $elements[$key]) {
      _form_validate($elements[$key]);
    }
  }
  /* Validate the current input */
  if (!isset($elements['#validated']) || !$elements['#validated']) {
    if (isset($elements['#needs_validation'])) {
      // An empty textfield returns '' so we use empty(). An empty checkbox
      // and a textfield could return '0' and empty('0') returns TRUE so we
      // need a special check for the '0' string.
      if ($elements['#required'] && empty($elements['#value']) &&
$elements['#value'] !== '0') {
        form_error($elements, t('!name field is required.', array('!name' =>
$elements['#title'])));
      }

      // Verify that the value is not longer than #maxlength.
      if (isset($elements['#maxlength']) && drupal_strlen($elements['#value'])
> $elements['#maxlength']) {
        form_error($elements, t('!name cannot be longer than %max characters
but is currently %length characters long.', array('!name' =>
empty($elements['#title']) ? $elements['#parents'][0] : $elements['#title'],
'%max' => $elements['#maxlength'], '%length' =>
drupal_strlen($elements['#value']))));
      }
```

```
       // Add legal choice check if element has #options. Can be skipped, but
       // then you must validate your own element.
      if (isset($elements['#options']) && isset($elements['#value']) &&
!isset($elements['#DANGEROUS_SKIP_CHECK'])) {
        if ($elements['#type'] == 'select') {
          $options = form_options_flatten($elements['#options']);
        }
        else {
          $options = $elements['#options'];
        }
        if (is_array($elements['#value'])) {
          $value = $elements['#type'] == 'checkboxes' ?
array_keys(array_filter($elements['#value'])) : $elements['#value'];
          foreach ($value as $v) {
            if (!isset($options[$v])) {
              form_error($elements, t('An illegal choice has been detected.
Please contact the site administrator.'));
              watchdog('form', t('Illegal choice %choice in !name element.',
array('%choice' => $v, '!name' => empty($elements['#title']) ?
$elements['#parents'][0] : $elements['#title'])), WATCHDOG_ERROR);
            }
          }
        }
        elseif (!isset($options[$elements['#value']])) {
          form_error($elements, t('An illegal choice has been detected. Please
contact the site administrator.'));
          watchdog('form', t('Illegal choice %choice in %name element.',
array('%choice' => $elements['#value'], '%name' => empty($elements['#title']) ?
$elements['#parents'][0] : $elements['#title'])), WATCHDOG_ERROR);
        }
      }
    }

    // Call user-defined validators.
    if (isset($elements['#validate'])) {
      foreach ($elements['#validate'] as $function => $args) {
        $args = array_merge(array($elements), $args);
        // For the full form we hand over a copy of $form_values.
        if (isset($form_id)) {
          $args = array_merge(array($form_id, $GLOBALS['form_values']), $args);
        }
        if (function_exists($function))  {
          call_user_func_array($function, $args);
        }
      }
    }
    $elements['#validated'] = TRUE;
  }
}

/**
 * File an error against a form element. If the name of the element is
 * edit[foo][bar] then you may pass either foo or foo][bar as $name
 * foo will set an error for all its children.
 */
function form_set_error($name = NULL, $message = '') {
  static $form = array();
  if (isset($name) && !isset($form[$name])) {
    $form[$name] = $message;
    if ($message) {
      drupal_set_message($message, 'error');
    }
```

```
  }
  return $form;
}

/**
 * Return an associative array of all errors.
 */
function form_get_errors() {
  $form = form_set_error();
  if (!empty($form)) {
    return $form;
  }
}

/**
 * Return the error message filed against the form with the specified name.
 */
function form_get_error($element) {
  $form = form_set_error();
  $key = $element['#parents'][0];
  if (isset($form[$key])) {
    return $form[$key];
  }
  $key = implode('][', $element['#parents']);
  if (isset($form[$key])) {
    return $form[$key];
  }
}

/**
 * Flag an element as having an error.
 */
function form_error(&$element, $message = '') {
  $element['#error'] = TRUE;
  form_set_error(implode('][', $element['#parents']), $message);
}

/**
 * Adds some required properties to each form element, which are used
 * internally in the form API. This function also automatically assigns
 * the value property from the $edit array, provided the element doesn't
 * already have an assigned value.
 *
 * @param $form_id
 *   A unique string identifying the form for validation, submission,
 *   theming, and hook_form_alter functions.
 * @param $form
 *   An associative array containing the structure of the form.
 */
function form_builder($form_id, $form) {
  global $form_values, $form_submitted, $form_button_counter;

  // Initialize as unprocessed.
  $form['#processed'] = FALSE;

  /* Use element defaults */
  if ((!empty($form['#type'])) && ($info = _element_info($form['#type']))) {
    // Overlay $info onto $form, retaining preexisting keys in $form.
    $form += $info;
  }

  if (isset($form['#input']) && $form['#input']) {
    if (!isset($form['#name'])) {
```

```
      $name = array_shift($form['#parents']);
      $form['#name'] = $name;
      if ($form['#type'] == 'file') {
        // To make it easier to handle $_FILES in file.inc, we place all
        // file fields in the 'files' array. Also, we do not support
        // nested file names.
        $form['#name'] = 'files['. $form['#name'] .']';
      }
      elseif (count($form['#parents'])) {
        $form['#name'] .= '['. implode('][', $form['#parents']) .']';
      }
      array_unshift($form['#parents'], $name);
    }
    if (!isset($form['#id'])) {
      $form['#id'] = form_clean_id('edit-'. implode('-', $form['#parents']));
    }

    if (isset($form['#disabled']) && $form['#disabled']) {
      $form['#attributes']['disabled'] = 'disabled';
    }

    if (!isset($form['#value']) && !array_key_exists('#value', $form)) {
      if (($form['#programmed']) || ((!isset($form['#access']) ||
$form['#access']) && isset($form['#post']) && (isset($form['#post']['form_id'])
&& $form['#post']['form_id'] == $form_id))) {
        $edit = $form['#post'];
        foreach ($form['#parents'] as $parent) {
          $edit = isset($edit[$parent]) ? $edit[$parent] : NULL;
        }
        if (!$form['#programmed'] || isset($edit)) {
          switch ($form['#type']) {
            case 'checkbox':
              $form['#value'] = !empty($edit) ? $form['#return_value'] : 0;
              break;

            case 'select':
              if (isset($form['#multiple']) && $form['#multiple']) {
                if (isset($edit) && is_array($edit)) {
                  $form['#value'] = drupal_map_assoc($edit);
                }
                else {
                  $form['#value'] = array();
                }
              }
              elseif (isset($edit)) {
                $form['#value'] = $edit;
              }
              break;

            case 'textfield':
              if (isset($edit)) {
                // Equate $edit to the form value to ensure it's marked for
                // validation.
                $edit = str_replace(array("\r", "\n"), '', $edit);
                $form['#value'] = $edit;
              }
              break;

            case 'token':
              $form['#value'] = (string)$edit;
              break;

            default:
```

```
              if (isset($edit)) {
                $form['#value'] = $edit;
              }
            }
            // Mark all posted values for validation.
            if ((isset($form['#value']) && $form['#value'] === $edit) ||
(isset($form['#required']) && $form['#required'])) {
              $form['#needs_validation'] = TRUE;
            }
          }
        }
        if (!isset($form['#value'])) {
          $function = $form['#type'] . '_value';
          if (function_exists($function)) {
            $function($form);
          }
          else {
            $form['#value'] = isset($form['#default_value']) ?
$form['#default_value'] : '';
          }
        }
      }
      if (isset($form['#executes_submit_callback'])) {
        // Count submit and non-submit buttons.
        $form_button_counter[$form['#executes_submit_callback']]++;
        // See if a submit button was pressed.
        if (isset($form['#post'][$form['#name']]) &&
$form['#post'][$form['#name']] == $form['#value']) {
          $form_submitted = $form_submitted ||
$form['#executes_submit_callback'];

          // In most cases, we want to use form_set_value() to manipulate the
          // global variables. In this special case, we want to make sure that
          // the value of this element is listed in $form_variables under 'op'.
          $form_values[$form['#name']] = $form['#value'];
        }
      }
    }

    // Allow for elements to expand to multiple elements, e.g., radios,
    // checkboxes and files.
    if (isset($form['#process']) && !$form['#processed']) {
      foreach ($form['#process'] as $process => $args) {
        if (function_exists($process)) {
          $args = array_merge(array($form), array($edit), $args);
          $form = call_user_func_array($process, $args);
        }
      }
      $form['#processed'] = TRUE;
    }

    // Set the $form_values key that gets passed to validate and submit.
    // We call this after #process gets called so that #process has a
    // chance to update #value if desired.
    if (isset($form['#input']) && $form['#input']) {
      form_set_value($form, $form['#value']);
    }

    // We start off assuming all form elements are in the correct order.
    $form['#sorted'] = TRUE;

    // Recurse through all child elements.
    $count = 0;
```

```
  foreach (element_children($form) as $key) {
    $form[$key]['#post'] = $form['#post'];
    $form[$key]['#programmed'] = $form['#programmed'];
    // Don't squash an existing tree value.
    if (!isset($form[$key]['#tree'])) {
      $form[$key]['#tree'] = $form['#tree'];
    }

    // Deny access to child elements if parent is denied.
    if (isset($form['#access']) && !$form['#access']) {
      $form[$key]['#access'] = FALSE;
    }

    // Don't squash existing parents value.
    if (!isset($form[$key]['#parents'])) {
      // Check to see if a tree of child elements is present. If so,
      // continue down the tree if required.
      $form[$key]['#parents'] = $form[$key]['#tree'] && $form['#tree'] ?
array_merge($form['#parents'], array($key)) : array($key);
    }

    // Assign a decimal placeholder weight to preserve original array order.
    if (!isset($form[$key]['#weight'])) {
      $form[$key]['#weight'] = $count/1000;
    }
    else {
      // If one of the child elements has a weight then we will need to sort
      // later.
      unset($form['#sorted']);
    }
    $form[$key] = form_builder($form_id, $form[$key]);
    $count++;
  }

  if (isset($form['#after_build']) && !isset($form['#after_build_done'])) {
    foreach ($form['#after_build'] as $function) {
      if (function_exists($function)) {
        $form = $function($form, $form_values);
      }
    }
    $form['#after_build_done'] = TRUE;
  }

  return $form;
}

/**
 * Use this function to make changes to form values in the form validate
 * phase, so they will be available in the submit phase in $form_values.
 *
 * Specifically, if $form['#parents'] is array('foo', 'bar')
 * and $value is 'baz' then this function will make
 * $form_values['foo']['bar'] to be 'baz'.
 *
 * @param $form
 *   The form item. Keys used: #parents, #value
 * @param $value
 *   The value for the form item.
 */
function form_set_value($form, $value) {
  global $form_values;
  _form_set_value($form_values, $form, $form['#parents'], $value);
}
```

```
/**
 * Helper function for form_set_value().
 *
 * We iterate over $parents and create nested arrays for them
 * in $form_values if needed. Then we insert the value into
 * the right array.
 */
function _form_set_value(&$form_values, $form, $parents, $value) {
  $parent = array_shift($parents);
  if (empty($parents)) {
    $form_values[$parent] = $value;
  }
  else {
    if (!isset($form_values[$parent])) {
      $form_values[$parent] = array();
    }
    _form_set_value($form_values[$parent], $form, $parents, $value);
  }
  return $form;
}

/**
 * Retrieve the default properties for the defined element type.
 */
function _element_info($type, $refresh = NULL) {
  static $cache;

  $basic_defaults = array(
    '#description' => NULL,
    '#attributes' => array(),
    '#required' => FALSE,
    '#tree' => FALSE,
    '#parents' => array()
  );
  if (!isset($cache) || $refresh) {
    $cache = array();
    foreach (module_implements('elements') as $module) {
      $elements = module_invoke($module, 'elements');
      if (isset($elements) && is_array($elements)) {
        $cache = array_merge_recursive($cache, $elements);
      }
    }
    if (sizeof($cache)) {
      foreach ($cache as $element_type => $info) {
        $cache[$element_type] = array_merge_recursive($basic_defaults, $info);
      }
    }
  }

  return $cache[$type];
}

function form_options_flatten($array, $reset = TRUE) {
  static $return;

  if ($reset) {
    $return = array();
  }

  foreach ($array as $key => $value) {
    if (is_object($value)) {
      form_options_flatten($value->option, FALSE);
```

```
    }
    else if (is_array($value)) {
      form_options_flatten($value, FALSE);
    }
    else {
      $return[$key] = 1;
    }
  }

  return $return;
}

/**
 * Format a dropdown menu or scrolling selection box.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: title, value, options, description, extra, multiple,
required
 * @return
 *   A themed HTML string representing the form element.
 *
 * It is possible to group options together; to do this, change the format of
 * $options to an associative array in which the keys are group labels, and the
 * values are associative arrays in the normal $options format.
 */
function theme_select($element) {
  $select = '';
  $size = $element['#size'] ? ' size="' . $element['#size'] . '"' : '';
  _form_set_class($element, array('form-select'));
  $multiple = isset($element['#multiple']) && $element['#multiple'];
  return theme('form_element', $element, '<select name="'. $element['#name']
.''. ($multiple ? '[]' : '') .'"'. ($multiple ? ' multiple="multiple" ' : '') .
drupal_attributes($element['#attributes']) .' id="'. $element['#id'] .'" '.
$size .'>'. form_select_options($element) .'</select>');
}

function form_select_options($element, $choices = NULL) {
  if (!isset($choices)) {
    $choices = $element['#options'];
  }
  // array_key_exists() accommodates the rare event where $element['#value'] is
NULL.
  // isset() fails in this situation.
  $value_valid = isset($element['#value']) || array_key_exists('#value',
$element);
  $value_is_array = is_array($element['#value']);
  $options = '';
  foreach ($choices as $key => $choice) {
    if (is_array($choice)) {
      $options .= '<optgroup label="'. $key .'">';
      $options .= form_select_options($element, $choice);
      $options .= '</optgroup>';
    }
    elseif (is_object($choice)) {
      $options .= form_select_options($element, $choice->option);
    }
    else {
      $key = (string)$key;
      if ($value_valid && ((string)$element['#value'] === $key ||
($value_is_array && in_array($key, $element['#value'])))) {
        $selected = ' selected="selected"';
      }
```

```
      else {
        $selected = '';
      }
      $options .= '<option value="'. check_plain($key) .'"'. $selected .'>'.
check_plain($choice) .'</option>';
    }
  }
  return $options;
}

/**
 * Traverses a select element's #option array looking for any values
 * that hold the given key. Returns an array of indexes that match.
 *
 * This function is useful if you need to modify the options that are
 * already in a form element; for example, to remove choices which are
 * not valid because of additional filters imposed by another module.
 * One example might be altering the choices in a taxonomy selector.
 * To correctly handle the case of a multiple hierarchy taxonomy,
 * #options arrays can now hold an array of objects, instead of a
 * direct mapping of keys to labels, so that multiple choices in the
 * selector can have the same key (and label). This makes it difficult
 * to manipulate directly, which is why this helper function exists.
 *
 * This function does not support optgroups (when the elements of the
 * #options array are themselves arrays), and will return FALSE if
 * arrays are found. The caller must either flatten/restore or
 * manually do their manipulations in this case, since returning the
 * index is not sufficient, and supporting this would make the
 * "helper" too complicated and cumbersome to be of any help.
 *
 * As usual with functions that can return array() or FALSE, do not
 * forget to use === and !== if needed.
 *
 * @param $element
 *   The select element to search.
 * @param $key
 *   The key to look for.
 * @return
 *   An array of indexes that match the given $key. Array will be
 *   empty if no elements were found. FALSE if optgroups were found.
 */
function form_get_options($element, $key) {
  $keys = array();
  foreach ($element['#options'] as $index => $choice) {
    if (is_array($choice)) {
      return FALSE;
    }
    else if (is_object($choice)) {
      if (isset($choice->option[$key])) {
        $keys[] = $index;
      }
    }
    else if ($index == $key) {
      $keys[] = $index;
    }
  }
  return $keys;
}

/**
 * Format a group of form items.
 *
```

```
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: attributes, title, value, description, children,
collapsible, collapsed
 * @return
 *   A themed HTML string representing the form item group.
 */
function theme_fieldset($element) {
  if ($element['#collapsible']) {
    drupal_add_js('misc/collapse.js');

    if (!isset($element['#attributes']['class'])) {
      $element['#attributes']['class'] = '';
    }

    $element['#attributes']['class'] .= ' collapsible';
    if ($element['#collapsed']) {
     $element['#attributes']['class'] .= ' collapsed';
    }
  }

  return '<fieldset' . drupal_attributes($element['#attributes']) .'>' .
($element['#title'] ? '<legend>'. $element['#title'] .'</legend>' : '') .
($element['#description'] ? '<div class="description">'.
$element['#description'] .'</div>' : '') . $element['#children'] .
$element['#value'] . "</fieldset>\n";
}

/**
 * Format a radio button.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: required, return_value, value, attributes, title,
description
 * @return
 *   A themed HTML string representing the form item group.
 */
function theme_radio($element) {
  _form_set_class($element, array('form-radio'));
  $output = '<input type="radio" ';
  $output .= 'name="' . $element['#name'] .'" ';
  $output .= 'value="'. $element['#return_value'] .'" ';
  $output .= (check_plain($element['#value']) == $element['#return_value']) ? '
checked="checked" ' : ' ';
  $output .= drupal_attributes($element['#attributes']) .' />';
  if (!is_null($element['#title'])) {
    $output = '<label class="option">'. $output .' '. $element['#title']
.'</label>';
  }

  unset($element['#title']);
  return theme('form_element', $element, $output);
}

/**
 * Format a set of radio buttons.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: title, value, options, description, required and
attributes.
 * @return
```

```
 *   A themed HTML string representing the radio button set.
 */
function theme_radios($element) {
  $class = 'form-radios';
  if (isset($element['#attributes']['class'])) {
    $class .= ' '. $element['#attributes']['class'];
  }
  $element['#children'] = '<div class="'. $class .'">'. $element['#children']
.'</div>';
  if ($element['#title'] || $element['#description']) {
    unset($element['#id']);
    return theme('form_element', $element, $element['#children']);
  }
  else {
    return $element['#children'];
  }
}

/**
 * Format a password_confirm item.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: title, value, id, required, error.
 * @return
 *   A themed HTML string representing the form item.
 */
function theme_password_confirm($element) {
  return theme('form_element', $element, $element['#children']);
}

/*
 * Expand a password_confirm field into two text boxes.
 *
 * Added #maxlength to password fields - CF
 */
function expand_password_confirm($element) {
  $element['pass1'] =  array(
    '#type' => 'password',
    '#title' => t('Password'),
    '#value' => $element['#value']['pass1'],
    '#maxlength' => 12,
  );
  $element['pass2'] =  array(
    '#type' => 'password',
    '#title' => t('Confirm password'),
    '#value' => $element['#value']['pass2'],
    '#maxlength' => 12,
  );

  $element['#validate'] = array('password_confirm_validate' => array());
  $element['#tree'] = TRUE;

  if (isset($element['#size'])) {
    $element['pass1']['#size'] = $element['pass2']['#size'] =
$element['#size'];
  }

  return $element;
}

/**
 * Validate password_confirm element.
```

```
 */
function password_confirm_validate($form) {
  $pass1 = trim($form['pass1']['#value']);
 if ( $form['#required'] )
 {
   if (!empty($pass1)) {
      $pass2 = trim($form['pass2']['#value']);
      if ($pass1 != $pass2) {
        form_error($form, t('The specified passwords do not match.'));
         }
    }
    elseif ($form['#required'] && !empty($form['#post'])) {
      form_error($form, t('Password field is required.'));
    }

    if(!(ctype_alnum($pass1) && strlen($pass1)>5 && strlen($pass1)<13
         && preg_match('`[0-9]`',$pass1) && (preg_match('`[A-Z]`',$pass1)
         || preg_match('`[a-z]`',$pass1))) )
         {
          form_error($form, t('The password must be 6-12 characters in length
and contain at least one digit and one character.'));
         }
    }
else
 {
   if (!empty($pass1))
       {
        $pass2 = trim($form['pass2']['#value']);
        if ($pass1 != $pass2) {
          form_error($form, t('The specified passwords do not match.'));
        }
        if(!(ctype_alnum($pass1) && strlen($pass1)>5 && strlen($pass1)<13
         && preg_match('`[0-9]`',$pass1) && (preg_match('`[A-Z]`',$pass1)
         || preg_match('`[a-z]`',$pass1))) )
         {
          form_error($form, t('The password must be 6-12 characters in length
and contain at least one digit and one character.'));
         }
      }
 }


  // Password field must be converted from a two-element array into a single
  // string regardless of validation results.
  form_set_value($form['pass1'], NULL);
  form_set_value($form['pass2'], NULL);
  form_set_value($form, $pass1);

  return $form;
}

/**
 * Format a date selection element.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: title, value, options, description, required and
attributes.
 * @return
 *   A themed HTML string representing the date selection boxes.
 */
function theme_date($element) {
```

```
  return theme('form_element', $element, '<div class="container-inline">'.
$element['#children'] .'</div>');
}

/**
 * Roll out a single date element.
 */
function expand_date($element) {
  // Default to current date
  if (!isset($element['#value'])) {
    $element['#value'] = array('day' => format_date(time(), 'custom', 'j'),
                               'month' => format_date(time(), 'custom', 'n'),
                               'year' => format_date(time(), 'custom', 'Y'));
  }

  $element['#tree'] = TRUE;

  // Determine the order of day, month, year in the site's chosen date format.
  $format = variable_get('date_format_short', 'm/d/Y - H:i');
  $sort = array();
  $sort['day'] = max(strpos($format, 'd'), strpos($format, 'j'));
  $sort['month'] = max(strpos($format, 'm'), strpos($format, 'M'));
  $sort['year'] = strpos($format, 'Y');
  asort($sort);
  $order = array_keys($sort);

  // Output multi-selector for date.
  foreach ($order as $type) {
    switch ($type) {
      case 'day':
        $options = drupal_map_assoc(range(1, 31));
        break;
      case 'month':
        $options = drupal_map_assoc(range(1, 12), 'map_month');
        break;
      case 'year':
        $options = drupal_map_assoc(range(1900, 2050));
        break;
    }
    $parents = $element['#parents'];
    $parents[] = $type;
    $element[$type] = array(
      '#type' => 'select',
      '#value' => $element['#value'][$type],
      '#attributes' => $element['#attributes'],
      '#options' => $options,
    );
  }

  return $element;
}

/**
 * Validates the date type to stop dates like February 30, 2006.
 */
function date_validate($form) {
  if (!checkdate($form['#value']['month'], $form['#value']['day'],
$form['#value']['year'])) {
    form_error($form, t('The specified date is invalid.'));
  }
}

/**
```

```
 * Helper function for usage with drupal_map_assoc to display month names.
 */
function map_month($month) {
  return format_date(gmmktime(0, 0, 0, $month, 2, 1970), 'custom', 'M', 0);
}

/**
 * Helper function to load value from default value for checkboxes.
 */
function checkboxes_value(&$form) {
  $value = array();
  foreach ((array)$form['#default_value'] as $key) {
    $value[$key] = 1;
  }
  $form['#value'] = $value;
}

/**
 * If no default value is set for weight select boxes, use 0.
 */
function weight_value(&$form) {
  if (isset($form['#default_value'])) {
    $form['#value'] = $form['#default_value'];
  }
  else {
    $form['#value'] = 0;
  }
}

/**
 * Roll out a single radios element to a list of radios,
 * using the options array as index.
 */
function expand_radios($element) {
  if (count($element['#options']) > 0) {
    foreach ($element['#options'] as $key => $choice) {
      if (!isset($element[$key])) {
        $element[$key] = array('#type' => 'radio', '#title' => $choice,
'#return_value' => check_plain($key), '#default_value' =>
$element['#default_value'], '#attributes' => $element['#attributes'],
'#parents' => $element['#parents'], '#spawned' => TRUE);
      }
    }
  }
  return $element;
}

/**
 * Format a form item.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used:  title, value, description, required, error
 * @return
 *   A themed HTML string representing the form item.
 */
function theme_item($element) {
  return theme('form_element', $element, $element['#value'] .
$element['#children']);
}

/**
 * Format a checkbox.
```

```
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used:  title, value, return_value, description, required
 * @return
 *   A themed HTML string representing the checkbox.
 */
function theme_checkbox($element) {
  _form_set_class($element, array('form-checkbox'));
  $checkbox = '<input ';
  $checkbox .= 'type="checkbox" ';
  $checkbox .= 'name="'. $element['#name'] .'" ';
  $checkbox .= 'id="'. $element['#id'].'" ' ;
  $checkbox .= 'value="'. $element['#return_value'] .'" ';
  $checkbox .= $element['#value'] ? ' checked="checked" ' : ' ';
  $checkbox .= drupal_attributes($element['#attributes']) . ' />';

  if (!is_null($element['#title'])) {
    $checkbox = '<label class="option">'. $checkbox .' '. $element['#title']
.'</label>';
  }

  unset($element['#title']);
  return theme('form_element', $element, $checkbox);
}

/**
 * Format a set of checkboxes.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 * @return
 *   A themed HTML string representing the checkbox set.
 */
function theme_checkboxes($element) {
  $class = 'form-checkboxes';
  if (isset($element['#attributes']['class'])) {
    $class .= ' '. $element['#attributes']['class'];
  }
  $element['#children'] = '<div class="'. $class .'">'. $element['#children']
.'</div>';
  if ($element['#title'] || $element['#description']) {
    unset($element['#id']);
    return theme('form_element', $element, $element['#children']);
  }
  else {
    return $element['#children'];
  }
}

function expand_checkboxes($element) {
  $value = is_array($element['#value']) ? $element['#value'] : array();
  $element['#tree'] = TRUE;
  if (count($element['#options']) > 0) {
    if (!isset($element['#default_value']) || $element['#default_value'] == 0)
{
      $element['#default_value'] = array();
    }
    foreach ($element['#options'] as $key => $choice) {
      if (!isset($element[$key])) {
        $element[$key] = array('#type' => 'checkbox', '#processed' => TRUE,
'#title' => $choice, '#return_value' => $key, '#default_value' =>
isset($value[$key]), '#attributes' => $element['#attributes']);
```

```
      }
    }
  }
  return $element;
}

function theme_submit($element) {
  return theme('button', $element);
}

function theme_button($element) {
  // Make sure not to overwrite classes.
  if (isset($element['#attributes']['class'])) {
    $element['#attributes']['class'] = 'form-'. $element['#button_type'] .' '.
$element['#attributes']['class'];
  }
  else {
    $element['#attributes']['class'] = 'form-'. $element['#button_type'];
  }

  return '<input type="submit" '. (empty($element['#name']) ? '' : 'name="'.
$element['#name'] .'" ')  .'id="'. $element['#id'].'" value="'.
check_plain($element['#value']) .'" '.
drupal_attributes($element['#attributes']) ." />\n";
}

/**
 * Format a hidden form field.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used:  value, edit
 * @return
 *   A themed HTML string representing the hidden form field.
 */
function theme_hidden($element) {
  return '<input type="hidden" name="'. $element['#name'] . '" id="'.
$element['#id'] . '" value="'. check_plain($element['#value']) ."\" " .
drupal_attributes($element['#attributes']) ." />\n";
}

function theme_token($element) {
  return theme('hidden', $element);
}

/**
 * Format a textfield.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used:  title, value, description, size, maxlength, required,
attributes autocomplete_path
 * @return
 *   A themed HTML string representing the textfield.
 */
function theme_textfield($element) {
  $size = $element['#size'] ? ' size="' . $element['#size'] . '"' : '';
  $class = array('form-text');
  $extra = '';
  $output = '';

  if ($element['#autocomplete_path']) {
    drupal_add_js('misc/autocomplete.js');
```

```
    $class[] = 'form-autocomplete';
    $extra =  '<input class="autocomplete" type="hidden" id="'. $element['#id']
.'-autocomplete" value="'. check_url(url($element['#autocomplete_path'], NULL,
NULL, TRUE)) .'" disabled="disabled" />';
  }
  _form_set_class($element, $class);

  if (isset($element['#field_prefix'])) {
    $output .= '<span class="field-prefix">'. $element['#field_prefix']
.'</span> ';
  }

  $output .= '<input type="text" maxlength="'. $element['#maxlength'] .'"
name="'. $element['#name'] .'" id="'. $element['#id'] .'" '. $size .' value="'.
check_plain($element['#value']) .'"'.
drupal_attributes($element['#attributes']) .' />';

  if (isset($element['#field_suffix'])) {
    $output .= ' <span class="field-suffix">'. $element['#field_suffix']
.'</span>';
  }

  return theme('form_element', $element, $output). $extra;
}

/**
 * Format a form.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: action, method, attributes, children
 * @return
 *   A themed HTML string representing the form.
 */
function theme_form($element) {
  // Anonymous div to satisfy XHTML compliance.
  $action = $element['#action'] ? 'action="' . check_url($element['#action']) .
'" ' : '';
  return '<form '. $action . ' method="'. $element['#method'] .'" '. 'id="'.
$element['#id'] .'"'. drupal_attributes($element['#attributes']) .">\n<div>".
$element['#children'] ."\n</div></form>\n";
}

/**
 * Format a textarea.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: title, value, description, rows, cols, required,
attributes
 * @return
 *   A themed HTML string representing the textarea.
 */
function theme_textarea($element) {
  $class = array('form-textarea');
  if ($element['#resizable'] !== FALSE) {
    drupal_add_js('misc/textarea.js');
    $class[] = 'resizable';
  }

  $cols = $element['#cols'] ? ' cols="'. $element['#cols'] .'"' : '';
  _form_set_class($element, $class);
```

```
  return theme('form_element', $element, '<textarea'. $cols .' rows="'.
$element['#rows'] .'" name="'. $element['#name'] .'" id="'. $element['#id'] .'"
'. drupal_attributes($element['#attributes']) .'>'.
check_plain($element['#value']) .'</textarea>');
}

/**
 * Format HTML markup for use in forms.
 *
 * This is used in more advanced forms, such as theme selection and filter
format.
 *
 * @param $element
 *   An associative array containing the properties of the element.
 *   Properties used: value, children.
 * @return
 *   A themed HTML string representing the HTML markup.
 */

function theme_markup($element) {
  return (isset($element['#value']) ? $element['#value'] : '') .
(isset($element['#children']) ? $element['#children'] : '');
}

/**
* Format a password field.
*
* @param $element
*   An associative array containing the properties of the element.
*   Properties used:  title, value, description, size, maxlength, required,
attributes
* @return
*   A themed HTML string representing the form.
*/
function theme_password($element) {
  $size = $element['#size'] ? ' size="'. $element['#size'] .'" ' : '';
  $maxlength = $element['#maxlength'] ? ' maxlength="'. $element['#maxlength']
.'" ' : '';

  _form_set_class($element, array('form-text'));
  $output = '<input type="password" name="'. $element['#name'] .'" id="'.
$element['#id'] .'" '. $maxlength . $size .
drupal_attributes($element['#attributes']) .' />';
  return theme('form_element', $element, $output);
}

/**
 * Expand weight elements into selects.
 */
function process_weight($element) {
  for ($n = (-1 * $element['#delta']); $n <= $element['#delta']; $n++) {
    $weights[$n] = $n;
  }
  $element['#options'] = $weights;
  $element['#type'] = 'select';
  $element['#is_weight'] = TRUE;
  return $element;
}

/**
 * Format a file upload field.
 *
 * @param $title
```

```
 *   The label for the file upload field.
 * @param $name
 *   The internal name used to refer to the field.
 * @param $size
 *   A measure of the visible size of the field (passed directly to HTML).
 * @param $description
 *   Explanatory text to display after the form item.
 * @param $required
 *   Whether the user must upload a file to the field.
 * @return
 *   A themed HTML string representing the field.
 *
 * For assistance with handling the uploaded file correctly, see the API
 * provided by file.inc.
 */
function theme_file($element) {
  _form_set_class($element, array('form-file'));
  return theme('form_element', $element, '<input type="file" name="'.
$element['#name'] .'"'. ($element['#attributes'] ? ' '.
drupal_attributes($element['#attributes']) : '') .' id="'. $element['#id'] .'"
size="'. $element['#size'] ."\" />\n");
}

/**
 * Return a themed form element.
 *
 * @param element
 *   An associative array containing the properties of the element.
 *   Properties used: title, description, id, required
 * @param $value
 *   The form element's data.
 * @return
 *   A string representing the form element.
 */
function theme_form_element($element, $value) {
  $output  = '<div class="form-item">'."\n";
  $required = !empty($element['#required']) ? '<span class="form-required"
title="'. t('This field is required.') .'">*</span>' : '';

  if (!empty($element['#title'])) {
    $title = $element['#title'];
    if (!empty($element['#id'])) {
      $output .= ' <label for="'. $element['#id'] .'">'. t('!title: !required',
array('!title' => filter_xss_admin($title), '!required' => $required))
."</label>\n";
    }
    else {
      $output .= ' <label>'. t('!title: !required', array('!title' =>
filter_xss_admin($title), '!required' => $required)) ."</label>\n";
    }
  }

  $output .= " $value\n";

  if (!empty($element['#description'])) {
    $output .= ' <div class="description">'. $element['#description']
."</div>\n";
  }

  $output .= "</div>\n";

  return $output;
}
```

```
/**
 * Sets a form element's class attribute.
 *
 * Adds 'required' and 'error' classes as needed.
 *
 * @param &$element
 *   The form element.
 * @param $name
 *   Array of new class names to be added.
 */
function _form_set_class(&$element, $class = array()) {
  if ($element['#required']) {
    $class[] = 'required';
  }
  if (form_get_error($element)){
    $class[] = 'error';
  }
  if (isset($element['#attributes']['class'])) {
    $class[] = $element['#attributes']['class'];
  }
  $element['#attributes']['class'] = implode(' ', $class);
}

/**
 * Remove invalid characters from an HTML ID attribute string.
 *
 * @param $id
 *   The ID to clean.
 * @return
 *   The cleaned ID.
 */
function form_clean_id($id = NULL) {
  $id = str_replace(array('][', '_', ' '), '-', $id);
  return $id;
}

/**
 * @} End of "defgroup form".
 */
```

COMMON.INC

```
<?php
// $Id: common.inc,v 1.611 2007/01/10 23:30:07 unconed Exp $

/**
 * @file
 * Common functions that many Drupal modules will need to reference.
 *
 * The functions that are critical and need to be available even when serving
 * a cached page are instead located in bootstrap.inc.
 */

/**
 * Return status for saving which involved creating a new item.
 */
define('SAVED_NEW', 1);

/**
 * Return status for saving which involved an update to an existing item.
```

```
 */
define('SAVED_UPDATED', 2);

/**
 * Return status for saving which deleted an existing item.
 */
define('SAVED_DELETED', 3);

/**
 * Set content for a specified region.
 *
 * @param $region
 *   Page region the content is assigned to.
 *
 * @param $data
 *   Content to be set.
 */
function drupal_set_content($region = NULL, $data = NULL) {
  static $content = array();

  if (!is_null($region) && !is_null($data)) {
    $content[$region][] = $data;
  }
  return $content;
}

/**
 * Get assigned content.
 *
 * @param $region
 *   A specified region to fetch content for. If NULL, all regions will be
returned.
 *
 * @param $delimiter
 *   Content to be inserted between exploded array elements.
 */
function drupal_get_content($region = NULL, $delimiter = ' ') {
  $content = drupal_set_content();
  if (isset($region)) {
    if (isset($content[$region]) && is_array($content[$region])) {
      return implode($delimiter, $content[$region]);
    }
  }
  else {
    foreach (array_keys($content) as $region) {
      if (is_array($content[$region])) {
        $content[$region] = implode($delimiter, $content[$region]);
      }
    }
    return $content;
  }
}

/**
 * Set the breadcrumb trail for the current page.
 *
 * @param $breadcrumb
 *   Array of links, starting with "home" and proceeding up to but not
including
 *   the current page.
 */
function drupal_set_breadcrumb($breadcrumb = NULL) {
  static $stored_breadcrumb;
```

```
  if (!is_null($breadcrumb)) {
    $stored_breadcrumb = $breadcrumb;
  }
  return $stored_breadcrumb;
}

/**
 * Get the breadcrumb trail for the current page.
 */
function drupal_get_breadcrumb() {
  $breadcrumb = drupal_set_breadcrumb();

  if (is_null($breadcrumb)) {
    $breadcrumb = menu_get_active_breadcrumb();
  }

  return $breadcrumb;
}

/**
 * Add output to the head tag of the HTML page.
 * This function can be called as long the headers aren't sent.
 */
function drupal_set_html_head($data = NULL) {
  static $stored_head = '';

  if (!is_null($data)) {
    $stored_head .= $data ."\n";
  }
  return $stored_head;
}

/**
 * Retrieve output to be displayed in the head tag of the HTML page.
 */
function drupal_get_html_head() {
  $output = "<meta http-equiv=\"Content-Type\" content=\"text/html;
charset=utf-8\" />\n";
  return $output . drupal_set_html_head();
}

/**
 * Reset the static variable which holds the aliases mapped for this request.
 */
function drupal_clear_path_cache() {
  drupal_lookup_path('wipe');
}

/**
 * Set an HTTP response header for the current page.
 *
 * Note: when sending a Content-Type header, always include a 'charset' type
 * too. This is necessary to avoid security bugs (e.g. UTF-7 XSS).
 */
function drupal_set_header($header = NULL) {
  // We use an array to guarantee there are no leading or trailing delimiters.
  // Otherwise, header('') could get called when serving the page later, which
  // ends HTTP headers prematurely on some PHP versions.
  static $stored_headers = array();

  if (strlen($header)) {
    header($header);
```

```
    $stored_headers[] = $header;
  }
  return implode("\n", $stored_headers);
}

/**
 * Get the HTTP response headers for the current page.
 */
function drupal_get_headers() {
  return drupal_set_header();
}

/**
 * Add a feed URL for the current page.
 *
 * @param $url
 *   The url for the feed
 * @param $title
 *   The title of the feed
 */
function drupal_add_feed($url = NULL, $title = '') {
  static $stored_feed_links = array();

  if (!is_null($url)) {
    $stored_feed_links[$url] = theme('feed_icon', $url);

    drupal_add_link(array('rel' => 'alternate',
                          'type' => 'application/rss+xml',
                          'title' => $title,
                          'href' => $url));
  }
  return $stored_feed_links;
}

/**
 * Get the feed URLs for the current page.
 *
 * @param $delimiter
 *   The delimiter to split feeds by
 */
function drupal_get_feeds($delimiter = "\n") {
  $feeds = drupal_add_feed();
  return implode($feeds, $delimiter);
}

/**
 * @name HTTP handling
 * @{
 * Functions to properly handle HTTP responses.
 */

/**
 * Parse an array into a valid urlencoded query string.
 *
 * @param $query
 *   The array to be processed e.g. $_GET
 * @param $exclude
 *   The array filled with keys to be excluded. Use parent[child] to exclude
nested items.
 * @param $urlencode
 *   If TRUE, the keys and values are both urlencoded.
 * @param $parent
 *   Should not be passed, only used in recursive calls
```

```
 * @return
 *   urlencoded string which can be appended to/as the URL query string
 */
function drupal_query_string_encode($query, $exclude = array(), $parent = '') {
  $params = array();

  foreach ($query as $key => $value) {
    $key = drupal_urlencode($key);
    if ($parent) {
      $key = $parent .'['. $key .']';
    }

    if (in_array($key, $exclude)) {
      continue;
    }

    if (is_array($value)) {
      $params[] = drupal_query_string_encode($value, $exclude, $key);
    }
    else {
      $params[] = $key .'='. drupal_urlencode($value);
    }
  }

  return implode('&', $params);
}

/**
 * Prepare a destination query string for use in combination with
 * drupal_goto(). Used to direct the user back to the referring page
 * after completing a form. By default the current URL is returned.
 * If a destination exists in the previous request, that destination
 * is returned. As such, a destination can persist across multiple
 * pages.
 *
 * @see drupal_goto()
 */
function drupal_get_destination() {
  if (isset($_REQUEST['destination'])) {
    return 'destination='. urlencode($_REQUEST['destination']);
  }
  else {
    // Use $_GET here to retrieve the original path in source form.
    $path = isset($_GET['q']) ? $_GET['q'] : '';
    $query = drupal_query_string_encode($_GET, array('q'));
    if ($query != '') {
      $path .= '?'. $query;
    }
    return 'destination='. urlencode($path);
  }
}

/**
 * Send the user to a different Drupal page.
 *
 * This issues an on-site HTTP redirect. The function makes sure the redirected
 * URL is formatted correctly.
 *
 * Usually the redirected URL is constructed from this function's input
 * parameters. However you may override that behavior by setting a
 * <em>destination</em> in either the $_REQUEST-array (i.e. by using
 * the query string of an URI) or the $_REQUEST['edit']-array (i.e. by
 * using a hidden form field). This is used to direct the user back to
```

```
 * the proper page after completing a form. For example, after editing
 * a post on the 'admin/content/node'-page or after having logged on using the
 * 'user login'-block in a sidebar. The function drupal_get_destination()
 * can be used to help set the destination URL.
 *
 * It is advised to use drupal_goto() instead of PHP's header(), because
 * drupal_goto() will append the user's session ID to the URI when PHP is
 * compiled with "--enable-trans-sid".
 *
 * This function ends the request; use it rather than a print theme('page')
 * statement in your menu callback.
 *
 * @param $path
 *   A Drupal path or a full URL.
 * @param $query
 *   The query string component, if any.
 * @param $fragment
 *   The destination fragment identifier (named anchor).
 * @param $http_response_code
 *   Valid values for an actual "goto" as per RFC 2616 section 10.3 are:
 *   - 301 Moved Permanently (the recommended value for most redirects)
 *   - 302 Found (default in Drupal and PHP, sometimes used for spamming search
 *         engines)
 *   - 303 See Other
 *   - 304 Not Modified
 *   - 305 Use Proxy
 *   - 307 Temporary Redirect (an alternative to "503 Site Down for
Maintenance")
 *   Note: Other values are defined by RFC 2616, but are rarely used and poorly
 *         supported.
 * @see drupal_get_destination()
 */
function drupal_goto($path = '', $query = NULL, $fragment = NULL,
$http_response_code = 302) {
  if (isset($_REQUEST['destination'])) {
    extract(parse_url(urldecode($_REQUEST['destination'])));
  }
  else if (isset($_REQUEST['edit']['destination'])) {
    extract(parse_url(urldecode($_REQUEST['edit']['destination'])));
  }

  $url = url($path, $query, $fragment, TRUE);

  // Before the redirect, allow modules to react to the end of the page
request.
  module_invoke_all('exit', $url);

  header('Location: '. $url, TRUE, $http_response_code);

  // The "Location" header sends a REDIRECT status code to the http
  // daemon. In some cases this can go wrong, so we make sure none
  // of the code below the drupal_goto() call gets executed when we redirect.
  exit();
}

/**
 * Generates a site off-line message
 */
function drupal_site_offline() {
  drupal_set_header('HTTP/1.1 503 Service unavailable');
  drupal_set_title(t('Site off-line'));
  print theme('maintenance_page',
filter_xss_admin(variable_get('site_offline_message',
```

```
    t('@site is currently under maintenance. We should be back shortly. Thank
you for your patience.', array('@site' => variable_get('site_name',
'Drupal'))))));
}

/**
 * Generates a 404 error if the request can not be handled.
 */
function drupal_not_found() {
  drupal_set_header('HTTP/1.1 404 Not Found');

  watchdog('page not found', check_plain($_GET['q']), WATCHDOG_WARNING);

  // Keep old path for reference
  if (!isset($_REQUEST['destination'])) {
    $_REQUEST['destination'] = $_GET['q'];
  }

  $path = drupal_get_normal_path(variable_get('site_404', ''));
  if ($path && $path != $_GET['q']) {
    menu_set_active_item($path);
    $return = menu_execute_active_handler();
  }
  else {
    // Redirect to a non-existent menu item to make possible tabs disappear.
    menu_set_active_item('');
  }

  if (empty($return)) {
    drupal_set_title(t('Page not found'));
  }
  // To conserve CPU and bandwidth, omit the blocks
  print theme('page', $return, FALSE);
}

/**
 * Generates a 403 error if the request is not allowed.
 */
function drupal_access_denied() {
  drupal_set_header('HTTP/1.1 403 Forbidden');
  watchdog('access denied', check_plain($_GET['q']), WATCHDOG_WARNING);

// Keep old path for reference
  if (!isset($_REQUEST['destination'])) {
    $_REQUEST['destination'] = $_GET['q'];
  }

  $path = drupal_get_normal_path(variable_get('site_403', ''));
  if ($path && $path != $_GET['q']) {
    menu_set_active_item($path);
    $return = menu_execute_active_handler();
  }
  else {
    // Redirect to a non-existent menu item to make possible tabs disappear.
    menu_set_active_item('');
  }

  if (empty($return)) {
    drupal_set_title(t('Access denied'));
    $return = t('You are not authorized to access this page.');
  }
  print theme('page', $return);
}
```

```
/**
 * Perform an HTTP request.
 *
 * This is a flexible and powerful HTTP client implementation. Correctly
handles
 * GET, POST, PUT or any other HTTP requests. Handles redirects.
 *
 * @param $url
 *   A string containing a fully qualified URI.
 * @param $headers
 *   An array containing an HTTP header => value pair.
 * @param $method
 *   A string defining the HTTP request to use.
 * @param $data
 *   A string containing data to include in the request.
 * @param $retry
 *   An integer representing how many times to retry the request in case of a
 *   redirect.
 * @return
 *   An object containing the HTTP request headers, response code, headers,
 *   data, and redirect status.
 */
function drupal_http_request($url, $headers = array(), $method = 'GET', $data =
NULL, $retry = 3) {
  $result = new stdClass();

  // Parse the URL, and make sure we can handle the schema.
  $uri = parse_url($url);
  switch ($uri['scheme']) {
    case 'http':
      $port = isset($uri['port']) ? $uri['port'] : 80;
      $host = $uri['host'] . ($port != 80 ? ':'. $port : '');
      $fp = @fsockopen($uri['host'], $port, $errno, $errstr, 15);
      break;
    case 'https':
      // Note: Only works for PHP 4.3 compiled with OpenSSL.
      $port = isset($uri['port']) ? $uri['port'] : 443;
      $host = $uri['host'] . ($port != 443 ? ':'. $port : '');
      $fp = @fsockopen('ssl://'. $uri['host'], $port, $errno, $errstr, 20);
      break;
    default:
      $result->error = 'invalid schema '. $uri['scheme'];
      return $result;
  }

  // Make sure the socket opened properly.
  if (!$fp) {
    $result->error = trim($errno .' '. $errstr);
    return $result;
  }

  // Construct the path to act on.
  $path = isset($uri['path']) ? $uri['path'] : '/';
  if (isset($uri['query'])) {
    $path .= '?'. $uri['query'];
  }

  // Create HTTP request.
  $defaults = array(
    // RFC 2616: "non-standard ports MUST, default ports MAY be included".
    // We don't add the port to prevent from breaking rewrite rules checking
    // the host that do not take into account the port number.
```

```
    'Host' => "Host: $host",
    'User-Agent' => 'User-Agent: Drupal (+http://drupal.org/)',
    'Content-Length' => 'Content-Length: '. strlen($data)
  );

  foreach ($headers as $header => $value) {
    $defaults[$header] = $header .': '. $value;
  }

  $request = $method .' '. $path ." HTTP/1.0\r\n";
  $request .= implode("\r\n", $defaults);
  $request .= "\r\n\r\n";
  if ($data) {
    $request .= $data ."\r\n";
  }
  $result->request = $request;

  fwrite($fp, $request);

  // Fetch response.
  $response = '';
  while (!feof($fp) && $chunk = fread($fp, 1024)) {
    $response .= $chunk;
  }
  fclose($fp);

  // Parse response.
  list($split, $result->data) = explode("\r\n\r\n", $response, 2);
  $split = preg_split("/\r\n|\n|\r/", $split);

  list($protocol, $code, $text) = explode(' ', trim(array_shift($split)), 3);
  $result->headers = array();

  // Parse headers.
  while ($line = trim(array_shift($split))) {
    list($header, $value) = explode(':', $line, 2);
    if (isset($result->headers[$header]) && $header == 'Set-Cookie') {
      // RFC 2109: the Set-Cookie response header comprises the token Set-
      // Cookie:, followed by a comma-separated list of one or more cookies.
      $result->headers[$header] .= ','. trim($value);
    }
    else {
      $result->headers[$header] = trim($value);
    }
  }

  $responses = array(
    100 => 'Continue', 101 => 'Switching Protocols',
    200 => 'OK', 201 => 'Created', 202 => 'Accepted', 203 => 'Non-Authoritative
Information', 204 => 'No Content', 205 => 'Reset Content', 206 => 'Partial
Content',
    300 => 'Multiple Choices', 301 => 'Moved Permanently', 302 => 'Found', 303
=> 'See Other', 304 => 'Not Modified', 305 => 'Use Proxy', 307 => 'Temporary
Redirect',
    400 => 'Bad Request', 401 => 'Unauthorized', 402 => 'Payment Required', 403
=> 'Forbidden', 404 => 'Not Found', 405 => 'Method Not Allowed', 406 => 'Not
Acceptable', 407 => 'Proxy Authentication Required', 408 => 'Request Time-out',
409 => 'Conflict', 410 => 'Gone', 411 => 'Length Required', 412 =>
'Precondition Failed', 413 => 'Request Entity Too Large', 414 => 'Request-URI
Too Large', 415 => 'Unsupported Media Type', 416 => 'Requested range not
satisfiable', 417 => 'Expectation Failed',
```

```
    500 => 'Internal Server Error', 501 => 'Not Implemented', 502 => 'Bad
Gateway', 503 => 'Service Unavailable', 504 => 'Gateway Time-out', 505 => 'HTTP
Version not supported'
  );
  // RFC 2616 states that all unknown HTTP codes must be treated the same as
  // the base code in their class.
  if (!isset($responses[$code])) {
    $code = floor($code / 100) * 100;
  }

  switch ($code) {
    case 200: // OK
    case 304: // Not modified
      break;
    case 301: // Moved permanently
    case 302: // Moved temporarily
    case 307: // Moved temporarily
      $location = $result->headers['Location'];

      if ($retry) {
        $result = drupal_http_request($result->headers['Location'], $headers,
$method, $data, --$retry);
        $result->redirect_code = $result->code;
      }
      $result->redirect_url = $location;

      break;
    default:
      $result->error = $text;
  }

  $result->code = $code;
  return $result;
}
/**
 * @} End of "HTTP handling".
 */

/**
 * Log errors as defined by administrator
 * Error levels:
 *  0 = Log errors to database.
 *  1 = Log errors to database and to screen.
 */
function error_handler($errno, $message, $filename, $line) {
  // If the @ error suppression operator was used, error_reporting is
temporarily set to 0
  if (error_reporting() == 0) {
    return;
  }

  if ($errno & (E_ALL ^ E_NOTICE)) {
    $types = array(1 => 'error', 2 => 'warning', 4 => 'parse error', 8 =>
'notice', 16 => 'core error', 32 => 'core warning', 64 => 'compile error', 128
=> 'compile warning', 256 => 'user error', 512 => 'user warning', 1024 => 'user
notice', 2048 => 'strict warning');
    $entry = $types[$errno] .': '. $message .' in '. $filename .' on line '.
$line .'.';

    // Force display of error messages in update.php
    if (variable_get('error_level', 1) == 1 || strstr($_SERVER['PHP_SELF'],
'update.php')) {
      drupal_set_message($entry, 'error');
```

```
    }

    watchdog('php', t('%message in %file on line %line.', array('%error' =>
$types[$errno], '%message' => $message, '%file' => $filename, '%line' =>
$line)), WATCHDOG_ERROR);
  }
}

function _fix_gpc_magic(&$item) {
  if (is_array($item)) {
    array_walk($item, '_fix_gpc_magic');
  }
  else {
    $item = stripslashes($item);
  }
}

/**
 * Helper function to strip slashes from $_FILES skipping over the tmp_name
keys
 * since PHP generates single backslashes for file paths on Windows systems.
 *
 * tmp_name does not have backslashes added see
 * http://php.net/manual/en/features.file-upload.php#42280
 */
function _fix_gpc_magic_files(&$item, $key) {
  if ($key != 'tmp_name') {
    if (is_array($item)) {
      array_walk($item, '_fix_gpc_magic_files');
    }
    else {
      $item = stripslashes($item);
    }
  }
}

/**
 * Correct double-escaping problems caused by "magic quotes" in some PHP
 * installations.
 */
function fix_gpc_magic() {
  static $fixed = FALSE;
  if (!$fixed && ini_get('magic_quotes_gpc')) {
    array_walk($_GET, '_fix_gpc_magic');
    array_walk($_POST, '_fix_gpc_magic');
    array_walk($_COOKIE, '_fix_gpc_magic');
    array_walk($_REQUEST, '_fix_gpc_magic');
    array_walk($_FILES, '_fix_gpc_magic_files');
    $fixed = TRUE;
  }
}

/**
 * Initialize the localization system.
 */
function locale_initialize() {
  global $user;

  if (function_exists('i18n_get_lang')) {
    return i18n_get_lang();
  }

  if (function_exists('locale')) {
```

```
    $languages = locale_supported_languages();
    $languages = $languages['name'];
  }
  else {
    // Ensure the locale/language is correctly returned, even without
locale.module.
    // Useful for e.g. XML/HTML 'lang' attributes.
    $languages = array('en' => 'English');
  }
  if ($user->uid && isset($languages[$user->language])) {
    return $user->language;
  }
  else {
    return key($languages);
  }
}

/**
 * Translate strings to the current locale.
 *
 * All human-readable text that will be displayed somewhere within a page
should be
 * run through the t() function.
 *
 * Examples:
 * @code
 *   if (!$info || !$info['extension']) {
 *     form_set_error('picture_upload', t('The uploaded file was not an
image.'));
 *   }
 *
 *   $form['submit'] = array(
 *     '#type' => 'submit',
 *     '#value' => t('Log in'),
 *   );
 * @endcode
 *
 * Any text within t() can be extracted by translators and changed into
 * the equivalent text in their native language.
 *
 * Special variables called "placeholders" are used to signal dynamic
 * information in a string which should not be translated. Placeholders
 * can also be used for text that may change from time to time
 * (such as link paths) to be changed without requiring updates to
translations.
 *
 * For example:
 * @code
 *   $output = t('There are currently %members and %visitors online.', array(
 *     '%members' => format_plural($total_users, '1 user', '@count users'),
 *     '%visitors' => format_plural($guests->count, '1 guest', '@count
guests')));
 * @endcode
 *
 * There are three styles of placeholders:
 * - !variable, which indicates that the text should be inserted as-is. This is
 *   useful for inserting variables into things like e-mail.
 *   @code
 *     $message[] = t("If you don't want to receive such e-mails, you can
change your settings at !url.", array('!url' => url("user/$account->uid", NULL,
NULL, TRUE)));
 *   @endcode
 *
```

```
 * - @variable, which indicates that the text should be run through
check_plain,
 *   to strip out HTML characters. Use this for any output that's displayed
within
 *   a Drupal page.
 *   @code
 *     drupal_set_title($title = t("@name's blog", array('@name' => $account-
>name)));
 *   @endcode
 *
 * - %variable, which indicates that the string should be highlighted with
 *   theme_placeholder() which shows up by default as <em>emphasized</em>.
 *   @code
 *     watchdog('mail', t('%name-from sent %name-to an e-mail.', array('%name-
from' => $user->name, '%name-to' => $account->name)));
 *   @endcode
 *
 * When using t(), try to put entire sentences and strings in one t() call.
 * This makes it easier for translators, as it provides context as to what
 * each word refers to. HTML markup within translation strings is allowed,
 * but should be avoided if possible. The exception is embedded links; link
 * titles add additional context for translators so should be kept in the main
 * string.
 *
 * Here is an example of an incorrect use if t():
 * @code
 *   $output .= t('<p>Go to the @contact-page.</p>', array('@contact-page' =>
l(t('contact page'), 'contact')));
 * @endcode
 *
 * Here is an example of t() used correctly:
 * @code
 *   $output .= '<p>'. t('Go to the <a href="@contact-page">contact page</a>.',
array('@contact-page' => url('contact'))) .'</p>';
 * @endcode
 *
 * Also avoid escaping quotation marks wherever possible.
 *
 * Incorrect:
 * @code
 *   $output .= t('Don\'t click me.');
 * @endcode
 *
 * Correct:
 * @code
 *   $output .= t("Don't click me.");
 * @endcode
 *
 * @param $string
 *   A string containing the English string to translate.
 * @param $args
 *   An associative array of replacements to make after translation. Incidences
 *   of any key in this array are replaced with the corresponding value.
 *   Based on the first character of the key, the value is escaped and/or
themed:
 *    - !variable: inserted as is
 *    - @variable: escape plain text to HTML (check_plain)
 *    - %variable: escape text and theme as a placeholder for user-submitted
 *      content (check_plain + theme_placeholder)
 * @return
 *   The translated string.
 */
function t($string, $args = 0) {
```

```
  global $locale;
  if (function_exists('locale') && $locale != 'en') {
    $string = locale($string);
  }
  if (!$args) {
    return $string;
  }
  else {
    // Transform arguments before inserting them
    foreach ($args as $key => $value) {
      switch ($key[0]) {
        // Escaped only
        case '@':
          $args[$key] = check_plain($value);
        break;
        // Escaped and placeholder
        case '%':
        default:
          $args[$key] = theme('placeholder', $value);
          break;
        // Pass-through
        case '!':
      }
    }
    return strtr($string, $args);
  }
}

/**
 * @defgroup validation Input validation
 * @{
 * Functions to validate user input.
 */

/**
 * Verify the syntax of the given e-mail address.
 *
 * Empty e-mail addresses are allowed. See RFC 2822 for details.
 *
 * @param $mail
 *   A string containing an e-mail address.
 * @return
 *   TRUE if the address is in a valid format.
 */
function valid_email_address($mail) {
  $user = '[a-zA-Z0-9_\-\.\+\^!#\$%&*+\/\=\?\`\|\{\}~\']+';
  $domain = '(?:(?:[a-zA-Z0-9]|[a-zA-Z0-9][a-zA-Z0-9\-]*[a-zA-Z0-9])\.?)+';
  $ipv4 = '[0-9]{1,3}(\.[0-9]{1,3}){3}';
  $ipv6 = '[0-9a-fA-F]{1,4}(\:[0-9a-fA-F]{1,4}){7}';
  $ipv7 = '[0-9a-zA-Z]{2,4}';

/*   return preg_match("/^$user@($domain|(\[($ipv4|$ipv6)\]))$/", $mail); */
  return preg_match("/^$user@$domain\.($ipv7)$/", $mail);

}

/**
 * Verify the syntax of the given URL.
 *
 * This function should only be used on actual URLs. It should not be used for
 * Drupal menu paths, which can contain arbitrary characters.
 *
 * @param $url
```

```
 *   The URL to verify.
 * @param $absolute
 *   Whether the URL is absolute (beginning with a scheme such as "http:").
 * @return
 *   TRUE if the URL is in a valid format.
 */
function valid_url($url, $absolute = FALSE) {
  $allowed_characters = '[a-z0-9\/:_\-_\.\?\$,;~=#&%\+]';
  if ($absolute) {
    return preg_match("/^(http|https|ftp):\/\/". $allowed_characters ."+$/i",
$url);
  }
  else {
    return preg_match("/^". $allowed_characters ."+$/i", $url);
  }
}

/**
 * Register an event for the current visitor (hostname/IP) to the flood control
mechanism.
 *
 * @param $name
 *   The name of the event.
 */
function flood_register_event($name) {
  db_query("INSERT INTO {flood} (event, hostname, timestamp) VALUES ('%s',
'%s', %d)", $name, $_SERVER['REMOTE_ADDR'], time());
}

/**
 * Check if the current visitor (hostname/IP) is allowed to proceed with the
specified event.
 * The user is allowed to proceed if he did not trigger the specified event
more than
 * $threshold times per hour.
 *
 * @param $name
 *   The name of the event.
 * @param $number
 *   The maximum number of the specified event per hour (per visitor).
 * @return
 *   True if the user did not exceed the hourly threshold. False otherwise.
 */
function flood_is_allowed($name, $threshold) {
  $number = db_num_rows(db_query("SELECT event FROM {flood} WHERE event = '%s'
AND hostname = '%s' AND timestamp > %d", $name, $_SERVER['REMOTE_ADDR'], time()
- 3600));
  return ($number < $threshold ? TRUE : FALSE);
}

function check_file($filename) {
  return is_uploaded_file($filename);
}

/**
 * Prepare a URL for use in an HTML attribute. Strips harmful protocols.
 *
 */
function check_url($uri) {
  return filter_xss_bad_protocol($uri, FALSE);
}

/**
```

```
 * @defgroup format Formatting
 * @{
 * Functions to format numbers, strings, dates, etc.
 */

/**
 * Formats an RSS channel.
 *
 * Arbitrary elements may be added using the $args associative array.
 */
function format_rss_channel($title, $link, $description, $items, $language =
'en', $args = array()) {
  // arbitrary elements may be added using the $args associative array

  $output = "<channel>\n";
  $output .= ' <title>'. check_plain($title) ."</title>\n";
  $output .= ' <link>'. check_url($link) ."</link>\n";

  // The RSS 2.0 "spec" doesn't indicate HTML can be used in the description.
  // We strip all HTML tags, but need to prevent double encoding from properly
  // escaped source data (such as &amp becoming &amp;amp;).
  $output .= ' <description>'.
check_plain(decode_entities(strip_tags($description))) ."</description>\n";
  $output .= ' <language>'. check_plain($language) ."</language>\n";
  $output .= format_xml_elements($args);
  $output .= $items;
  $output .= "</channel>\n";

  return $output;
}

/**
 * Format a single RSS item.
 *
 * Arbitrary elements may be added using the $args associative array.
 */
function format_rss_item($title, $link, $description, $args = array()) {
  $output = "<item>\n";
  $output .= ' <title>'. check_plain($title) ."</title>\n";
  $output .= ' <link>'. check_url($link) ."</link>\n";
  $output .= ' <description>'. check_plain($description) ."</description>\n";
  $output .= format_xml_elements($args);
  $output .= "</item>\n";

  return $output;
}

/**
 * Format XML elements.
 *
 * @param $array
 *   An array where each item represent an element and is either a:
 *   - (key => value) pair (<key>value</key>)
 *   - Associative array with fields:
 *     - 'key': element name
 *     - 'value': element contents
 *     - 'attributes': associative array of element attributes
 *
 * In both cases, 'value' can be a simple string, or it can be another array
 * with the same format as $array itself for nesting.
 */
function format_xml_elements($array) {
  foreach ($array as $key => $value) {
```

```
    if (is_numeric($key)) {
      if ($value['key']) {
        $output .= ' <'. $value['key'];
        if (isset($value['attributes']) && is_array($value['attributes'])) {
          $output .= drupal_attributes($value['attributes']);
        }

        if ($value['value'] != '') {
          $output .= '>'. (is_array($value['value']) ?
format_xml_elements($value['value']) : check_plain($value['value'])) .'</'.
$value['key'] .">\n";
        }
        else {
          $output .= " />\n";
        }
      }
    }
    else {
      $output .= ' <'. $key .'>'. (is_array($value) ?
format_xml_elements($value) : check_plain($value)) ."</$key>\n";
    }
  }
  return $output;
}

/**
 * Format a string containing a count of items.
 *
 * This function ensures that the string is pluralized correctly. Since t() is
 * called by this function, make sure not to pass already-localized strings to
it.
 *
 * @param $count
 *   The item count to display.
 * @param $singular
 *   The string for the singular case. Please make sure it is clear this is
 *   singular, to ease translation (e.g. use "1 new comment" instead of "1
new").
 * @param $plural
 *   The string for the plural case. Please make sure it is clear this is
plural,
 *   to ease translation. Use @count in place of the item count, as in "@count
 *   new comments".
 * @return
 *   A translated string.
 */
function format_plural($count, $singular, $plural) {
  if ($count == 1) return t($singular, array("@count" => $count));

  // get the plural index through the gettext formula
  $index = (function_exists('locale_get_plural')) ? locale_get_plural($count) :
-1;
  if ($index < 0) { // backward compatibility
    return t($plural, array("@count" => $count));
  }
  else {
    switch ($index) {
      case "0":
        return t($singular, array("@count" => $count));
      case "1":
        return t($plural, array("@count" => $count));
      default:
```

```
        return t(strtr($plural, array("@count" => '@count['. $index .']')),
array('@count['. $index .']' => $count));
    }
  }
}

/**
 * Parse a given byte count.
 *
 * @param $size
 *   The size expressed as a number of bytes with optional SI size and unit
 *   suffix (e.g. 2, 3K, 5MB, 10G).
 * @return
 *   An integer representation of the size.
 */
function parse_size($size) {
  $suffixes = array(
    '' => 1,
    'k' => 1024,
    'm' => 1048576, // 1024 * 1024
    'g' => 1073741824, // 1024 * 1024 * 1024
  );
  if (preg_match('/([0-9]+)\s*(k|m|g)?(b?(ytes?)?)/i', $size, $match)) {
    return $match[1] * $suffixes[drupal_strtolower($match[2])];
  }
}

/**
 * Generate a string representation for the given byte count.
 *
 * @param $size
 *   The size in bytes.
 * @return
 *   A translated string representation of the size.
 */
function format_size($size) {
  if ($size < 1024) {
    return format_plural($size, '1 byte', '@count bytes');
  }
  else {
    $size = round($size / 1024, 2);
    $suffix = t('KB');
    if ($size >= 1024) {
      $size = round($size / 1024, 2);
      $suffix = t('MB');
    }
    return t('@size @suffix', array('@size' => $size, '@suffix' => $suffix));
  }
}

/**
 * Format a time interval with the requested granularity.
 *
 * @param $timestamp
 *   The length of the interval in seconds.
 * @param $granularity
 *   How many different units to display in the string.
 * @return
 *   A translated string representation of the interval.
 */
function format_interval($timestamp, $granularity = 2) {
```

```
  $units = array('1 year|@count years' => 31536000, '1 week|@count weeks' =>
604800, '1 day|@count days' => 86400, '1 hour|@count hours' => 3600, '1
min|@count min' => 60, '1 sec|@count sec' => 1);
  $output = '';
  foreach ($units as $key => $value) {
    $key = explode('|', $key);
    if ($timestamp >= $value) {
      $output .= ($output ? ' ' : '') . format_plural(floor($timestamp /
$value), $key[0], $key[1]);
      $timestamp %= $value;
      $granularity--;
    }

    if ($granularity == 0) {
      break;
    }
  }
  return $output ? $output : t('0 sec');
}

/**
 * Format a date with the given configured format or a custom format string.
 *
 * Drupal allows administrators to select formatting strings for 'small',
 * 'medium' and 'large' date formats. This function can handle these formats,
 * as well as any custom format.
 *
 * @param $timestamp
 *   The exact date to format, as a UNIX timestamp.
 * @param $type
 *   The format to use. Can be "small", "medium" or "large" for the
preconfigured
 *   date formats. If "custom" is specified, then $format is required as well.
 * @param $format
 *   A PHP date format string as required by date(). A backslash should be used
 *   before a character to avoid interpreting the character as part of a date
 *   format.
 * @param $timezone
 *   Time zone offset in seconds; if omitted, the user's time zone is used.
 * @return
 *   A translated date string in the requested format.
 */
function format_date($timestamp, $type = 'medium', $format = '', $timezone =
NULL) {
  if (!isset($timezone)) {
    global $user;
    if (variable_get('configurable_timezones', 1) && $user->uid &&
strlen($user->timezone)) {
      $timezone = $user->timezone;
    }
    else {
      $timezone = variable_get('date_default_timezone', 0);
    }
  }

  $timestamp += $timezone;

  switch ($type) {
    case 'small':
      $format = variable_get('date_format_short', 'm/d/Y - H:i');
      break;
    case 'large':
      $format = variable_get('date_format_long', 'l, F j, Y - H:i');
```

```
      break;
    case 'custom':
      // No change to format
      break;
    case 'medium':
    default:
      $format = variable_get('date_format_medium', 'D, m/d/Y - H:i');
  }

  $max = strlen($format);
  $date = '';
  for ($i = 0; $i < $max; $i++) {
    $c = $format[$i];
    if (strpos('AaDFlM', $c) !== FALSE) {
      $date .= t(gmdate($c, $timestamp));
    }
    else if (strpos('BdgGhHiIjLmnsStTUwWYyz', $c) !== FALSE) {
      $date .= gmdate($c, $timestamp);
    }
    else if ($c == 'r') {
      $date .= format_date($timestamp - $timezone, 'custom', 'D, d M Y H:i:s
O', $timezone);
    }
    else if ($c == 'O') {
      $date .= sprintf('%s%02d%02d', ($timezone < 0 ? '-' : '+'), abs($timezone
/ 3600), abs($timezone % 3600) / 60);
    }
    else if ($c == 'Z') {
      $date .= $timezone;
    }
    else if ($c == '\\') {
      $date .= $format[++$i];
    }
    else {
      $date .= $c;
    }
  }

  return $date;
}

/**
 * @} End of "defgroup format".
 */

/**
 * Generate a URL from a Drupal menu path. Will also pass-through existing
URLs.
 *
 * @param $path
 *   The Drupal path being linked to, such as "admin/content/node", or an
existing URL
 *   like "http://drupal.org/".
 * @param $query
 *   A query string to append to the link or URL.
 * @param $fragment
 *   A fragment identifier (named anchor) to append to the link. If an existing
 *   URL with a fragment identifier is used, it will be replaced. Note, do not
 *   include the '#'.
 * @param $absolute
 *   Whether to force the output to be an absolute link (beginning with http:).
 *   Useful for links that will be displayed outside the site, such as in an
 *   RSS feed.
```

```
 * @return
 *   a string containing a URL to the given path.
 *
 * When creating links in modules, consider whether l() could be a better
 * alternative than url().
 */
function url($path = NULL, $query = NULL, $fragment = NULL, $absolute = FALSE)
{
  if (isset($fragment)) {
    $fragment = '#'. $fragment;

  }

  // Return an external link if $path contains an allowed absolute URL.
  // Only call the slow filter_xss_bad_protocol if $path contains a ':' before
any / ? or #.
  $colonpos = strpos($path, ':');
  if ($colonpos !== FALSE && !preg_match('![/?#]!', substr($path, 0,
$colonpos)) && filter_xss_bad_protocol($path, FALSE) == check_plain($path)) {
    // Split off the fragment
    if (strpos($path, '#') !== FALSE) {
      list($path, $old_fragment) = explode('#', $path, 2);
      if (isset($old_fragment) && !isset($fragment)) {
        $fragment = '#'. $old_fragment;
      }
    }
    // Append the query
    if (isset($query)) {
      $path .= (strpos($path, '?') !== FALSE ? '&' : '?') . $query;
    }
    // Reassemble
    return $path . $fragment;
  }

  global $base_url;
  static $script;
  static $clean_url;

  if (empty($script)) {
    // On some web servers, such as IIS, we can't omit "index.php". So, we
    // generate "index.php?q=foo" instead of "?q=foo" on anything that is not
    // Apache.
    $script = (strpos($_SERVER['SERVER_SOFTWARE'], 'Apache') === FALSE) ?
'index.php' : '';
  }

  // Cache the clean_url variable to improve performance.
  if (!isset($clean_url)) {
    $clean_url = (bool)variable_get('clean_url', '0');
  }

  $base = ($absolute ? $base_url . '/' : base_path());

  // The special path '<front>' links to the default front page.
  if (!empty($path) && $path != '<front>') {
    $path = drupal_get_path_alias($path);
    $path = drupal_urlencode($path);
    if (!$clean_url) {
      if (isset($query)) {
        return $base . $script .'?q='. $path .'&'. $query . $fragment;
      }
      else {
        return $base . $script .'?q='. $path . $fragment;
```

```
      }
    }
    else {
      if (isset($query)) {
        return $base . $path .'?'. $query . $fragment;
      }
      else {
        return $base . $path . $fragment;
      }
    }
  }
  else {
    if (isset($query)) {
      return $base . $script .'?'. $query . $fragment;
    }
    else {
      return $base . $fragment;
    }
  }
}

/**
 * Format an attribute string to insert in a tag.
 *
 * @param $attributes
 *   An associative array of HTML attributes.
 * @return
 *   An HTML string ready for insertion in a tag.
 */
function drupal_attributes($attributes = array()) {
  if (is_array($attributes)) {
    $t = '';
    foreach ($attributes as $key => $value) {
      $t .= " $key=".'"'. check_plain($value) .'"';
    }
    return $t;
  }
}

/**
 * Format an internal Drupal link.
 *
 * This function correctly handles aliased paths, and allows themes to
highlight
 * links to the current page correctly, so all internal links output by modules
 * should be generated by this function if possible.
 *
 * @param $text
 *   The text to be enclosed with the anchor tag.
 * @param $path
 *   The Drupal path being linked to, such as "admin/content/node". Can be an
external
 *   or internal URL.
 *     - If you provide the full URL, it will be considered an
 *   external URL.
 *     - If you provide only the path (e.g. "admin/content/node"), it is
considered an
 *   internal link. In this case, it must be a system URL as the url() function
 *   will generate the alias.
 * @param $attributes
 *   An associative array of HTML attributes to apply to the anchor tag.
 * @param $query
 *   A query string to append to the link.
```

```
 * @param $fragment
 *   A fragment identifier (named anchor) to append to the link.
 * @param $absolute
 *   Whether to force the output to be an absolute link (beginning with http:).
 *   Useful for links that will be displayed outside the site, such as in an
RSS
 *   feed.
 * @param $html
 *   Whether the title is HTML, or just plain-text. For example for making an
 *   image a link, this must be set to TRUE, or else you will see the encoded
 *   HTML.
 * @return
 *   an HTML string containing a link to the given path.
 */
function l($text, $path, $attributes = array(), $query = NULL, $fragment =
NULL, $absolute = FALSE, $html = FALSE) {
  if ($path == $_GET['q']) {
    if (isset($attributes['class'])) {
      $attributes['class'] .= ' active';
    }
    else {
      $attributes['class'] = 'active';
    }
  }
  return '<a href="'. check_url(url($path, $query, $fragment, $absolute)) .'"'.
drupal_attributes($attributes) .'>'. ($html ? $text : check_plain($text))
.'</a>';
}

/**
 * Perform end-of-request tasks.
 *
 * This function sets the page cache if appropriate, and allows modules to
 * react to the closing of the page by calling hook_exit().
 */
function drupal_page_footer() {
  if (variable_get('cache', 0)) {
    page_set_cache();
  }

  module_invoke_all('exit');
}

/**
 * Form an associative array from a linear array.
 *
 * This function walks through the provided array and constructs an associative
 * array out of it. The keys of the resulting array will be the values of the
 * input array. The values will be the same as the keys unless a function is
 * specified, in which case the output of the function is used for the values
 * instead.
 *
 * @param $array
 *   A linear array.
 * @param $function
 *   The name of a function to apply to all values before output.
 * @result
 *   An associative array.
 */
function drupal_map_assoc($array, $function = NULL) {
  if (!isset($function)) {
    $result = array();
    foreach ($array as $value) {
```

```
      $result[$value] = $value;
    }
    return $result;
  }
  elseif (function_exists($function)) {
    $result = array();
    foreach ($array as $value) {
      $result[$value] = $function($value);
    }
    return $result;
  }
}

/**
 * Evaluate a string of PHP code.
 *
 * This is a wrapper around PHP's eval(). It uses output buffering to capture
both
 * returned and printed text. Unlike eval(), we require code to be surrounded
by
 * <?php ?> tags; in other words, we evaluate the code as if it were a stand-
alone
 * PHP file.
 *
 * Using this wrapper also ensures that the PHP code which is evaluated can not
 * overwrite any variables in the calling code, unlike a regular eval() call.
 *
 * @param $code
 *   The code to evaluate.
 * @return
 *   A string containing the printed output of the code, followed by the
returned
 *   output of the code.
 */
function drupal_eval($code) {
  ob_start();
  print eval('?>'. $code);
  $output = ob_get_contents();
  ob_end_clean();
  return $output;
}

/**
 * Returns the path to a system item (module, theme, etc.).
 *
 * @param $type
 *   The type of the item (i.e. theme, theme_engine, module).
 * @param $name
 *   The name of the item for which the path is requested.
 *
 * @return
 *   The path to the requested item.
 */
function drupal_get_path($type, $name) {
  return dirname(drupal_get_filename($type, $name));
}

/**
 * Returns the base URL path of the Drupal installation.
 * At the very least, this will always default to /.
 */
function base_path() {
  return $GLOBALS['base_path'];
```

```
}

/**
 * Provide a substitute clone() function for PHP4.
 */
function drupal_clone($object) {
  return version_compare(phpversion(), '5.0') < 0 ? $object : clone($object);
}

/**
 * Add a <link> tag to the page's HEAD.
 */
function drupal_add_link($attributes) {
  drupal_set_html_head('<link'. drupal_attributes($attributes) ." />\n");
}

/**
 * Adds a CSS file to the stylesheet queue.
 *
 * @param $path
 *   (optional) The path to the CSS file relative to the base_path(), e.g.,
 *   /modules/devel/devel.css.
 * @param $type
 *   (optional) The type of stylesheet that is being added. Types are: module
 *   or theme.
 * @param $media
 *   (optional) The media type for the stylesheet, e.g., all, print, screen.
 * @param $preprocess
 *   (optional) Should this CSS file be aggregated and compressed if this
 *   feature has been turned on under the performance section?
 *
 *   What does this actually mean?
 *   CSS preprocessing is the process of aggregating a bunch of separate CSS
 *   files into one file that is then compressed by removing all extraneous
 *   white space.
 *
 *   The reason for merging the CSS files is outlined quite thoroughly here:
 *   http://www.die.net/musings/page_load_time/
 *   "Load fewer external objects. Due to request overhead, one bigger file
 *   just loads faster than two smaller ones half its size."
 *
 *   However, you should *not* preprocess every file as this can lead to
 *   redundant caches. You should set $preprocess = FALSE when:
 *
 *     - Your styles are only used rarely on the site. This could be a special
 *       admin page, the homepage, or a handful of pages that does not
represent
 *       the majority of the pages on your site.
 *
 *   Typical candidates for caching are for example styles for nodes across
 *   the site, or used in the theme.
 * @return
 *   An array of CSS files.
 */
function drupal_add_css($path = NULL, $type = 'module', $media = 'all',
$preprocess = TRUE) {
  static $css = array();

  // Create an array of CSS files for each media type first, since each type
needs to be served
  // to the browser differently.
  if (isset($path)) {
```

```
    // This check is necessary to ensure proper cascading of styles and is
faster than an asort().
    if (!isset($css[$media])) {
      $css[$media] = array('module' => array(), 'theme' => array());
    }
    $css[$media][$type][$path] = $preprocess;
  }

  return $css;
}

/**
 * Returns a themed representation of all stylesheets that should be attached
to the page.
 * It loads the CSS in order, with 'core' CSS first, then 'module' CSS, then
'theme' CSS files.
 * This ensures proper cascading of styles for easy overriding in modules and
themes.
 *
 * @param $css
 *   (optional) An array of CSS files. If no array is provided, the default
stylesheets array is used instead.
 * @return
 *   A string of XHTML CSS tags.
 */
function drupal_get_css($css = NULL) {
  $output = '';
  if (!isset($css)) {
    $css = drupal_add_css();
  }

  $preprocess_css = variable_get('preprocess_css', FALSE);
  $directory = file_directory_path();
  $is_writable = is_dir($directory) && is_writable($directory) &&
(variable_get('file_downloads', FILE_DOWNLOADS_PUBLIC) ==
FILE_DOWNLOADS_PUBLIC);

  foreach ($css as $media => $types) {
    // If CSS preprocessing is off, we still need to output the styles.
    // Additionally, go through any remaining styles if CSS preprocessing is on
and output the non-cached ones.
    foreach ($types as $type => $files) {
      foreach ($types[$type] as $file => $preprocess) {
        if (!$preprocess || !($is_writable && $preprocess_css)) {
          // If a CSS file is not to be preprocessed and it's a module CSS
file, it needs to *always* appear at the *top*,
          // regardless of whether preprocessing is on or off.
          if (!$preprocess && $type == 'module') {
            $no_module_preprocess .= '<style type="text/css" media="'. $media
.'">@import "'. base_path() . $file .'";</style>' ."\n";
          }
          // If a CSS file is not to be preprocessed and it's a theme CSS file,
it needs to *always* appear at the *bottom*,
          // regardless of whether preprocessing is on or off.
          else if (!$preprocess && $type == 'theme') {
            $no_theme_preprocess .= '<style type="text/css" media="'. $media
.'">@import "'. base_path() . $file .'";</style>' ."\n";
          }
          else {
            $output .= '<style type="text/css" media="'. $media .'">@import "'.
base_path() . $file .'";</style>' ."\n";
          }
        }
```

```
      }
    }

    if ($is_writable && $preprocess_css) {
      $filename = md5(serialize($types)) .'.css';
      $preprocess_file = drupal_build_css_cache($types, $filename);
      $output .= '<style type="text/css" media="'. $media .'">@import "'.
base_path() . $preprocess_file .'";</style>'. "\n";
    }
  }

  return $no_module_preprocess . $output . $no_theme_preprocess;
}

/**
 * Aggregate and optimize CSS files, putting them in the files directory.
 *
 * @param $types
 *   An array of types of CSS files (e.g., screen, print) to aggregate and
compress into one file.
 * @param $filename
 *   The name of the aggregate CSS file.
 * @return
 *   The name of the CSS file.
 */
function drupal_build_css_cache($types, $filename) {
  $data = '';

  // Create the css/ within the files folder.
  $csspath = file_create_path('css');
  file_check_directory($csspath, FILE_CREATE_DIRECTORY);

  if (!file_exists($csspath .'/'. $filename)) {
    // Build aggregate CSS file.
    foreach ($types as $type) {
      foreach ($type as $file => $cache) {
        if ($cache) {
          $contents = file_get_contents($file);
          // Return the path to where this CSS file originated from, stripping
off the name of the file at the end of the path.
          $path = base_path() . substr($file, 0, strrpos($file, '/')) .'/';
          // Wraps all @import arguments in url().
          $contents = preg_replace('/@import\s+(?!url)[\'"]?(\S*)\b[\'"]?/i',
'@import url("\1")', $contents);
          // Fix all paths within this CSS file, ignoring absolute paths.
          $data .= preg_replace('/url\(([\'"]?)(?![a-z]+:)/i', 'url(\1'. $path
. '\2', $contents);
        }
      }
    }

    // @import rules must proceed any other style, so we move those to the top.
    $regexp = '/@import[^;]+;/i';
    preg_match_all($regexp, $data, $matches);
    $data = preg_replace($regexp, '', $data);
    $data = implode('', $matches[0]) . $data;

    // Perform some safe CSS optimizations.
    $data = preg_replace('<
      \s*([@{}:;,]|\)\s|\s\()\s* |  # Remove whitespace around separators, but
keep space around parentheses.
      /\*([^*\\\]|\*(?!/))+\*/ |   # Remove comments that are not CSS hacks.
      [\n\r]                       # Remove line breaks.
```

```
      >x', '\1', $data);

    // Create the CSS file.
    file_save_data($data, $csspath .'/'. $filename, FILE_EXISTS_REPLACE);
  }
  return $csspath .'/'. $filename;
}

/**
 * Delete all cached CSS files.
 */
function drupal_clear_css_cache() {
  file_scan_directory(file_create_path('css'), '.*', array('.', '..', 'CVS'),
'file_delete', TRUE);
}

/**
 * Add a JavaScript file, setting or inline code to the page.
 *
 * The behavior of this function depends on the parameters it is called with.
 * Generally, it handles the addition of JavaScript to the page, either as
 * reference to an existing file or as inline code. The following actions can
be
 * performed using this function:
 *
 * - Add a file ('core', 'module' and 'theme'):
 *   Adds a reference to a JavaScript file to the page. JavaScript files
 *   are placed in a certain order, from 'core' first, to 'module' and finally
 *   'theme' so that files, that are added later, can override previously added
 *   files with ease.
 *
 * - Add inline JavaScript code ('inline'):
 *   Executes a piece of JavaScript code on the current page by placing the
code
 *   directly in the page. This can, for example, be useful to tell the user
that
 *   a new message arrived, by opening a pop up, alert box etc.
 *
 * - Add settings ('setting'):
 *   Adds a setting to Drupal's global storage of JavaScript settings. Per-page
 *   settings are required by some modules to function properly. The settings
 *   will be accessible at Drupal.settings.
 *
 * @param $data
 *   (optional) If given, the value depends on the $type parameter:
 *   - 'core', 'module' or 'theme': Path to the file relative to base_path().
 *   - 'inline': The JavaScript code that should be placed in the given scope.
 *   - 'setting': An array with configuration options as associative array. The
 *       array is directly placed in Drupal.settings. You might want to wrap
your
 *       actual configuration settings in another variable to prevent the
pollution
 *       of the Drupal.settings namespace.
 * @param $type
 *   (optional) The type of JavaScript that should be added to the page.
Allowed
 *   values are 'core', 'module', 'theme', 'inline' and 'setting'. You
 *   can, however, specify any value. It is treated as a reference to a
JavaScript
 *   file. Defaults to 'module'.
 * @param $scope
 *   (optional) The location in which you want to place the script. Possible
 *   values are 'header' and 'footer' by default. If your theme implements
```

```
 *   different locations, however, you can also use these.
 * @param $defer
 *   (optional) If set to TRUE, the defer attribute is set on the <script> tag.
 *   Defaults to FALSE. This parameter is not used with $type == 'setting'.
 * @param $cache
 *   (optional) If set to FALSE, the JavaScript file is loaded anew on every
page
 *   call, that means, it is not cached. Defaults to TRUE. Used only when $type
 *   references a JavaScript file.
 * @return
 *   If the first parameter is NULL, the JavaScript array that has been built
so
 *   far for $scope is returned.
 */
function drupal_add_js($data = NULL, $type = 'module', $scope = 'header',
$defer = FALSE, $cache = TRUE) {
  if (!is_null($data)) {
    _drupal_add_js('misc/jquery.js', 'core', 'header', FALSE, $cache);
    _drupal_add_js('misc/drupal.js', 'core', 'header', FALSE, $cache);
  }
  return _drupal_add_js($data, $type, $scope, $defer, $cache);
}

/**
 * Helper function for drupal_add_js().
 */
function _drupal_add_js($data, $type, $scope, $defer, $cache) {
  static $javascript = array();

  if (!isset($javascript[$scope])) {
    $javascript[$scope] = array('core' => array(), 'module' => array(), 'theme'
=> array(), 'setting' => array(), 'inline' => array());
  }

  if (!isset($javascript[$scope][$type])) {
    $javascript[$scope][$type] = array();
  }

  if (!is_null($data)) {
    switch ($type) {
      case 'setting':
        $javascript[$scope][$type][] = $data;
        break;
      case 'inline':
        $javascript[$scope][$type][] = array('code' => $data, 'defer' =>
$defer);
        break;
      default:
        $javascript[$scope][$type][$data] = array('cache' => $cache, 'defer' =>
$defer);
    }
  }

  return $javascript[$scope];
}

/**
 * Returns a themed presentation of all JavaScript code for the current page.
 * References to JavaScript files are placed in a certain order: first, all
 * 'core' files, then all 'module' and finally all 'theme' JavaScript files
 * are added to the page. Then, all settings are output, followed by 'inline'
 * JavaScript code.
 *
```

```
 * @parameter $scope
 *   (optional) The scope for which the JavaScript rules should be returned.
 *   Defaults to 'header'.
 * @parameter $javascript
 *   (optional) An array with all JavaScript code. Defaults to the default
 *   JavaScript array for the given scope.
 * @return
 *   All JavaScript code segments and includes for the scope as HTML tags.
 */
function drupal_get_js($scope = 'header', $javascript = NULL) {
  $output = '';
  if (is_null($javascript)) {
    $javascript = drupal_add_js(NULL, NULL, $scope);
  }

  foreach ($javascript as $type => $data) {
    if (!$data) continue;

    switch ($type) {
      case 'setting':
        $output .= '<script type="text/javascript">Drupal.extend({ settings: '.
drupal_to_js(call_user_func_array('array_merge_recursive', $data)) ."
});</script>\n";
        break;
      case 'inline':
        foreach ($data as $info) {
          $output .= '<script type="text/javascript"'. ($info['defer'] ? '
defer="defer"' : '') .'>'. $info['code'] ."</script>\n";
        }
        break;
      default:
        foreach ($data as $path => $info) {
          $output .= '<script type="text/javascript"'. ($info['defer'] ? '
defer="defer"' : '') .' src="'. check_url(base_path() . $path) .
($info['cache'] ? '' : '?'. time()) ."\"></script>\n";
        }
    }
  }

  return $output;
}

/**
 * Converts a PHP variable into its Javascript equivalent.
 *
 * We use HTML-safe strings, i.e. with <, > and & escaped.
 */
function drupal_to_js($var) {
  switch (gettype($var)) {
    case 'boolean':
      return $var ? 'true' : 'false'; // Lowercase necessary!
    case 'integer':
    case 'double':
      return $var;
    case 'resource':
    case 'string':
      return '"'. str_replace(array("\r", "\n", "<", ">", "&"),
                              array('\r', '\n', '\x3c', '\x3e', '\x26'),
                              addslashes($var)) .'"';
    case 'array':
      if (array_keys($var) === range(0, sizeof($var) - 1)) {
        $output = array();
        foreach ($var as $v) {
```

```
          $output[] = drupal_to_js($v);
        }
        return '[ '. implode(', ', $output) .' ]';
      }
      // Fall through
    case 'object':
      $output = array();
      foreach ($var as $k => $v) {
        $output[] = drupal_to_js(strval($k)) .': '. drupal_to_js($v);
      }
      return '{ '. implode(', ', $output) .' }';
    default:
      return 'null';
  }
}

/**
 * Wrapper around urlencode() which avoids Apache quirks.
 *
 * Should be used when placing arbitrary data in an URL. Note that Drupal paths
 * are urlencoded() when passed through url() and do not require urlencoding()
 * of individual components.
 *
 * Notes:
 * - For esthetic reasons, we do not escape slashes. This also avoids a
'feature'
 *   in Apache where it 404s on any path containing '%2F'.
 * - mod_rewrite's unescapes %-encoded ampersands and hashes when clean URLs
 *   are used, which are interpreted as delimiters by PHP. These characters are
 *   double escaped so PHP will still see the encoded version.
 *
 * @param $text
 *   String to encode
 */
function drupal_urlencode($text) {
  if (variable_get('clean_url', '0')) {
    return str_replace(array('%2F', '%26', '%23'),
                       array('/', '%2526', '%2523'),
                       urlencode($text));
  }
  else {
    return str_replace('%2F', '/', urlencode($text));
  }
}

/**
 * Ensure the private key variable used to generate tokens is set.
 *
 * @return
 *   The private key
 */
function drupal_get_private_key() {
  if (!($key = variable_get('drupal_private_key', 0))) {
    $key = md5(uniqid(mt_rand(), true)) . md5(uniqid(mt_rand(), true));
    variable_set('drupal_private_key', $key);
  }
  return $key;
}

/**
 * Generate a token based on $value, the current user session and private key.
 *
 * @param $value
```

```
 *   An additional value to base the token on
 */
function drupal_get_token($value = '') {
  $private_key = drupal_get_private_key();
  return md5(session_id() . $value . $private_key);
}

/**
 * Validate a token based on $value, the current user session and private key.
 *
 * @param $token
 *   The token to be validated.
 * @param $value
 *   An additional value to base the token on.
 * @param $skip_anonymous
 *   Set to true to skip token validation for anonymous users.
 * @return
 *   True for a valid token, false for an invalid token. When $skip_anonymous
is true, the return value will always be true for anonymous users.
 */
function drupal_valid_token($token, $value = '', $skip_anonymous = FALSE) {
  global $user;
  return (($skip_anonymous && $user->uid == 0) || ($token == md5(session_id() .
$value . variable_get('drupal_private_key', ''))));
}

/**
 * Performs one or more XML-RPC request(s).
 *
 * @param $url
 *   An absolute URL of the XML-RPC endpoint.
 *     Example:
 *     http://www.example.com/xmlrpc.php
 * @param ...
 *   For one request:
 *     The method name followed by a variable number of arguments to the
method.
 *   For multiple requests (system.multicall):
 *     An array of call arrays. Each call array follows the pattern of the
single
 *     request: method name followed by the arguments to the method.
 * @return
 *   For one request:
 *     Either the return value of the method on success, or FALSE.
 *     If FALSE is returned, see xmlrpc_errno() and xmlrpc_error_msg().
 *   For multiple requests:
 *     An array of results. Each result will either be the result
 *     returned by the method called, or an xmlrpc_error object if the call
 *     failed. See xmlrpc_error().
 */
function xmlrpc($url) {
  require_once './includes/xmlrpc.inc';
  $args = func_get_args();
  return call_user_func_array('_xmlrpc', $args);
}

function _drupal_bootstrap_full() {
  static $called;
  global $locale;

  if ($called) {
    return;
  }
```

```
  $called = 1;
  require_once './includes/theme.inc';
  require_once './includes/pager.inc';
  require_once './includes/menu.inc';
  require_once './includes/tablesort.inc';
  require_once './includes/file.inc';
  require_once './includes/unicode.inc';
  require_once './includes/image.inc';
  require_once './includes/form.inc';
  // Set the Drupal custom error handler.
  set_error_handler('error_handler');
  // Emit the correct charset HTTP header.
  drupal_set_header('Content-Type: text/html; charset=utf-8');
  // Detect string handling method
  unicode_check();
  // Undo magic quotes
  fix_gpc_magic();
  // Load all enabled modules
  module_load_all();
  // Initialize the localization system.  Depends on i18n.module being loaded
already.
  $locale = locale_initialize();
  // Let all modules take action before menu system handles the reqest
  module_invoke_all('init');

}

/**
 * Store the current page in the cache.
 *
 * We try to store a gzipped version of the cache. This requires the
 * PHP zlib extension (http://php.net/manual/en/ref.zlib.php).
 * Presence of the extension is checked by testing for the function
 * gzencode. There are two compression algorithms: gzip and deflate.
 * The majority of all modern browsers support gzip or both of them.
 * We thus only deal with the gzip variant and unzip the cache in case
 * the browser does not accept gzip encoding.
 *
 * @see drupal_page_header
 */
function page_set_cache() {
  global $user, $base_root;

  if (!$user->uid && $_SERVER['REQUEST_METHOD'] == 'GET' &&
count(drupal_get_messages(NULL, FALSE)) == 0) {
    // This will fail in some cases, see page_get_cache() for the explanation.
    if ($data = ob_get_contents()) {
      $cache = TRUE;
      if (function_exists('gzencode')) {
        // We do not store the data in case the zlib mode is deflate.
        // This should be rarely happening.
        if (zlib_get_coding_type() == 'deflate') {
          $cache = FALSE;
        }
        else if (zlib_get_coding_type() == FALSE) {
          $data = gzencode($data, 9, FORCE_GZIP);
        }
        // The remaining case is 'gzip' which means the data is
        // already compressed and nothing left to do but to store it.
      }
      ob_end_flush();
      if ($cache && $data) {
```

```
        cache_set($base_root . request_uri(), 'cache_page', $data,
CACHE_TEMPORARY, drupal_get_headers());
      }
    }
  }
}

/**
 * Send an e-mail message, using Drupal variables and default settings.
 * More information in the <a
href="http://php.net/manual/en/function.mail.php">
 * PHP function reference for mail()</a>
 * @param $mailkey
 *   A key to identify the mail sent, for altering.
 * @param $to
 *   The mail address or addresses where the message will be send to. The
 *   formatting of this string must comply with RFC 2822. Some examples are:
 *    user@example.com
 *    user@example.com, anotheruser@example.com
 *    User <user@example.com>
 *    User <user@example.com>, Another User <anotheruser@example.com>
 * @param $subject
 *   Subject of the e-mail to be sent. This must not contain any newline
 *   characters, or the mail may not be sent properly.
 * @param $body
 *   Message to be sent. Drupal will format the correct line endings for you.
 * @param $from
 *   Sets From, Reply-To, Return-Path and Error-To to this value, if given.
 * @param $headers
 *   Associative array containing the headers to add. This is typically
 *   used to add extra headers (From, Cc, and Bcc).
 *   <em>When sending mail, the mail must contain a From header.</em>
 * @return Returns TRUE if the mail was successfully accepted for delivery,
 *   FALSE otherwise.
 */
function drupal_mail($mailkey, $to, $subject, $body, $from = NULL, $headers =
array()) {
  $defaults = array(
    'MIME-Version' => '1.0',
    'Content-Type' => 'text/plain; charset=UTF-8; format=flowed',
    'Content-Transfer-Encoding' => '8Bit',
    'X-Mailer' => 'Drupal'
  );
  if (isset($from)) {
    $defaults['From'] = $defaults['Reply-To'] = $defaults['Return-Path'] =
$defaults['Errors-To'] = $from;
  }
  $headers = array_merge($defaults, $headers);
  // Custom hook traversal to allow pass by reference
  foreach (module_implements('mail_alter') AS $module) {
    $function = $module .'_mail_alter';
    $function($mailkey, $to, $subject, $body, $from, $headers);
  }
  // Allow for custom mail backend
  if (variable_get('smtp_library', '') &&
file_exists(variable_get('smtp_library', ''))) {
    include_once './' . variable_get('smtp_library', '');
    return drupal_mail_wrapper($mailkey, $to, $subject, $body, $from,
$headers);
  }
  else {
    /*
    ** Note: if you are having problems with sending mail, or mails look wrong
```

```
    ** when they are received you may have to modify the str_replace to suit
    ** your systems.
    **  - \r\n will work under dos and windows.
    **  - \n will work for linux, unix and BSDs.
    **  - \r will work for macs.
    **
    ** According to RFC 2646, it's quite rude to not wrap your e-mails:
    **
    ** "The Text/Plain media type is the lowest common denominator of
    ** Internet e-mail, with lines of no more than 997 characters (by
    ** convention usually no more than 80), and where the CRLF sequence
    ** represents a line break [MIME-IMT]."
    **
    ** CRLF === \r\n
    **
    ** http://www.rfc-editor.org/rfc/rfc2646.txt
    **
    */
    $mimeheaders = array();
    foreach ($headers as $name => $value) {
      $mimeheaders[] = $name .': '. mime_header_encode($value);
    }
    return mail(
      $to,
      mime_header_encode($subject),
      str_replace("\r", '', $body),
      join("\n", $mimeheaders)
    );
  }
}

/**
 * Executes a cron run when called
 * @return
 * Returns TRUE if ran successfully
 */
function drupal_cron_run() {
  // If not in 'safe mode', increase the maximum execution time:
  if (!ini_get('safe_mode')) {
    set_time_limit(240);
  }

  // Fetch the cron semaphore
  $semaphore = variable_get('cron_semaphore', FALSE);

  if ($semaphore) {
    if (time() - $semaphore > 3600) {
      // Either cron has been running for more than an hour or the semaphore
      // was not reset due to a database error.
      watchdog('cron', t('Cron has been running for more than an hour and is
most likely stuck.'), WATCHDOG_ERROR);

      // Release cron semaphore
      variable_del('cron_semaphore');
    }
    else {
      // Cron is still running normally.
      watchdog('cron', t('Attempting to re-run cron while it is already
running.'), WATCHDOG_WARNING);
    }
  }
  else {
    // Register shutdown callback
```

```
    register_shutdown_function('drupal_cron_cleanup');

    // Lock cron semaphore
    variable_set('cron_semaphore', time());

    // Iterate through the modules calling their cron handlers (if any):
    module_invoke_all('cron');

    // Record cron time
    variable_set('cron_last', time());
    watchdog('cron', t('Cron run completed.'), WATCHDOG_NOTICE);

    // Release cron semaphore
    variable_del('cron_semaphore');

    // Return TRUE so other functions can check if it did run successfully
    return TRUE;
  }
}

/**
 * Shutdown function for cron cleanup.
 */
function drupal_cron_cleanup() {
  // See if the semaphore is still locked.
  if (variable_get('cron_semaphore', FALSE)) {
    watchdog('cron', t('Cron run exceeded the time limit and was aborted.'),
WATCHDOG_WARNING);

    // Release cron semaphore
    variable_del('cron_semaphore');
  }
}

/**
 * Returns an array of files objects of the given type from the site-wide
 * directory (i.e. modules/), the all-sites directory (i.e.
 * sites/all/modules/), the profiles directory, and site-specific directory
 * (i.e. sites/somesite/modules/). The returned array will be keyed using the
 * key specified (name, basename, filename). Using name or basename will cause
 * site-specific files to be prioritized over similar files in the default
 * directories. That is, if a file with the same name appears in both the
 * site-wide directory and site-specific directory, only the site-specific
 * version will be included.
 *
 * @param $mask
 *   The regular expression of the files to find.
 * @param $directory
 *   The subdirectory name in which the files are found. For example,
 *   'modules' will search in both modules/ and
 *   sites/somesite/modules/.
 * @param $key
 *   The key to be passed to file_scan_directory().
 * @param $min_depth
 *   Minimum depth of directories to return files from.
 *
 * @return
 *   An array of file objects of the specified type.
 */
function drupal_system_listing($mask, $directory, $key = 'name', $min_depth =
1) {
  global $profile;
  $config = conf_path();
```

```
  // When this function is called during Drupal's initial installation process,
  // the name of the profile that's about to be installed is stored in the
global
  // $profile variable. At all other times, the standard Drupal systems
variable
  // table contains the name of the current profile, and we can call
variable_get()
  // to determine what one is active.
  if (!isset($profile)) {
    $profile = variable_get('install_profile', 'default');
  }
  $searchdir = array($directory);
  $files = array();

  // Always search sites/all/* as well as the global directories
  $searchdir[] = 'sites/all/'. $directory;

  // The 'profiles' directory contains pristine collections of modules and
  // themes as organized by a distribution.  It is pristine in the same way
  // that /modules is pristine for core; users should avoid changing anything
  // there in favor of sites/all or sites/<domain> directories.
  if (file_exists("profiles/$profile/$directory")) {
    $searchdir[] = "profiles/$profile/$directory";
  }

  if (file_exists("$config/$directory")) {
    $searchdir[] = "$config/$directory";
  }

  // Get current list of items
  foreach ($searchdir as $dir) {
    $files = array_merge($files, file_scan_directory($dir, $mask, array('.',
'..', 'CVS'), 0, TRUE, $key, $min_depth));
  }

  return $files;
}

/**
 * Renders HTML given a structured array tree. Recursively iterates over each
 * of the array elements, generating HTML code. This function is usually
 * called from within a another function, like drupal_get_form() or
node_view().
 *
 * @param $elements
 *   The structured array describing the data to be rendered.
 * @return
 *   The rendered HTML.
 */
function drupal_render(&$elements) {
  if (!isset($elements) || (isset($elements['#access']) &&
!$elements['#access'])) {
    return NULL;
  }

  $content = '';
  // Either the elements did not go through form_builder or one of the children
  // has a #weight.
  if (!isset($elements['#sorted'])) {
    uasort($elements, "_element_sort");
  }
  if (!isset($elements['#children'])) {
```

```
    $children = element_children($elements);
    /* Render all the children that use a theme function */
    if (isset($elements['#theme']) && empty($elements['#theme_used'])) {
      $elements['#theme_used'] = TRUE;

      $previous = array();
      foreach (array('#value', '#type', '#prefix', '#suffix') as $key) {
        $previous[$key] = isset($elements[$key]) ? $elements[$key] : NULL;
      }
      // If we rendered a single element, then we will skip the renderer.
      if (empty($children)) {
        $elements['#printed'] = TRUE;
      }
      else {
        $elements['#value'] = '';
      }
      $elements['#type'] = 'markup';

      unset($elements['#prefix'], $elements['#suffix']);
      $content = theme($elements['#theme'], $elements);

      foreach (array('#value', '#type', '#prefix', '#suffix') as $key) {
        $elements[$key] = isset($previous[$key]) ? $previous[$key] : NULL;
      }
    }
    /* render each of the children using drupal_render and concatenate them */
    if (!isset($content) || $content === '') {
      foreach ($children as $key) {
        $content .= drupal_render($elements[$key]);
      }
    }
  }
  if (isset($content) && $content !== '') {
    $elements['#children'] = $content;
  }

  // Until now, we rendered the children, here we render the element itself
  if (!isset($elements['#printed'])) {
    $content = theme(!empty($elements['#type']) ? $elements['#type'] :
'markup', $elements);
    $elements['#printed'] = TRUE;
  }

  if (isset($content) && $content !== '') {
    $prefix = isset($elements['#prefix']) ? $elements['#prefix'] : '';
    $suffix = isset($elements['#suffix']) ? $elements['#suffix'] : '';
    return $prefix . $content . $suffix;
  }
}

/**
 * Function used by uasort in drupal_render() to sort structured arrays
 * by weight.
 */
function _element_sort($a, $b) {
  $a_weight = (is_array($a) && isset($a['#weight'])) ? $a['#weight'] : 0;
  $b_weight = (is_array($b) && isset($b['#weight'])) ? $b['#weight'] : 0;
  if ($a_weight == $b_weight) {
    return 0;
  }
  return ($a_weight < $b_weight) ? -1 : 1;
}
```

```
/**
 * Check if the key is a property.
 */
function element_property($key) {
  return $key[0] == '#';
}

/**
 * Get properties of a structured array element. Properties begin with '#'.
 */
function element_properties($element) {
  return array_filter(array_keys((array) $element), 'element_property');
}

/**
 * Check if the key is a child.
 */
function element_child($key) {
  return $key[0] != '#';
}

/**
 * Get keys of a structured array tree element that are not properties
 * (i.e., do not begin with '#').
 */
function element_children($element) {
  return array_filter(array_keys((array) $element), 'element_child');
}
```

USER.MODULE

```
<?php
// $Id: user.module,v 1.745.2.1 2007/01/29 19:08:46 dries Exp $

/**
 * @file
 * Enables the user registration and login system.
 */

define('USERNAME_MAX_LENGTH', 60);
define('EMAIL_MAX_LENGTH', 64);

/**
 * Invokes hook_user() in every module.
 *
 * We cannot use module_invoke() for this, because the arguments need to
 * be passed by reference.
 */
function user_module_invoke($type, &$array, &$user, $category = NULL) {
  foreach (module_list() as $module) {
    $function = $module .'_user';
    if (function_exists($function)) {
      $function($type, $array, $user, $category);
    }
  }
}

function user_external_load($authname) {
  $result = db_query("SELECT uid FROM {authmap} WHERE authname = '%s'",
$authname);
```

```
  if ($user = db_fetch_array($result)) {
    return user_load($user);
  }
  else {
    return 0;
  }
}

/**
 * Fetch a user object.
 *
 * @param $array
 *   An associative array of attributes to search for in selecting the
 *   user, such as user name or e-mail address.
 *
 * @return
 *   A fully-loaded $user object upon successful user load or FALSE if user
cannot be loaded.
 */
function user_load($array = array()) {
  // Dynamically compose a SQL query:
  $query = array();
  $params = array();

  foreach ($array as $key => $value) {
    if ($key == 'uid' || $key == 'status') {
      $query[] = "$key = %d";
      $params[] = $value;
    }
    else if ($key == 'pass') {
      $query[] = "pass = '%s'";
      $params[] = md5($value);
    }
    else {
      $query[]= "LOWER($key) = LOWER('%s')";
      $params[] = $value;
    }
  }
  $result = db_query('SELECT * FROM {users} u WHERE '. implode(' AND ',
$query), $params);

  if (db_num_rows($result)) {
    $user = db_fetch_object($result);
    $user = drupal_unpack($user);

    $user->roles = array();
    if ($user->uid) {
      $user->roles[DRUPAL_AUTHENTICATED_RID] = 'authenticated user';
    }
    else {
      $user->roles[DRUPAL_ANONYMOUS_RID] = 'anonymous user';
    }
    $result = db_query('SELECT r.rid, r.name FROM {role} r INNER JOIN
{users_roles} ur ON ur.rid = r.rid WHERE ur.uid = %d', $user->uid);
    while ($role = db_fetch_object($result)) {
      $user->roles[$role->rid] = $role->name;
    }
    user_module_invoke('load', $array, $user);
  }
  else {
    $user = FALSE;
  }
```

```
  return $user;
}

/**
 * Save changes to a user account or add a new user.
 *
 * @param $account
 *   The $user object for the user to modify or add. If $user->uid is
 *   omitted, a new user will be added.
 *
 * @param $array
 *   An array of fields and values to save. For example array('name' => 'My
name');
 *   Setting a field to NULL deletes it from the data column.
 *
 * @param $category
 *   (optional) The category for storing profile information in.
 */
function user_save($account, $array = array(), $category = 'account') {
  // Dynamically compose a SQL query:
  $user_fields = user_fields();
  if ($account->uid) {
    user_module_invoke('update', $array, $account, $category);

    $data = unserialize(db_result(db_query('SELECT data FROM {users} WHERE uid
= %d', $account->uid)));
    foreach ($array as $key => $value) {
      if ($key == 'pass' && !empty($value)) {
        $query .= "$key = '%s', ";
        $v[] = md5($value);
      }
      else if ((substr($key, 0, 4) !== 'auth') && ($key != 'pass')) {
        if (in_array($key, $user_fields)) {
          // Save standard fields
          $query .= "$key = '%s', ";
          $v[] = $value;
        }
        else if ($key != 'roles') {
          // Roles is a special case: it used below.
          if ($value === NULL) {
            unset($data[$key]);
          }
          else {
             /*

                    savprof
                    SAVING PROFILE HERE


             */
             if($key == 'profile_full_name') {
                    $chuckTemp = $data[$key];
             }
             else if($key == 'profile_grad_year') {
                    $chuckTemp = $data[$key];
             }
            $data[$key] = $value;
          }
        }
      }
    }
    $query .= "data = '%s' ";
    $v[] = serialize($data);
```

```
    db_query("UPDATE {users} SET $query WHERE uid = %d", array_merge($v,
array($account->uid)));

    // Reload user roles if provided
    if (is_array($array['roles'])) {
      db_query('DELETE FROM {users_roles} WHERE uid = %d', $account->uid);

      foreach (array_keys($array['roles']) as $rid) {
        if (!in_array($rid, array(DRUPAL_ANONYMOUS_RID,
DRUPAL_AUTHENTICATED_RID))) {
          db_query('INSERT INTO {users_roles} (uid, rid) VALUES (%d, %d)',
$account->uid, $rid);
        }
      }
    }

    // Delete a blocked user's sessions to kick them if they are online.
    if (isset($array['status']) && $array['status'] == 0) {
      sess_destroy_uid($account->uid);
    }

    // Refresh user object
    $user = user_load(array('uid' => $account->uid));
    user_module_invoke('after_update', $array, $user, $category);
  }
  else {
    $array['uid'] = db_next_id('{users}_uid');

    if (!isset($array['created'])) {    // Allow 'created' to be set by
hook_auth
      $array['created'] = time();
    }

    // Note, we wait with saving the data column to prevent module-handled
    // fields from being saved there. We cannot invoke hook_user('insert') here
    // because we don't have a fully initialized user object yet.
    foreach ($array as $key => $value) {
      switch ($key) {
        case 'pass':
          $fields[] = $key;
          $values[] = md5($value);
          $s[] = "'%s'";
          break;
        case 'uid':        case 'mode':     case 'sort':
        case 'threshold':  case 'created':  case 'access':
        case 'login':      case 'status':
          $fields[] = $key;
          $values[] = $value;
          $s[] = "%d";
          break;
        default:
          if (substr($key, 0, 4) !== 'auth' && in_array($key, $user_fields)) {
            $fields[] = $key;
            $values[] = $value;
            $s[] = "'%s'";
          }
          break;
      }
    }
    db_query('INSERT INTO {users} (' . implode(', ', $fields) .') VALUES ('.
implode(', ', $s) .')', $values);
```

```
    // Build the initial user object.
    $user = user_load(array('uid' => $array['uid']));

    user_module_invoke('insert', $array, $user, $category);

    // Build and save the serialized data field now
    $data = array();
    foreach ($array as $key => $value) {
      if ((substr($key, 0, 4) !== 'auth') && ($key != 'roles') &&
(!in_array($key, $user_fields)) && ($value !== NULL)) {
        $data[$key] = $value;
      }
    }
    db_query("UPDATE {users} SET data = '%s' WHERE uid = %d", serialize($data),
$user->uid);

    // Save user roles (delete just to be safe).
    if (is_array($array['roles'])) {
      db_query('DELETE FROM {users_roles} WHERE uid = %d', $array['uid']);
      foreach (array_keys($array['roles']) as $rid) {
        if (!in_array($rid, array(DRUPAL_ANONYMOUS_RID,
DRUPAL_AUTHENTICATED_RID))) {
          db_query('INSERT INTO {users_roles} (uid, rid) VALUES (%d, %d)',
$array['uid'], $rid);
        }
      }
    }

    // Build the finished user object.
    $user = user_load(array('uid' => $array['uid']));
  }

  // Save distributed authentication mappings
  $authmaps = array();
  foreach ($array as $key => $value) {
    if (substr($key, 0, 4) == 'auth') {
      $authmaps[$key] = $value;
    }
  }
  if (sizeof($authmaps) > 0) {
    user_set_authmaps($user, $authmaps);
  }

  return $user;
}

/**
 * Verify the syntax of the given name.
 */
function user_validate_name($name) {
  if (!strlen($name)) return t('You must enter a username.');
  if (substr($name, 0, 1) == ' ') return t('The username cannot begin with a
space.');
  if (substr($name, -1) == ' ') return t('The username cannot end with a
space.');
  if (strpos($name, '  ') !== FALSE) return t('The username cannot contain
multiple spaces in a row.');
  if (ereg("[^\x80-\xF7 [:alnum:]@_.-]", $name)) return t('The username
contains an illegal character.');
  if (preg_match('/[\x{80}-\x{A0}'.          // Non-printable ISO-8859-1 + NBSP
                 '\x{AD}'.                   // Soft-hyphen
                 '\x{2000}-\x{200F}'.        // Various space characters
                 '\x{2028}-\x{202F}'.        // Bidirectional text overrides
```

```
                  '\x{205F}-\x{206F}'.      // Various text hinting characters
                  '\x{FEFF}'.               // Byte order mark
                  '\x{FF01}-\x{FF60}'.      // Full-width latin
                  '\x{FFF9}-\x{FFFD}'.      // Replacement characters
                  '\x{0}]/u',               // NULL byte
                  $name)) {
    return t('The username contains an illegal character.');
  }
  if (strpos($name, '@') !== FALSE && !eregi('@([0-9a-z](-?[0-9a-z])*.)+[a-
z]{2}([zmuvtg]|fo|me)?$', $name)) return t('The username is not a valid
authentication ID.');
  if (strlen($name) > USERNAME_MAX_LENGTH) return t('The username %name is too
long: it must be %max characters or less.', array('%name' => $name, '%max' =>
USERNAME_MAX_LENGTH));
}

function user_validate_mail($mail) {
  if (!$mail) return t('You must enter an e-mail address.');
  if (!valid_email_address($mail)) {
    return t('The e-mail address %mail is not valid.', array('%mail' =>
$mail));
  }
}

function user_validate_picture($file, &$edit, $user) {
  global $form_values;
  // Initialize the picture:
  $form_values['picture'] = $user->picture;

  // Check that uploaded file is an image, with a maximum file size
  // and maximum height/width.
  $info = image_get_info($file->filepath);
  list($maxwidth, $maxheight) = explode('x',
variable_get('user_picture_dimensions', '85x85'));

  if (!$info || !$info['extension']) {
    form_set_error('picture_upload', t('The uploaded file was not an image.'));
  }
  else if (image_get_toolkit()) {
    image_scale($file->filepath, $file->filepath, $maxwidth, $maxheight);
  }
  else if (filesize($file->filepath) > (variable_get('user_picture_file_size',
'30') * 1000)) {
    form_set_error('picture_upload', t('The uploaded image is too large; the
maximum file size is %size kB.', array('%size' =>
variable_get('user_picture_file_size', '30'))));
  }
  else if ($info['width'] > $maxwidth || $info['height'] > $maxheight) {
    form_set_error('picture_upload', t('The uploaded image is too large; the
maximum dimensions are %dimensions pixels.', array('%dimensions' =>
variable_get('user_picture_dimensions', '85x85'))));
  }

  if (!form_get_errors()) {
    if ($file = file_save_upload('picture_upload',
variable_get('user_picture_path', 'pictures') .'/picture-'. $user->uid .'.'.
$info['extension'], 1)) {
      $form_values['picture'] = $file->filepath;
    }
    else {
      form_set_error('picture_upload', t("Failed to upload the picture image;
the %directory directory doesn't exist.", array('%directory' =>
variable_get('user_picture_path', 'pictures'))));
```

```
    }
  }
}

/**
 * Generate a random alphanumeric password.
 */
function user_password($length = 10) {
  // This variable contains the list of allowable characters for the
  // password. Note that the number 0 and the letter 'O' have been
  // removed to avoid confusion between the two. The same is true
  // of 'I', 1, and l.
  $allowable_characters =
'abcdefghijkmnopqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ23456789';

  // Zero-based count of characters in the allowable list:
  $len = strlen($allowable_characters) - 1;

  // Declare the password as a blank string.
  $pass = '';

  // Loop the number of times specified by $length.
  for ($i = 0; $i < $length; $i++) {

    // Each iteration, pick a random character from the
    // allowable string and append it to the password:
    $pass .= $allowable_characters[mt_rand(0, $len)];
  }

  return $pass;
}

/**
 * Determine whether the user has a given privilege.
 *
 * @param $string
 *   The permission, such as "administer nodes", being checked for.
 * @param $account
 *   (optional) The account to check, if not given use currently logged in
user.
 *
 * @return
 *   boolean TRUE if the current user has the requested permission.
 *
 * All permission checks in Drupal should go through this function. This
 * way, we guarantee consistent behavior, and ensure that the superuser
 * can perform all actions.
 */
function user_access($string, $account = NULL) {
  global $user;
  static $perm = array();

  if (is_null($account)) {
    $account = $user;
  }

  // User #1 has all privileges:
  if ($account->uid == 1) {
    return TRUE;
  }

  // To reduce the number of SQL queries, we cache the user's permissions
  // in a static variable.
```

```
  if (!isset($perm[$account->uid])) {
    $result = db_query("SELECT DISTINCT(p.perm) FROM {role} r INNER JOIN
{permission} p ON p.rid = r.rid WHERE r.rid IN (%s)", implode(',',
array_keys($account->roles)));

    $perm[$account->uid] = '';
    while ($row = db_fetch_object($result)) {
      $perm[$account->uid] .= "$row->perm, ";
    }
  }

  if (isset($perm[$account->uid])) {
    return strpos($perm[$account->uid], "$string, ") !== FALSE;
  }

  return FALSE;
}

/**
 * Checks for usernames blocked by user administration
 *
 * @return boolean TRUE for blocked users, FALSE for active
 */
function user_is_blocked($name) {
  $deny  = db_fetch_object(db_query("SELECT name FROM {users} WHERE status = 0
AND name = LOWER('%s')", $name));

  return $deny;
}

function user_fields() {
  static $fields;

  if (!$fields) {
    $result = db_query('SELECT * FROM {users} WHERE uid = 1');
    if (db_num_rows($result)) {
      $fields = array_keys(db_fetch_array($result));
    }
    else {
      // Make sure we return the default fields at least
      $fields = array('uid', 'name', 'pass', 'mail', 'picture', 'mode', 'sort',
'threshold', 'theme', 'signature', 'created', 'access', 'login', 'status',
'timezone', 'language', 'init', 'data');
    }
  }

  return $fields;
}

/**
 * Implementation of hook_perm().
 */
function user_perm() {
  return array('administer access control', 'administer users', 'access user
profiles', 'change own username');
}

/**
 * Implementation of hook_file_download().
 *
 * Ensure that user pictures (avatars) are always downloadable.
 */
function user_file_download($file) {
```

```
  if (strpos($file, variable_get('user_picture_path', 'pictures') .'/picture-')
=== 0) {
    $info = image_get_info(file_create_path($file));
    return array('Content-type: '. $info['mime_type']);
  }
}

/**
 * Implementation of hook_search().
 */

/**
function user_search($op = 'search', $keys = NULL) {
  switch ($op) {
    case 'name':
      if (user_access('access user profiles')) {
        return t('Users');
      }
    case 'search':
      if (user_access('access user profiles')) {
        $find = array();
        // Replace wildcards with MySQL/PostgreSQL wildcards.
        $keys = preg_replace('!\*+!', '%', $keys);
        $result = pager_query("SELECT * FROM {users} WHERE LOWER(name) LIKE
LOWER('%%%s%%')", 15, 0, NULL, $keys);
        while ($account = db_fetch_object($result)) {
          $find[] = array('title' => $account->name, 'link' => url('user/'.
$account->uid, NULL, NULL, TRUE));
        }
        return $find;
      }
  }
}

*/

/**
 * Implementation of hook_user().
 */
function user_user($type, &$edit, &$user, $category = NULL) {
  if ($type == 'view') {
    $items['history'] = array('title' => t('Member for'),
      'value' => format_interval(time() - $user->created),
      'class' => 'member',
    );

    return array(t('History') => $items);
  }
  if ($type == 'form' && $category == 'account') {
    return user_edit_form(arg(1), $edit);
  }

  if ($type == 'validate' && $category == 'account') {
    return _user_edit_validate(arg(1), $edit);
  }

  if ($type == 'submit' && $category == 'account') {
    return _user_edit_submit(arg(1), $edit);
  }

  if ($type == 'categories') {
    return array(array('name' => 'account', 'title' => t('Account settings'),
'weight' => 1));
```

```
  }
}

function user_login_block() {
  $form = array(
    '#action' => url($_GET['q'], drupal_get_destination()),
    '#id' => 'user-login-form',
    '#base' => 'user_login',
  );
  $form['name'] = array('#type' => 'textfield',
    '#title' => t('Username'),
    '#maxlength' => USERNAME_MAX_LENGTH,
    '#size' => 15,
    '#required' => TRUE,
  );
  $form['pass'] = array('#type' => 'password',
    '#title' => t('Password'),
    '#maxlength' => 60,
    '#size' => 15,
    '#required' => TRUE,
  );
  $form['submit'] = array('#type' => 'submit',
    '#value' => t('Log in'),
  );
  $items = array();
  if (variable_get('user_register', 1)) {
    $items[] = l(t('Create new account'), 'user/register', array('title' =>
t('Create a new user account.')));
  }
  $items[] = l(t('Request new password'), 'user/password', array('title' =>
t('Request new password via e-mail.')));
  $form['links'] = array('#value' => theme('item_list', $items));
  return $form;
}

/**
 * Implementation of hook_block().
 */
function user_block($op = 'list', $delta = 0, $edit = array()) {
  global $user;

  if ($op == 'list') {
     $blocks[0]['info'] = t('User login');
     $blocks[1]['info'] = t('Navigation');
     $blocks[2]['info'] = t('Who\'s new');
     $blocks[3]['info'] = t('Who\'s online');

     return $blocks;
  }
  else if ($op == 'configure' && $delta == 2) {
    $form['user_block_whois_new_count'] = array(
      '#type' => 'select',
      '#title' => t('Number of users to display'),
      '#default_value' => variable_get('user_block_whois_new_count', 5),
      '#options' => drupal_map_assoc(array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)),
    );
    return $form;
  }
  else if ($op == 'configure' && $delta == 3) {
    $period = drupal_map_assoc(array(30, 60, 120, 180, 300, 600, 900, 1800,
2700, 3600, 5400, 7200, 10800, 21600, 43200, 86400), 'format_interval');
    $form['user_block_seconds_online'] = array('#type' => 'select', '#title' =>
t('User activity'), '#default_value' =>
```

```
variable_get('user_block_seconds_online', 900), '#options' => $period,
'#description' => t('A user is considered online for this long after they have
last viewed a page.'));
    $form['user_block_max_list_count'] = array('#type' => 'select', '#title' =>
t('User list length'), '#default_value' =>
variable_get('user_block_max_list_count', 10), '#options' =>
drupal_map_assoc(array(0, 5, 10, 15, 20, 25, 30, 40, 50, 75, 100)),
'#description' => t('Maximum number of currently online users to display.'));

    return $form;
  }
  else if ($op == 'save' && $delta == 2) {
    variable_set('user_block_whois_new_count',
$edit['user_block_whois_new_count']);
  }
  else if ($op == 'save' && $delta == 3) {
    variable_set('user_block_seconds_online',
$edit['user_block_seconds_online']);
    variable_set('user_block_max_list_count',
$edit['user_block_max_list_count']);
  }
  else if ($op == 'view') {
    $block = array();

    switch ($delta) {
      case 0:
        // For usability's sake, avoid showing two login forms on one page.
        if (!$user->uid && !(arg(0) == 'user' && !is_numeric(arg(1)))) {

          $block['subject'] = t('User login');
          $block['content'] = drupal_get_form('user_login_block');
        }
        return $block;

      case 1:
        if ($menu = theme('menu_tree')) {
           $block['subject'] = $user->uid ? check_plain($user->name) :
t('Navigation');
           $block['content'] = $menu;
        }
        return $block;

      case 2:
        if (user_access('access content')) {
          // Retrieve a list of new users who have subsequently accessed the
site successfully.
          $result = db_query_range('SELECT uid, name FROM {users} WHERE status
!= 0 AND access != 0 ORDER BY name ASC', 0,
variable_get('user_block_whois_new_count', 5));
          while ($account = db_fetch_object($result)) {
            $items[] = $account;
          }
          $output = theme('user_list', $items);

          $block['subject'] = t('Who\'s new');
          $block['content'] = $output;
        }
        return $block;

      case 3:
        if (user_access('access content')) {
          // Count users with activity in the past defined period.
          $interval = time() - variable_get('user_block_seconds_online', 900);
```

```
          // Perform database queries to gather online user lists.  We use
s.timestamp
          // rather than u.access because it is much faster is much faster..
          $anonymous_count = sess_count($interval);
          $authenticated_users = db_query('SELECT u.uid, u.name FROM {users} u
INNER JOIN {sessions} s ON u.uid = s.uid WHERE s.timestamp >= %d AND s.uid > 0
ORDER BY s.timestamp DESC', $interval);
          $authenticated_count = db_num_rows($authenticated_users);

          // Format the output with proper grammar.
          if ($anonymous_count == 1 && $authenticated_count == 1) {
            $output = t('There is currently %members and %visitors online.',
array('%members' => format_plural($authenticated_count, '1 user', '@count
users'), '%visitors' => format_plural($anonymous_count, '1 guest', '@count
guests')));
          }
          else {
            $output = t('There are currently %members and %visitors online.',
array('%members' => format_plural($authenticated_count, '1 user', '@count
users'), '%visitors' => format_plural($anonymous_count, '1 guest', '@count
guests')));
          }

          // Display a list of currently online users.
          $max_users = variable_get('user_block_max_list_count', 10);
          if ($authenticated_count && $max_users) {
            $items = array();

            while ($max_users-- && $account =
db_fetch_object($authenticated_users)) {
              $items[] = $account;
            }

            $output .= theme('user_list', $items, t('Online users'));
          }

          $block['subject'] = t('Who\'s online');
          $block['content'] = $output;
        }
        return $block;
    }
  }
}

function theme_user_picture($account) {
  if (variable_get('user_pictures', 0)) {
    if ($account->picture && file_exists($account->picture)) {
      $picture = file_create_url($account->picture);
    }
    else if (variable_get('user_picture_default', '')) {
      $picture = variable_get('user_picture_default', '');
    }

    if (isset($picture)) {
      $alt = t("@user's picture", array('@user' => $account->name ? $account-
>name : variable_get('anonymous', t('Anonymous'))));
      $picture = theme('image', $picture, $alt, $alt, '', FALSE);
      if (!empty($account->uid) && user_access('access user profiles')) {
        $picture = l($picture, "user/$account->uid", array('title' => t('View
user profile.')), NULL, NULL, FALSE, TRUE);
      }
```

```
      return "<div class=\"picture\">$picture</div>";
    }
  }
}

/**
 * Theme a user page
 * @param $account the user object
 * @param $fields a multidimensional array for the fields, in the form of array
(
 *   'category1' => array(item_array1, item_array2), 'category2' =>
array(item_array3,
 *    .. etc.). Item arrays are formatted as array(array('title' => 'item
title',
 * 'value' => 'item value', 'class' => 'class-name'), ... etc.). Module names
are incorporated
 * into the CSS class.
 *
 * @ingroup themeable
 */


function theme_user_profile($account, $fields) {
 $start = '<div class="profile">';
 $count1 = 0;
 $count2 = 0;
 $quote = '';
 foreach ($fields as $category => $items) {
    $output = '';
    if (strlen($category) > 0) {
      if($count1 == 0)
       {
         $career = '<div id="career">';
       }
       else if($count1 == 1)
       {
         $picture .= theme('user_picture', $account);
         $college = '<div id="college">';
         $count1 = $count1 + 1;
       }
       else if( $count1 == 2)
       {
       }
       else if ( $count1 == 3)
       {
         $personal = '<div id="personal">';
       }
      $output .= '<h2 class="title">'. $category .'</h2>';
    }
    $output .= '<dl>';
    foreach ($items as $item) {
      if (isset($item['title'])) {
         $output .= '<dt class="'. $item['class'] .'">'. $item['title']
.'</dt>';
      }
         $output .= '<dd class="'. $item['class'] .'">'. $item['value']
.'</dd>';
         if ( $_SESSION['admin'] == 1)
         {
           if($item['class']=='profile-profile_full_name') {
              $fullname = $item['value'];
              $fullname = strip_tags($fullname);
           }
```

```
      else if($item['class']=='profile-profile_grad_year') {
         $gradyear = $item['value'];
         $gradyear = strip_tags($gradyear);
      }
      else if($item['class']=='profile-profile_home_address') {
         $location = $item['value'];
         $location = strip_tags($location);
      }
      else if($item['class']=='profile-profile_major') {
         $major = $item['value'];
         $major = strip_tags($major);
      }
      else if($item['class']=='profile-profile_fav_class') {
         $favclass = $item['value'];
         $favclass = strip_tags($favclass);
      }
      else if($item['class']=='profile-profile_exp') {
         $exper = $item['value'];
         $exper = strip_tags($exper);
         $order = array("\r\n", "\n", "\r");
         $exper = str_replace($order, ' ', $exper);
         $exper = str_replace("'", "\'", $exper);
      }
      else if($item['class']=='profile-profile_web_site') {
         $website = $item['value'];
         $website = strip_tags($website);
      }
      else if($item['class']=='profile-profile_company') {
         $company = $item['value'];
         $company = strip_tags($company);
      }
      else if($item['class']=='profile-profile_minor') {
         $minor = $item['value'];
         $minor = strip_tags($minor);
      }
      else if($item['class']=='profile-profile_fav_teacher') {
         $favteacher = $item['value'];
         $favteacher = strip_tags($favteacher);
      }
      else if($item['class']=='profile-profile_quote') {
         $favquote = $item['value'];
         $favquote = strip_tags($favquote);
         $order = array("\r\n", "\n", "\r");
         $favquote = str_replace($order, ' ', $favquote);
         $favquote = str_replace("'", "\'", $favquote);
      }
      else if($item['class']=='profile-profile_grad_school') {
         $gradschool = $item['value'];
         $gradschool = strip_tags($gradschool);
      }
    }
}

$output .= '</dl>';
$count1 = $count1 + 1;

if(($count1 !=2) && ($count1 <5)) {
  $output .= '</div>';
}

if($count2 == 0)
{
    $career .= $output;
```

```
    }
    else if($count2 == 1)
    {
        $college .= $output;
        $count2 = $count2 + 1;
    }
    else if( $count2 == 2)
    {
    }
    else if ( $count2 == 3)
    {
        $personal .= $output;
    }
    $count2 = $count2 + 1;
  }

  // Construct output the way we need it to be
  $tempHold = $output;

  $output = $start;
  $output .= $picture;
  $output .= $personal;
  $output .= $college;
  $output .= $career;
  $output .= '</div>';
  $output .= '<p style="clear:both"> </p>';

  if( $_SESSION['admin'] == 1)
   {
     $UserIDNum = $account->uid;
     $output .= '<input type="button" value="Spotlight User"
onClick="document.location=\'/spotlight/spotlight.php?FullName=' . $fullname;
     $output .= '&GradYear='.$gradyear . '&Hometown=' . $location . '&Major=' .
$major . '&FavClass=' . $favclass . '&Experience=' . $exper;
     $output .= '&WebSite='.$website . '&Company=' . $company . '&Minor=' .
$minor . '&FavTeacher=' . $favteacher . '&FavQuote=' .$favquote;
     $output .= '&GradSchool=' . $gradschool;
     $output .= '&UserIDNum='.$UserIDNum.'\';">';
   }

  return $output;
}


/**
function theme_user_profile($account, $fields) {
  $output = '<div class="profile">';
  $output .= theme('user_picture', $account);
  foreach ($fields as $category => $items) {
    if (strlen($category) > 0) {
      $output .= '<h2 class="title">'. $category .'</h2>';
    }
    $output .= '<dl>';
    foreach ($items as $item) {
      if (isset($item['title'])) {
        $output .= '<dt class="'. $item['class'] .'">'. $item['title']
.'</dt>';
      }
      $output .= '<dd class="'. $item['class'] .'">'. $item['value'] .'</dd>';
    }
    $output .= '</dl>';
  }
  $output .= '</div>';
```

```
  return $output;
}
*/
/**
 * Make a list of users.
 * @param $items an array with user objects. Should contain at least the name
and uid
 *
 * @ingroup themeable
 */
function theme_user_list($users, $title = NULL) {
  if (!empty($users)) {
    foreach ($users as $user) {
      $items[] = theme('username', $user);
    }
  }
  return theme('item_list', $items, $title);
}

/**
 * Implementation of hook_menu().
 */
function user_menu($may_cache) {
  global $user;

  $items = array();

  $admin_access = user_access('administer users');
  $access_access = user_access('administer access control');
  $view_access = user_access('access user profiles');

  if ($may_cache) {
    $items[] = array('path' => 'user', 'title' => t('User account'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_login'),
      'access' => !$user->uid, 'type' => MENU_CALLBACK);

    $items[] = array('path' => 'user/autocomplete', 'title' => t('User
autocomplete'),
      'callback' => 'user_autocomplete', 'access' => $view_access, 'type' =>
MENU_CALLBACK);

    // Registration and login pages.
    $items[] = array('path' => 'user/login', 'title' => t('Log in'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_login'),
      'access' => !$user->uid, 'type' => MENU_DEFAULT_LOCAL_TASK);
    $items[] = array('path' => 'user/register', 'title' => t('Create new
account'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_register'), 'access' => !$user->uid &&
variable_get('user_register', 1), 'type' => MENU_LOCAL_TASK);
    $items[] = array('path' => 'user/password', 'title' => t('Request new
password'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_pass'), 'access' => !$user->uid, 'type' => MENU_LOCAL_TASK);
    $items[] = array('path' => 'user/reset', 'title' => t('Reset password'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_pass_reset'), 'access' => TRUE, 'type' => MENU_CALLBACK);
    $items[] = array('path' => 'user/help', 'title' => t('Help'),
      'callback' => 'user_help_page', 'type' => MENU_CALLBACK);
```

```
    // Admin user pages
    $items[] = array('path' => 'admin/user',
      'title' => t('User management'),
      'description' => t('Manage your site\'s users, groups and access to site
features.'),
      'position' => 'left',
      'callback' => 'system_admin_menu_block_page',
      'access' => user_access('administer site configuration'),
    );
    $items[] = array('path' => 'admin/user/user', 'title' => t('Users'),
      'description' => t('List, add, and edit users.'),
      'callback' => 'user_admin', 'callback arguments' => array('list'),
'access' => $admin_access);
    $items[] = array('path' => 'admin/user/user/list', 'title' => t('List'),
      'type' => MENU_DEFAULT_LOCAL_TASK, 'weight' => -10);
    $items[] = array('path' => 'admin/user/user/create', 'title' => t('Add
user'),
      'callback' => 'user_admin', 'callback arguments' => array('create'),
'access' => $admin_access,
      'type' => MENU_LOCAL_TASK);
    $items[] = array('path' => 'admin/user/settings', 'title' => t('User
settings'),
      'description' => t('Configure default behavior of users, including
registration requirements, e-mails, and user pictures.'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_admin_settings'));

    // Admin access pages
    $items[] = array('path' => 'admin/user/access', 'title' => t('Access
control'),
      'description' => t('Determine access to features by selecting permissions
for roles.'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_admin_perm'), 'access' => $access_access);
    $items[] = array('path' => 'admin/user/roles', 'title' => t('Roles'),
      'description' => t('List, edit, or add user roles.'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_admin_new_role'), 'access' => $access_access,
      'type' => MENU_NORMAL_ITEM);
    $items[] = array('path' => 'admin/user/roles/edit', 'title' => t('Edit
role'),
       'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_admin_role'), 'access' => $access_access,
      'type' => MENU_CALLBACK);
    $items[] = array('path' => 'admin/user/rules', 'title' => t('Access
rules'),
      'description' => t('List and create rules to disallow usernames, e-mail
addresses, and IP addresses.'),
      'callback' => 'user_admin_access', 'access' => $access_access);
    $items[] = array('path' => 'admin/user/rules/list', 'title' => t('List'),
      'access' => $access_access, 'type' => MENU_DEFAULT_LOCAL_TASK, 'weight'
=> -10);
    $items[] = array('path' => 'admin/user/rules/add', 'title' => t('Add
rule'),
      'callback' => 'user_admin_access_add', 'access' => $access_access,
      'type' => MENU_LOCAL_TASK);
    $items[] = array('path' => 'admin/user/rules/check', 'title' => t('Check
rules'),
      'callback' => 'user_admin_access_check', 'access' => $access_access,
      'type' => MENU_LOCAL_TASK);
    $items[] = array('path' => 'admin/user/rules/edit', 'title' => t('Edit
rule'),
      'callback' => 'user_admin_access_edit', 'access' => $access_access,
```

```
      'type' => MENU_CALLBACK);
    $items[] = array('path' => 'admin/user/rules/delete', 'title' => t('Delete
rule'),
      'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_admin_access_delete_confirm'),
      'access' => $access_access, 'type' => MENU_CALLBACK);

    if (module_exists('search')) {
      $items[] = array('path' => 'admin/user/search', 'title' => t('Search
users'),
        'description' => t('Search users by name.'),
        'callback' => 'user_admin', 'callback arguments' => array('search'),
'access' => $admin_access,
        'type' => MENU_NORMAL_ITEM);
    }

    // Your personal page
    if ($user->uid) {
      $items[] = array('path' => 'user/'. $user->uid, 'title' => t('My
account'),
        'callback' => 'user_view', 'callback arguments' => array(arg(1)),
'access' => TRUE,
        'type' => MENU_DYNAMIC_ITEM);
    }

    $items[] = array('path' => 'logout', 'title' => t('Log out'),
      'access' => $user->uid,
      'callback' => 'user_logout',
      'weight' => 10);
  }
  else {
    // Add the CSS for this module. We put this in !$may_cache so it is only
    // added once per request.
    drupal_add_css(drupal_get_path('module', 'user') .'/user.css', 'module');
    if ($_GET['q'] == 'user' && $user->uid) {
      // We want to make the current user's profile accessible without knowing
      // their uid, so just linking to /user is enough.
      drupal_goto('user/'. $user->uid);
    }

    if (arg(0) == 'user' && is_numeric(arg(1)) && arg(1) > 0) {
      $account = user_load(array('uid' => arg(1)));

      if ($user !== FALSE) {
        // Always let a user view their own account
        $view_access |= $user->uid == arg(1);
        // Only admins can view blocked accounts
        $view_access &= $account->status || $admin_access;

        $items[] = array('path' => 'user/'. arg(1), 'title' => t('User'),
          'type' => MENU_CALLBACK, 'callback' => 'user_view',
          'callback arguments' => array(arg(1)), 'access' => $view_access);

        $items[] = array('path' => 'user/'. arg(1) .'/view', 'title' =>
t('View'),
          'access' => $view_access, 'type' => MENU_DEFAULT_LOCAL_TASK, 'weight'
=> -10);

        $items[] = array('path' => 'user/'. arg(1) .'/edit', 'title' =>
t('Edit'),
          'callback' => 'drupal_get_form', 'callback arguments' =>
array('user_edit'),
```

```
          'access' => $admin_access || $user->uid == arg(1), 'type' =>
MENU_LOCAL_TASK);

       $items[] = array('path' => 'user/'. arg(1) .'/delete', 'title' =>
t('Delete'),
          'callback' => 'user_edit', 'access' => $admin_access,
          'type' => MENU_CALLBACK);

        if (arg(2) == 'edit') {
          if (($categories = _user_categories($account)) && (count($categories)
> 1)) {
            foreach ($categories as $key => $category) {
              $items[] = array(
                'path' => 'user/'. arg(1) .'/edit/'. $category['name'],
                'title' => $category['title'],
                'type' => $category['name'] == 'account' ?
MENU_DEFAULT_LOCAL_TASK : MENU_LOCAL_TASK,
                'weight' => $category['weight'],
                'access' => ($admin_access || $user->uid == arg(1)));
            }
          }
        }
      }
    }
  }

  return $items;
}

/**
 * Accepts an user object, $account, or a DA name and returns an associative
 * array of modules and DA names. Called at external login.
 */
function user_get_authmaps($authname = NULL) {
  $result = db_query("SELECT authname, module FROM {authmap} WHERE authname =
'%s'", $authname);
  if (db_num_rows($result) > 0) {
    while ($authmap = db_fetch_object($result)) {
      $authmaps[$authmap->module] = $authmap->authname;
    }
    return $authmaps;
  }
  else {
    return 0;
  }
}

function user_set_authmaps($account, $authmaps) {
  foreach ($authmaps as $key => $value) {
    $module = explode('_', $key, 2);
    if ($value) {
      db_query("UPDATE {authmap} SET authname = '%s' WHERE uid = %d AND module
= '%s'", $value, $account->uid, $module[1]);
      if (!db_affected_rows()) {
        db_query("INSERT INTO {authmap} (authname, uid, module) VALUES ('%s',
%d, '%s')", $value, $account->uid, $module[1]);
      }
    }
    else {
      db_query("DELETE FROM {authmap} WHERE uid = %d AND module = '%s'",
$account->uid, $module[1]);
    }
  }
```

```
}

function user_auth_help_links() {
  $links = array();
  foreach (module_list() as $module) {
    if (module_hook($module, 'auth')) {
      $links[] = l(module_invoke($module, 'info', 'name'), 'user/help',
array(), NULL, $module);
    }
  }
  return $links;
}

/*** User features ***********************************************************/



function user_login($msg = '') {
  global $user;

  // If we are already logged on, go to the user page instead.
  if ($user->uid) {
    drupal_goto('user/'. $user->uid);
  }

  // Display login form:
  if ($msg) {
    $form['message'] = array('#value' => '<p>'. check_plain($msg) .'</p>');
  }
  $form['name'] = array('#type' => 'textfield',
    '#title' => t('Username'),
    '#size' => 60,
    '#maxlength' => USERNAME_MAX_LENGTH,
    '#required' => TRUE,
    '#attributes' => array('tabindex' => '1'),
  );
  if (variable_get('drupal_authentication_service', FALSE) &&
count(user_auth_help_links()) > 0) {
    $form['name']['#description'] = t('Enter your @s username, or an ID from
one of our affiliates: !a.', array('@s' => variable_get('site_name', 'Drupal'),
'!a' => implode(', ', user_auth_help_links())));
  }
  else {
    $form['name']['#description'] = t('Enter your @s username.', array('@s' =>
variable_get('site_name', 'Drupal')));
  }
  $form['pass'] = array('#type' => 'password',
    '#title' => t('Password'),
    '#description' => t('Enter the password that accompanies your username.'),
    '#required' => TRUE,
    '#attributes' => array('tabindex' => '2'),
  );
  $form['submit'] = array('#type' => 'submit', '#value' => t('Log in'),
'#weight' => 2, '#attributes' => array('tabindex' => '3'));

  return $form;
}

function user_login_validate($form_id, $form_values) {
  if ($form_values['name']) {
    if (user_is_blocked($form_values['name'])) {
      // blocked in user administration
```

```
      form_set_error('login', t('The username %name has not been activated or
is blocked.', array('%name' => $form_values['name'])));
    }
    else if (drupal_is_denied('user', $form_values['name'])) {
      // denied by access controls
      form_set_error('login', t('The name %name is a reserved username.',
array('%name' => $form_values['name'])));
    }
    else if ($form_values['pass']) {
      $user = user_authenticate($form_values['name'],
trim($form_values['pass']));

      if (!$user->uid) {
        form_set_error('login', t('Sorry, unrecognized username or password. <a
href="@password">Have you forgotten your password?</a>', array('@password' =>
url('user/password'))));
        watchdog('user', t('Login attempt failed for %user.', array('%user' =>
$form_values['name'])));
      }
    }
  }
}

function user_login_submit($form_id, $form_values) {
  global $user;
  if ($user->uid) {
    // To handle the edge case where this function is called during a
    // bootstrap, check for the existence of t().
    if (function_exists('t')) {
      $message = t('Session opened for %name.', array('%name' => $user->name));
    }
    else {
       $message = "Session opened for ". check_plain($user->name);
    }
    watchdog('user', $message);

    // Update the user table timestamp noting user has logged in.
    db_query("UPDATE {users} SET login = %d WHERE uid = %d", time(), $user-
>uid);

    user_module_invoke('login', $form_values, $user);

    sess_regenerate();
    $_SESSION['admin'] = $user->uid;
    return 'user/'. $user->uid;
  }
}

function user_authenticate($name, $pass) {
  global $user;

  // Try to log in the user locally. Don't set $user unless successful.
  if ($account = user_load(array('name' => $name, 'pass' => $pass, 'status' =>
1))) {
    $user = $account;
    return $user;
  }

  // Strip name and server from ID:
  if ($server = strrchr($name, '@')) {
    $name = substr($name, 0, strlen($name) - strlen($server));
    $server = substr($server, 1);
  }
```

```
  // When possible, determine corresponding external auth source. Invoke
  // source, and log in user if successful:
  if ($server && ($result = user_get_authmaps("$name@$server"))) {
    if (module_invoke(key($result), 'auth', $name, $pass, $server)) {
      $user = user_external_load("$name@$server");
      watchdog('user', t('External load by %user using module %module.',
array('%user' => $name .'@'. $server, '%module' => key($result))));
    }
  }

  // Try each external authentication source in series. Register user if
  // successful.
  else {
    foreach (module_implements('auth') as $module) {
      if (module_invoke($module, 'auth', $name, $pass, $server)) {
        if ($server) {
          $name .= '@'. $server;
        }
        $user = user_load(array('name' => $name));
        if (!$user->uid) { // Register this new user.
          $userinfo = array('name' => $name, 'pass' => user_password(), 'init'
=> $name, 'status' => 1);
          if ($server) {
            $userinfo["authname_$module"] = $name;
          }
          $user = user_save('', $userinfo);
          watchdog('user', t('New external user: %user using module %module.',
array('%user' => $name, '%module' => $module)), WATCHDOG_NOTICE, l(t('edit'),
'user/'. $user->uid .'/edit'));
          break;
        }
      }
    }
  }
  return $user;
}

/**
 * Menu callback; logs the current user out, and redirects to the home page.
 */
function user_logout() {
  global $user;

  watchdog('user', t('Session closed for %name.', array('%name' => $user-
>name)));

  // Destroy the current session:
  session_destroy();
  module_invoke_all('user', 'logout', NULL, $user);

  // Load the anonymous user
  $user = drupal_anonymous_user();

  drupal_goto();
}

function user_pass() {

$string = '<p> To get your user name/password please send an email to the admin
at <a href="mailto:ebreimer@gmail.com?subject=Siena College Alumni Spotlight
System: Lost Password">by clicking here.</a> You can also send him an email at
ebreimer@gmail.com. The admin will return you your current password, or you may
```

```
request to have it reset. Simply state what you would like your new password to
be.</p>';
  drupal_set_message(t($string));
  $form['submit'] = array('#type' => 'submit',
    '#value' => t('Go Back to Home'),
    '#weight' => 2
  );

  return $form;

}

function user_pass_validate($form_id, $form_values) {
  unset($_SESSION['messages']);
  drupal_goto('node/8');
/*
  $name = $form_values['name'];
  $account = user_load(array('mail' => $name, 'status' => 1));
  if (!$account) {
    $account = user_load(array('name' => $name, 'status' => 1));
  }
  if ($account->uid) {
    form_set_value(array('#parents' => array('account')), $account);
  }
  else {
    form_set_error('name', t('Sorry, %name is not recognized as a user name or
an email address.', array('%name' => $name)));
  }
*/
}

function user_pass_submit($form_id, $form_values) {
/*
  global $base_url;

  $account = $form_values['account'];
  $from = variable_get('site_mail', ini_get('sendmail_from'));

  // Mail one time login URL and instructions.
  $variables = array('!username' => $account->name, '!site' =>
variable_get('site_name', 'Drupal'), '!login_url' =>
user_pass_reset_url($account), '!uri' => $base_url, '!uri_brief' =>
substr($base_url, strlen('http://')), '!mailto' => $account->mail, '!date' =>
format_date(time()), '!login_uri' => url('user', NULL, NULL, TRUE), '!edit_uri'
=> url('user/'. $account->uid .'/edit', NULL, NULL, TRUE));
  $subject = _user_mail_text('pass_subject', $variables);
  $body = _user_mail_text('pass_body', $variables);
  $mail_success = drupal_mail('user-pass', $account->mail, $subject, $body,
$from);

  if ($mail_success) {
    watchdog('user', t('Password reset instructions mailed to %name at
%email.', array('%name' => $account->name, '%email' => $account->mail)));
    drupal_set_message(t('Further instructions have been sent to your e-mail
address.'));
  }
  else {
    watchdog('user', t('Error mailing password reset instructions to %name at
%email.', array('%name' => $account->name, '%email' => $account->mail)),
WATCHDOG_ERROR);
    drupal_set_message(t('Unable to send mail. Please contact the site
admin.'));
  }
```

```
  return 'user';
*/
}

/**
 * Menu callback; process one time login link and redirects to the user page on
success.
 */
function user_pass_reset($uid, $timestamp, $hashed_pass, $action = NULL) {
  global $user;

  // Check if the user is already logged in. The back button is often the
culprit here.
  if ($user->uid) {
    drupal_set_message(t('You have already used this one-time login link. It is
not necessary to use this link to login anymore. You are already logged in.'));
    drupal_goto();
  }
  else {
    // Time out, in seconds, until login URL expires. 24 hours = 86400 seconds.
    $timeout = 86400;
    $current = time();
    // Some redundant checks for extra security ?
    if ($timestamp < $current && $account = user_load(array('uid' => $uid,
'status' => 1)) ) {
      // No time out for first time login.
      if ($account->login && $current - $timestamp > $timeout) {
        drupal_set_message(t('You have tried to use a one-time login link that
has expired. Please request a new one using the form below.'));
        drupal_goto('user/password');
      }
      else if ($account->uid && $timestamp > $account->login && $timestamp <
$current && $hashed_pass == user_pass_rehash($account->pass, $timestamp,
$account->login)) {
        // First stage is a confirmation form, then login
        if ($action == 'login') {
          watchdog('user', t('User %name used one-time login link at time
%timestamp.', array('%name' => $account->name, '%timestamp' => $timestamp)));
          // Update the user table noting user has logged in.
          // And this also makes this hashed password a one-time-only login.
          db_query("UPDATE {users} SET login = %d WHERE uid = %d", time(),
$account->uid);
          // Now we can set the new user.
          $user = $account;
          // And proceed with normal login, going to user page.
          $edit = array();
          user_module_invoke('login', $edit, $user);
          drupal_set_message(t('You have just used your one-time login link. It
is no longer necessary to use this link to login. Please change your
password.'));
          drupal_goto('user/'. $user->uid .'/edit');
        }
        else {
          $form['message'] = array('#value' => t('<p>This is a one-time login
for %user_name and will expire on %expiration_date</p><p>Click on this button
to login to the site and change your password.</p>', array('%user_name' =>
$account->name, '%expiration_date' => format_date($timestamp + $timeout))));
          $form['help'] = array('#value' => '<p>'. t('This login can be used
only once.') .'</p>');
          $form['submit'] = array('#type' => 'submit', '#value' => t('Log
in'));
          $form['#action'] =
url("user/reset/$uid/$timestamp/$hashed_pass/login");
```

```
          return $form;
        }
      }
      else {
        drupal_set_message(t('You have tried to use a one-time login link which
has either been used or is no longer valid. Please request a new one using the
form below.'));
        drupal_goto('user/password');
      }
    }
    else {
      // Deny access, no more clues.
      // Everything will be in the watchdog's URL for the administrator to
check.
      drupal_access_denied();
    }
  }
}

function user_pass_reset_url($account) {
  $timestamp = time();
  return url("user/reset/$account->uid/$timestamp/".user_pass_rehash($account-
>pass, $timestamp, $account->login), NULL, NULL, TRUE);
}

function user_pass_rehash($password, $timestamp, $login) {
  return md5($timestamp . $password . $login);
}

function user_register() {
  global $user;

  $admin = user_access('administer users');

  // If we aren't admin but already logged on, go to the user page instead.
  if (!$admin && $user->uid) {
    drupal_goto('user/'. $user->uid);
  }

  $form = array();

  // Display the registration form.
  if (!$admin) {
    $form['user_registration_help'] = array('#value' =>
filter_xss_admin(variable_get('user_registration_help', '')));
  }
  $affiliates = user_auth_help_links();
  if (!$admin && count($affiliates) > 0) {
    $affiliates = implode(', ', $affiliates);
    $form['affiliates'] = array('#value' => '<p>'. t('Note: if you have an
account with one of our affiliates (!s), you may <a href="@login_uri">login
now</a> instead of registering.', array('!s' => $affiliates, '@login_uri' =>
url('user'))) .'</p>');
  }
  // Merge in the default user edit fields.
  $form = array_merge($form, user_edit_form(NULL, NULL, TRUE));
  if ($admin) {
    $form['account']['notify'] = array(
     '#type' => 'checkbox',
     '#title' => t('Notify user of new account')
    );
    // Redirect back to page which initiated the create request; usually
admin/user/user/create
```

```
    $form['destination'] = array('#type' => 'hidden', '#value' => $_GET['q']);
  }

  // Create a dummy variable for pass-by-reference parameters.
  $null = NULL;
  $extra = _user_forms($null, NULL, NULL, 'register');

  // Remove form_group around default fields if there are no other groups.
  if (!$extra) {
    $form['name'] = $form['account']['name'];
    $form['mail'] = $form['account']['mail'];
    $form['pass'] = $form['account']['pass'];
    $form['status'] = $form['account']['status'];
    $form['roles'] = $form['account']['roles'];
    $form['notify'] = $form['account']['notify'];
    unset($form['account']);
  }
  else {
    $form = array_merge($form, $extra);
  }
  $form['submit'] = array('#type' => 'submit', '#value' => t('Create new
account'), '#weight' => 30);

  return $form;
}

function user_register_validate($form_id, $form_values) {
  user_module_invoke('validate', $form_values, $form_values, 'account');
}

function user_register_submit($form_id, $form_values) {
  global $base_url;
  $admin = user_access('administer users');

  $mail = $form_values['mail'];
  $name = $form_values['name'];
  if (!variable_get('user_email_verification', TRUE) || $admin) {
    $pass = $form_values['pass'];
  }
  else {
    $pass = user_password();
  };
  $notify = $form_values['notify'];
  $from = variable_get('site_mail', ini_get('sendmail_from'));
  if (isset($form_values['roles'])) {
    $roles = array_filter($form_values['roles']);     // Remove unset roles
  }

  if (!$admin && array_intersect(array_keys($form_values), array('uid',
'roles', 'init', 'session', 'status'))) {
    watchdog('security', t('Detected malicious attempt to alter protected user
fields.'), WATCHDOG_WARNING);
    return 'user/register';
  }
  //the unset below is needed to prevent these form values from being saved as
user data
  unset($form_values['form_token'], $form_values['submit'], $form_values['op'],
$form_values['notify'], $form_values['form_id'], $form_values['affiliates'],
$form_values['destination']);

  $merge_data = array('pass' => $pass, 'init' => $mail, 'roles' => $roles);
  if (!$admin) {
    // Set the user's status because it was not displayed in the form.
```

```
    $merge_data['status'] = variable_get('user_register', 1) == 1;
  }
  $account = user_save('', array_merge($form_values, $merge_data));
  watchdog('user', t('New user: %name %email.', array('%name' => $name,
'%email' => '<'. $mail .'>')), WATCHDOG_NOTICE, l(t('edit'), 'user/'. $account-
>uid .'/edit'));

  $variables = array('!username' => $name, '!site' => variable_get('site_name',
'Drupal'), '!password' => $pass, '!uri' => $base_url, '!uri_brief' =>
substr($base_url, strlen('http://')), '!mailto' => $mail, '!date' =>
format_date(time()), '!login_uri' => url('user', NULL, NULL, TRUE), '!edit_uri'
=> url('user/'. $account->uid .'/edit', NULL, NULL, TRUE), '!login_url' =>
user_pass_reset_url($account));

  // The first user may login immediately, and receives a customized welcome e-
mail.
  if ($account->uid == 1) {
    drupal_mail('user-register-admin', $mail, t('Drupal user account details
for !s', array('!s' => $name)), strtr(t("!username,\n\nYou may now login to
!uri using the following username and password:\n\n  username: !username\n
password: !password\n\n!edit_uri\n\n--drupal"), $variables), $from);
    drupal_set_message(t('<p>Welcome to Drupal. You are user #1, which gives
you full and immediate access. All future registrants will receive their
passwords via e-mail, so please make sure your website e-mail address is set
properly under the general settings on the <a href="@settings">site information
settings page</a>.</p><p> Your password is <strong>%pass</strong>. You may
change your password below.</p>', array('%pass' => $pass, '@settings' =>
url('admin/settings/site-information'))));
    user_authenticate($account->name, trim($pass));

    return 'user/1/edit';
  }
  else {
    if ($admin && !$notify) {
      drupal_set_message(t('Created a new user account. No e-mail has been
sent.'));
    }
    else if (!variable_get('user_email_verification', TRUE) && $account->status
&& !$admin) {
      // No e-mail verification is required, create new user account, and login
user immediately.
      $subject = _user_mail_text('welcome_subject', $variables);
      $body = _user_mail_text('welcome_body', $variables);
      drupal_mail('user-register-welcome', $mail, $subject, $body, $from);
      user_authenticate($account->name, trim($pass));
      drupal_goto();
    }
    else if ($account->status || $notify) {
      // Create new user account, no administrator approval required.
      $subject = $notify ? _user_mail_text('admin_subject', $variables) :
_user_mail_text('welcome_subject', $variables);
      $body = $notify ? _user_mail_text('admin_body', $variables) :
_user_mail_text('welcome_body', $variables);

      drupal_mail(($notify ? 'user-register-notify' : 'user-register-welcome'),
$mail, $subject, $body, $from);

      if ($notify) {
        drupal_set_message(t('Password and further instructions have been e-
mailed to the new user %user.', array('%user' => $name)));
      }
      else {
```

```
        drupal_set_message(t('Your password and further instructions have been
sent to your e-mail address.'));
        return '';
      }
    }
    else {
      // Create new user account, administrator approval required.
      $subject = _user_mail_text('approval_subject', $variables);
      $body = _user_mail_text('approval_body', $variables);

      drupal_mail('user-register-approval-user', $mail, $subject, $body,
$from);
      drupal_mail('user-register-approval-admin', $from, $subject, t("!username
has applied for an account.\n\n!edit_uri", $variables), $from);
      drupal_set_message(t('Thank you for applying for an account. Your account
is currently pending approval by the site administrator.<br />In the meantime,
your password and further instructions have been sent to your e-mail
address.'));

    }
  }
}

function user_edit_form($uid, $edit, $register = FALSE) {
  $admin = user_access('administer users');

  // Account information:
  $form['account'] = array('#type' => 'fieldset',
    '#title' => t('Account information'),
  );
  if (user_access('change own username') || $admin || $register) {
    $form['account']['name'] = array('#type' => 'textfield',
      '#title' => t('Username'),
      '#default_value' => $edit['name'],
      '#maxlength' => USERNAME_MAX_LENGTH,
      '#description' => t('Your preferred username; punctuation is not allowed
except for periods, hyphens, and underscores.'),
      '#required' => TRUE,
    );
  }
  $form['account']['mail'] = array('#type' => 'textfield',
    '#title' => t('E-mail address'),
    '#default_value' => $edit['mail'],
    '#maxlength' => EMAIL_MAX_LENGTH,
    '#description' => t('A valid e-mail address. All e-mails from the system
will be sent to this address. The e-mail address is not made public and will
only be used if you wish to receive a new password or wish to receive certain
news or notifications by e-mail.'),
    '#required' => TRUE,
  );
  if (!$register) {
    $form['account']['pass'] = array('#type' => 'password_confirm',
      '#description' => t('To change the current user password, enter the new
password in both fields.'),
      '#size' => 25,
       '#maxlength' => 12,
       '#required' => FALSE,
    );
  }

  elseif (!variable_get('user_email_verification', TRUE) || $admin) {
    $form['account']['pass'] = array(
      '#type' => 'password_confirm',
```

```
      '#description' => t('Provide a password for the new account in both
fields. This password must be 6-12 characters long, and contain at least one
digit.'),
      '#required' => TRUE,
      '#size' => 25,
       '#maxlength' => 12,
    );
  }
  if ($admin) {
    $form['account']['status'] = array('#type' => 'radios', '#title' =>
t('Status'), '#default_value' => isset($edit['status']) ? $edit['status'] : 1,
'#options' => array(t('Blocked'), t('Active')));
  }
  if (user_access('administer access control')) {
    $roles = user_roles(1);
    unset($roles[DRUPAL_AUTHENTICATED_RID]);
    if ($roles) {
      $form['account']['roles'] = array('#type' => 'checkboxes', '#title' =>
t('Roles'), '#default_value' => array_keys((array)$edit['roles']), '#options'
=> $roles, '#description' => t('The user receives the combined permissions of
the %au role, and all roles selected here.', array('%au' => t('authenticated
user'))));
    }
  }

  // Picture/avatar:
  if (variable_get('user_pictures', 0) && !$register) {
    $form['picture'] = array('#type' => 'fieldset', '#title' => t('Picture'),
'#weight' => 1);
    $picture = theme('user_picture', (object)$edit);
    if ($picture) {
      $form['picture']['current_picture'] = array('#value' => $picture);
      $form['picture']['picture_delete'] = array('#type' => 'checkbox',
'#title' => t('Delete picture'), '#description' => t('Check this box to delete
your current picture.'));
    }
    else {
      $form['picture']['picture_delete'] = array('#type' => 'hidden');
    }
    $form['picture']['picture_upload'] = array('#type' => 'file', '#title' =>
t('Upload picture'), '#size' => 48, '#description' => t('Your virtual face or
picture. Maximum dimensions are %dimensions and the maximum size is %size kB.',
array('%dimensions' => variable_get('user_picture_dimensions', '85x85'),
'%size' => variable_get('user_picture_file_size', '30'))) .' '.
variable_get('user_picture_guidelines', ''));
  }

  return $form;
}

function _user_edit_validate($uid, &$edit) {
  $user = user_load(array('uid' => $uid));
  // Validate the username:
  if (user_access('change own username') || user_access('administer users') ||
arg(1) == 'register') {
    if ($error = user_validate_name($edit['name'])) {
      form_set_error('name', $error);
    }
    else if (db_num_rows(db_query("SELECT uid FROM {users} WHERE uid != %d AND
LOWER(name) = LOWER('%s')", $uid, $edit['name'])) > 0) {
      form_set_error('name', t('The name %name is already taken.',
array('%name' => $edit['name'])));
    }
```

```
    else if (drupal_is_denied('user', $edit['name'])) {
      form_set_error('name', t('The name %name has been denied access.',
array('%name' => $edit['name'])));
    }
  }

  // Validate the e-mail address:
  if ($error = user_validate_mail($edit['mail'])) {
    form_set_error('mail', $error);
  }
  else if (db_num_rows(db_query("SELECT uid FROM {users} WHERE uid != %d AND
LOWER(mail) = LOWER('%s')", $uid, $edit['mail'])) > 0) {
    form_set_error('mail', t('The e-mail address %email is already registered.
<a href="@password">Have you forgotten your password?</a>', array('%email' =>
$edit['mail'], '@password' => url('user/password'))));
  }
  else if (drupal_is_denied('mail', $edit['mail'])) {
    form_set_error('mail', t('The e-mail address %email has been denied
access.', array('%email' => $edit['mail'])));
  }

  // If required, validate the uploaded picture.
  if ($file = file_check_upload('picture_upload')) {
    user_validate_picture($file, $edit, $user);
  }
}

function _user_edit_submit($uid, &$edit) {
  $user = user_load(array('uid' => $uid));
  // Delete picture if requested, and if no replacement picture was given.
  if ($edit['picture_delete']) {
    if ($user->picture && file_exists($user->picture)) {
      file_delete($user->picture);
    }
    $edit['picture'] = '';
  }
  if (isset($edit['roles'])) {
    $edit['roles'] = array_filter($edit['roles']);
  }
}

function user_edit($category = 'account') {
  global $user;

  $account = user_load(array('uid' => arg(1)));
  if ($account === FALSE) {
    drupal_set_message(t('The account does not exist or has already been
deleted.'));
    drupal_goto('admin/user/user');
  }
  $edit = $_POST['op'] ? $_POST : (array)$account;

  if (arg(2) == 'delete') {
    if ($edit['confirm']) {
      user_delete($edit, $account->uid);
      drupal_goto('admin/user/user');
    }
    else {
      return drupal_get_form('user_confirm_delete', $account->name, $account-
>uid);
    }
  }
  else if ($_POST['op'] == t('Delete')) {
```

```
    if ($_REQUEST['destination']) {
      $destination = drupal_get_destination();
      unset($_REQUEST['destination']);
    }
    // Note: we redirect from user/uid/edit to user/uid/delete to make the tabs
disappear.
    drupal_goto("user/$account->uid/delete", $destination);
  }

  $form = _user_forms($edit, $account, $category);
  $form['_category'] = array('#type' => 'value', '#value' => $category);
  $form['_account'] = array('#type' => 'value', '#value' => $account);
  $form['submit'] = array('#type' => 'submit', '#value' => t('Submit'),
'#weight' => 30);
  if (user_access('administer users')) {
    $form['delete'] = array('#type' => 'submit', '#value' => t('Delete'),
'#weight' => 31);
  }
  $form['#attributes']['enctype'] = 'multipart/form-data';

  drupal_set_title(check_plain($account->name));
  return $form;
}

function user_confirm_delete($name, $uid) {
  return confirm_form(array(),
    t('Are you sure you want to delete the account %name?', array('%name' =>
$name)),
    'user/'. $uid,
    t('All submissions made by this user will be attributed to the anonymous
account. This action cannot be undone.'),
    t('Delete'), t('Cancel'));
}

/**
 * Delete a user.
 *
 * @param $edit An array of submitted form values.
 * @param $uid The user ID of the user to delete.
 */
function user_delete($edit, $uid) {
  $account = user_load(array('uid' => $uid));
  sess_destroy_uid($uid);
  db_query('DELETE FROM {users} WHERE uid = %d', $uid);
  db_query('DELETE FROM {users_roles} WHERE uid = %d', $uid);
  db_query('DELETE FROM {authmap} WHERE uid = %d', $uid);
  $array = array('%name' => $account->name, '%email' => '<'. $account->mail
.'>');
  watchdog('user', t('Deleted user: %name %email.', $array), WATCHDOG_NOTICE);
  drupal_set_message(t('%name has been deleted.', $array));
  module_invoke_all('user', 'delete', $edit, $account);
}

function user_edit_validate($form_id, $form_values) {
  user_module_invoke('validate', $form_values, $form_values['_account'],
$form_values['_category']);
  // Validate input to ensure that non-privileged users can't alter protected
data.
  if ((!user_access('administer users') &&
array_intersect(array_keys($form_values), array('uid', 'init', 'session'))) ||
(!user_access('administer access control') && isset($form_values['roles']))) {
    $message = t('Detected malicious attempt to alter protected user fields.');
    watchdog('security', $message, WATCHDOG_WARNING);
```

```
    // set this to a value type field
    form_set_error('category', $message);
  }
}

function user_edit_submit($form_id, $form_values) {
  $account = $form_values['_account'];
  $category = $form_values['_category'];
  unset($form_values['_account'], $form_values['op'], $form_values['submit'],
$form_values['delete'], $form_values['form_token'], $form_values['form_id'],
$form_values['_category']);
  user_module_invoke('submit', $form_values, $account, $category);
  user_save($account, $form_values, $category);

  // Delete that user's menu cache:
  cache_clear_all($account->uid .':', 'cache_menu', TRUE);

  // Clear the page cache because pages can contain usernames and/or profile
information:
  cache_clear_all();

  drupal_set_message(t('The changes have been saved.'));
  return 'user/'. $account->uid;
}

function user_view($uid = 0) {
  global $user;

  $account = user_load(array('uid' => $uid));
  if ($account === FALSE || ($account->access == 0 && !user_access('administer
users'))) {
    return drupal_not_found();
  }
  // Retrieve and merge all profile fields:
  $fields = array();
  foreach (module_list() as $module) {
    if ($data = module_invoke($module, 'user', 'view', '', $account)) {
      foreach ($data as $category => $items) {
        foreach ($items as $key => $item) {
          $item['class'] = "$module-". $item['class'];
          $fields[$category][$key] = $item;
        }
      }
    }
  }

  // Let modules change the returned fields - useful for personal privacy
  // controls. Since modules communicate changes by reference, we cannot use
  // module_invoke_all().
  foreach (module_implements('profile_alter') as $module) {
    $function = $module .'_profile_alter';
    $function($account, $fields);
  }

  drupal_set_title(check_plain($account->name));
  return theme('user_profile', $account, $fields);
}

/*** Administrative features ********************************************/

function _user_mail_text($messageid, $variables = array()) {

  // Check if an admin setting overrides the default string.
```

```
  if ($admin_setting = variable_get('user_mail_'. $messageid, FALSE)) {
    return strtr($admin_setting, $variables);
  }
  // No override, return with default strings.
  else {
    switch ($messageid) {
      case 'welcome_subject':
        return t('Account details for !username at !site', $variables);
      case 'welcome_body':
        return t("!username,\n\nThank you for registering at !site. You may now
log in to !login_uri using the following username and password:\n\nusername:
!username\npassword: !password\n\nYou may also log in by clicking on this link
or copying and pasting it in your browser:\n\n!login_url\n\nThis is a one-time
login, so it can be used only once.\n\nAfter logging in, you will be redirected
to !edit_uri so you can change your password.\n\n\n--  !site team",
$variables);
      case 'admin_subject':
        return t('An administrator created an account for you at !site',
$variables);
      case 'admin_body':
        return t("!username,\n\nA site administrator at !site has created an
account for you. You may now log in to !login_uri using the following username
and password:\n\nusername: !username\npassword: !password\n\nYou may also log
in by clicking on this link or copying and pasting it in your
browser:\n\n!login_url\n\nThis is a one-time login, so it can be used only
once.\n\nAfter logging in, you will be redirected to !edit_uri so you can
change your password.\n\n\n--  !site team", $variables);
      case 'approval_subject':
        return t('Account details for !username at !site (pending admin
approval)', $variables);
      case 'approval_body':
        return t("!username,\n\nThank you for registering at !site. Your
application for an account is currently pending approval. Once it has been
granted, you may log in to !login_uri using the following username and
password:\n\nusername: !username\npassword: !password\n\nYou may also log in by
clicking on this link or copying and pasting it in your
browser:\n\n!login_url\n\nThis is a one-time login, so it can be used only
once.\n\nAfter logging in, you may wish to change your password at
!edit_uri\n\n\n--  !site team", $variables);
      case 'pass_subject':
        return t('Replacement login information for !username at !site',
$variables);
      case 'pass_body':
        return t("!username,\n\nA request to reset the password for your
account has been made at !site.\n\nYou may now log in to !uri_brief clicking on
this link or copying and pasting it in your browser:\n\n!login_url\n\nThis is a
one-time login, so it can be used only once. It expires after one day and
nothing will happen if it's not used.\n\nAfter logging in, you will be
redirected to !edit_uri so you can change your password.", $variables);
    }
  }
}

function user_admin_check_user() {
  $form['user'] = array('#type' => 'fieldset', '#title' => t('Username'));
  $form['user']['test'] = array('#type' => 'textfield', '#title' => '',
'#description' => t('Enter a username to check if it will be denied or
allowed.'), '#size' => 30, '#maxlength' => USERNAME_MAX_LENGTH);
  $form['user']['type'] = array('#type' => 'hidden', '#value' => 'user');
  $form['user']['submit'] = array('#type' => 'submit', '#value' => t('Check
username'));
  $form['#base'] = 'user_admin_access_check';
  return $form;
```

```
}

function user_admin_check_mail() {
  $form['mail'] = array('#type' => 'fieldset', '#title' => t('E-mail'));
  $form['mail']['test'] = array('#type' => 'textfield', '#title' => '',
'#description' => t('Enter an e-mail address to check if it will be denied or
allowed.'), '#size' => 30, '#maxlength' => EMAIL_MAX_LENGTH);
  $form['mail']['type'] = array('#type' => 'hidden', '#value' => 'mail');
  $form['mail']['submit'] = array('#type' => 'submit', '#value' => t('Check e-
mail'));
  $form['#base'] = 'user_admin_access_check';
  return $form;
}

function user_admin_check_host() {
  $form['host'] = array('#type' => 'fieldset', '#title' => t('Hostname'));
  $form['host']['test'] = array('#type' => 'textfield', '#title' => '',
'#description' => t('Enter a hostname or IP address to check if it will be
denied or allowed.'), '#size' => 30, '#maxlength' => 64);
  $form['host']['type'] = array('#type' => 'hidden', '#value' => 'host');
  $form['host']['submit'] = array('#type' => 'submit', '#value' => t('Check
hostname'));
  $form['#base'] = 'user_admin_access_check';
  return $form;
}

/**
 * Menu callback: check an access rule
 */
function user_admin_access_check() {
  $output = drupal_get_form('user_admin_check_user');
  $output .= drupal_get_form('user_admin_check_mail');
  $output .= drupal_get_form('user_admin_check_host');
  return $output;
}

function user_admin_access_check_validate($form_id, $form_values) {
  if (empty($form_values['test'])) {
    form_set_error($form_values['type'], t('No value entered. Please enter a
test string and try again.'));
  }
}

function user_admin_access_check_submit($form_id, $form_values) {
  switch ($form_values['type']) {
    case 'user':
      if (drupal_is_denied('user', $form_values['test'])) {
        drupal_set_message(t('The username %name is not allowed.',
array('%name' => $form_values['test'])));
      }
      else {
        drupal_set_message(t('The username %name is allowed.', array('%name' =>
$form_values['test'])));
      }
      break;
    case 'mail':
      if (drupal_is_denied('mail', $form_values['test'])) {
        drupal_set_message(t('The e-mail address %mail is not allowed.',
array('%mail' => $form_values['test'])));
      }
      else {
        drupal_set_message(t('The e-mail address %mail is allowed.',
array('%mail' => $form_values['test'])));
```

```
      }
      break;
    case 'host':
      if (drupal_is_denied('host', $form_values['test'])) {
        drupal_set_message(t('The hostname %host is not allowed.',
array('%host' => $form_values['test'])));
      }
      else {
        drupal_set_message(t('The hostname %host is allowed.', array('%host' =>
$form_values['test'])));
      }
      break;
    default:
      break;
  }
}

/**
 * Menu callback: add an access rule
 */
function user_admin_access_add($mask = NULL, $type = NULL) {
  if ($edit = $_POST) {
    if (!$edit['mask']) {
      form_set_error('mask', t('You must enter a mask.'));
    }
    else {
      $aid = db_next_id('{access}_aid');
      db_query("INSERT INTO {access} (aid, mask, type, status) VALUES ('%s',
'%s', '%s', %d)", $aid, $edit['mask'], $edit['type'], $edit['status']);
      drupal_set_message(t('The access rule has been added.'));
      drupal_goto('admin/user/rules');
    }
  }
  else {
    $edit['mask'] = $mask;
    $edit['type'] = $type;
  }
  return drupal_get_form('user_admin_access_add_form', $edit, t('Add rule'));
}

/**
 * Menu callback: delete an access rule
 */
function user_admin_access_delete_confirm($aid = 0) {
  $access_types = array('user' => t('username'), 'mail' => t('e-mail'), 'host'
=> t('host'));
  $edit = db_fetch_object(db_query('SELECT aid, type, status, mask FROM
{access} WHERE aid = %d', $aid));

  $form = array();
  $form['aid'] = array('#type' => 'hidden', '#value' => $aid);
  $output = confirm_form($form,
                  t('Are you sure you want to delete the @type rule for
%rule?', array('@type' => $access_types[$edit->type], '%rule' => $edit->mask)),
                  'admin/user/rules',
                  t('This action cannot be undone.'),
                  t('Delete'),
                  t('Cancel'));
  return $output;
}

function user_admin_access_delete_confirm_submit($form_id, $form_values) {
  db_query('DELETE FROM {access} WHERE aid = %d', $form_values['aid']);
```

```
  drupal_set_message(t('The access rule has been deleted.'));
  return 'admin/user/rules';
}

/**
 * Menu callback: edit an access rule
 */
function user_admin_access_edit($aid = 0) {
  if ($edit = $_POST) {
    if (!$edit['mask']) {
      form_set_error('mask', t('You must enter a mask.'));
    }
    else {
      db_query("UPDATE {access} SET mask = '%s', type = '%s', status = '%s'
WHERE aid = %d", $edit['mask'], $edit['type'], $edit['status'], $aid);
      drupal_set_message(t('The access rule has been saved.'));
      drupal_goto('admin/user/rules');
    }
  }
  else {
    $edit = db_fetch_array(db_query('SELECT aid, type, status, mask FROM
{access} WHERE aid = %d', $aid));
  }
  return drupal_get_form('user_admin_access_edit_form', $edit, t('Save rule'));
}

function user_admin_access_form($edit, $submit) {
  $form['status'] = array(
    '#type' => 'radios',
    '#title' => t('Access type'),
    '#default_value' => $edit['status'],
    '#options' => array('1' => t('Allow'), '0' => t('Deny')),
  );
  $type_options = array('user' => t('Username'), 'mail' => t('E-mail'), 'host'
=> t('Host'));
  $form['type'] = array(
    '#type' => 'radios',
    '#title' => t('Rule type'),
    '#default_value' => (isset($type_options[$edit['type']]) ? $edit['type'] :
'user'),
    '#options' => $type_options,
  );
  $form['mask'] = array(
    '#type' => 'textfield',
    '#title' => t('Mask'),
    '#size' => 30,
    '#maxlength' => 64,
    '#default_value' => $edit['mask'],
    '#description' => '%: '. t('Matches any number of characters, even zero
characters') .'.<br />_: '. t('Matches exactly one character.'),
    '#required' => TRUE,
  );
  $form['submit'] = array('#type' => 'submit', '#value' => $submit);

  return $form;
}

/**
 * Menu callback: list all access rules
 */
function user_admin_access() {
```

```
  $header = array(array('data' => t('Access type'), 'field' => 'status'),
array('data' => t('Rule type'), 'field' => 'type'), array('data' => t('Mask'),
'field' => 'mask'), array('data' => t('Operations'), 'colspan' => 2));
  $result = db_query("SELECT aid, type, status, mask FROM {access}".
tablesort_sql($header));
  $access_types = array('user' => t('username'), 'mail' => t('e-mail'), 'host'
=> t('host'));
  $rows = array();
  while ($rule = db_fetch_object($result)) {
    $rows[] = array($rule->status ? t('allow') : t('deny'),
$access_types[$rule->type], $rule->mask, l(t('edit'), 'admin/user/rules/edit/'.
$rule->aid), l(t('delete'), 'admin/user/rules/delete/'. $rule->aid));
  }
  if (count($rows) == 0) {
    $rows[] = array(array('data' => '<em>'. t('There are currently no access
rules.') .'</em>', 'colspan' => 5));
  }
  $output .= theme('table', $header, $rows);

  return $output;
}

/**
 * Retrieve an array of roles matching specified conditions.
 *
 * @param $membersonly
 *   Set this to TRUE to exclude the 'anonymous' role.
 * @param $permission
 *   A string containing a permission. If set, only roles containing that
permission are returned.
 *
 * @return
 *   An associative array with the role id as the key and the role name as
value.
 */
function user_roles($membersonly = 0, $permission = 0) {
  $roles = array();

  if ($permission) {
    $result = db_query("SELECT r.* FROM {role} r INNER JOIN {permission} p ON
r.rid = p.rid WHERE p.perm LIKE '%%%s%%' ORDER BY r.name", $permission);
  }
  else {
    $result = db_query('SELECT * FROM {role} ORDER BY name');
  }
  while ($role = db_fetch_object($result)) {
    if (!$membersonly || ($membersonly && $role->rid != DRUPAL_ANONYMOUS_RID))
{
      $roles[$role->rid] = $role->name;
    }
  }
  return $roles;
}

/**
 * Menu callback: administer permissions.
 */
function user_admin_perm($rid = NULL) {
  if (is_numeric($rid)) {
    $result = db_query('SELECT r.rid, p.perm FROM {role} r LEFT JOIN
{permission} p ON r.rid = p.rid WHERE r.rid = %d', $rid);
  }
  else {
```

```
    $result = db_query('SELECT r.rid, p.perm FROM {role} r LEFT JOIN
{permission} p ON r.rid = p.rid ORDER BY name');
  }

  // Compile role array:
  // Add a comma at the end so when searching for a permission, we can
  // always search for "$perm," to make sure we do not confuse
  // permissions that are substrings of each other.
  while ($role = db_fetch_object($result)) {
    $role_permissions[$role->rid] = $role->perm .',';
  }

  if (is_numeric($rid)) {
    $result = db_query('SELECT rid, name FROM {role} r WHERE r.rid = %d ORDER
BY name', $rid);
  }
  else {
    $result = db_query('SELECT rid, name FROM {role} ORDER BY name');
  }

  $role_names = array();
  while ($role = db_fetch_object($result)) {
    $role_names[$role->rid] = $role->name;
  }

  // Render role/permission overview:
  $options = array();
  foreach (module_list(FALSE, FALSE, TRUE) as $module) {
    if ($permissions = module_invoke($module, 'perm')) {
      $form['permission'][] = array(
        '#value' => $module,
      );
      asort($permissions);
      foreach ($permissions as $perm) {
        $options[$perm] = '';
        $form['permission'][$perm] = array('#value' => t($perm));
        foreach ($role_names as $rid => $name) {
          // Builds arrays for checked boxes for each role
          if (strpos($role_permissions[$rid], $perm .',') !== FALSE) {
            $status[$rid][] = $perm;
          }
        }
      }
    }
  }

  // Have to build checkboxes here after checkbox arrays are built
  foreach ($role_names as $rid => $name) {
    $form['checkboxes'][$rid] = array('#type' => 'checkboxes', '#options' =>
$options, '#default_value' => $status[$rid]);
    $form['role_names'][$rid] = array('#value' => $name, '#tree' => TRUE);
  }
  $form['submit'] = array('#type' => 'submit', '#value' => t('Save
permissions'));

  return $form;
}

function theme_user_admin_perm($form) {
  foreach (element_children($form['permission']) as $key) {
    // Don't take form control structures
    if (is_array($form['permission'][$key])) {
```

```
      $row = array();
      // Module name
      if (is_numeric($key)) {
        $row[] = array('data' => t('@module module', array('@module' =>
drupal_render($form['permission'][$key]))), 'class' => 'module', 'id' =>
'module-'. $form['permission'][$key]['#value'], 'colspan' =>
count($form['role_names']) + 1);
      }
      else {
        $row[] = array('data' => drupal_render($form['permission'][$key]),
'class' => 'permission');
        foreach (element_children($form['checkboxes']) as $rid) {
          if (is_array($form['checkboxes'][$rid])) {
            $row[] = array('data' =>
drupal_render($form['checkboxes'][$rid][$key]), 'align' => 'center', 'title' =>
t($key));
          }
        }
      }
      $rows[] = $row;
    }
  }
  $header[] = (t('Permission'));
  foreach (element_children($form['role_names']) as $rid) {
    if (is_array($form['role_names'][$rid])) {
      $header[] = drupal_render($form['role_names'][$rid]);
    }
  }
  $output = theme('table', $header, $rows, array('id' => 'permissions'));
  $output .= drupal_render($form);
  return $output;
}

function user_admin_perm_submit($form_id, $form_values) {
  // Save permissions:
  $result = db_query('SELECT * FROM {role}');
  while ($role = db_fetch_object($result)) {
    if (isset($form_values[$role->rid])) {
      // Delete, so if we clear every checkbox we reset that role;
      // otherwise permissions are active and denied everywhere.
      db_query('DELETE FROM {permission} WHERE rid = %d', $role->rid);
      $form_values[$role->rid] = array_filter($form_values[$role->rid]);
      if (count($form_values[$role->rid])) {
        db_query("INSERT INTO {permission} (rid, perm) VALUES (%d, '%s')",
$role->rid, implode(', ', array_keys($form_values[$role->rid])));
      }
    }
  }

  drupal_set_message(t('The changes have been saved.'));

  // Clear the cached pages and menus:
  menu_rebuild();

}

/**
 * Menu callback: administer roles.
 */
function user_admin_role() {
  $id = arg(4);
  if ($id) {
    if (DRUPAL_ANONYMOUS_RID == $id || DRUPAL_AUTHENTICATED_RID == $id) {
```

```
      drupal_goto('admin/user/roles');
    }
    // Display the edit role form.
    $role = db_fetch_object(db_query('SELECT * FROM {role} WHERE rid = %d',
$id));
    $form['name'] = array(
      '#type' => 'textfield',
      '#title' => t('Role name'),
      '#default_value' => $role->name,
      '#size' => 30,
      '#required' => TRUE,
      '#maxlength' => 64,
      '#description' => t('The name for this role. Example: "moderator",
"editorial board", "site architect".'),
    );
    $form['rid'] = array(
      '#type' => 'value',
      '#value' => $id,
    );
    $form['submit'] = array(
      '#type' => 'submit',
      '#value' => t('Save role'),
    );
    $form['delete'] = array(
      '#type' => 'submit',
      '#value' => t('Delete role'),
    );
  }
  else {
    $form['name'] = array(
      '#type' => 'textfield',
      '#size' => 32,
      '#maxlength' => 64,
    );
    $form['submit'] = array(
      '#type' => 'submit',
      '#value' => t('Add role'),
    );
    $form['#base'] = 'user_admin_role';
  }
  return $form;
}

function user_admin_role_validate($form_id, $form_values) {
  if ($form_values['name']) {
    if ($form_values['op'] == t('Save role')) {
      if (db_result(db_query("SELECT COUNT(*) FROM {role} WHERE name = '%s' AND
rid != %d", $form_values['name'], $form_values['rid']))) {
        form_set_error('name', t('The role name %name already exists. Please
choose another role name.', array('%name' => $form_values['name'])));
      }
    }
    else if ($form_values['op'] == t('Add role')) {
      if (db_result(db_query("SELECT COUNT(*) FROM {role} WHERE name = '%s'",
$form_values['name']))) {
        form_set_error('name', t('The role name %name already exists. Please
choose another role name.', array('%name' => $form_values['name'])));
      }
    }
  }
  else {
    form_set_error('name', t('You must specify a valid role name.'));
  }
```

```
}

function user_admin_role_submit($form_id, $form_values) {
  if ($form_values['op'] == t('Save role')) {
    db_query("UPDATE {role} SET name = '%s' WHERE rid = %d",
$form_values['name'], $form_values['rid']);
    drupal_set_message(t('The role has been renamed.'));
  }
  else if ($form_values['op'] == t('Delete role')) {
    db_query('DELETE FROM {role} WHERE rid = %d', $form_values['rid']);
    db_query('DELETE FROM {permission} WHERE rid = %d', $form_values['rid']);
    // Update the users who have this role set:
    db_query('DELETE FROM {users_roles} WHERE rid = %d', $form_values['rid']);

    drupal_set_message(t('The role has been deleted.'));
  }
  else if ($form_values['op'] == t('Add role')) {
    db_query("INSERT INTO {role} (name) VALUES ('%s')", $form_values['name']);
    drupal_set_message(t('The role has been added.'));
  }
  return 'admin/user/roles';
}

function theme_user_admin_new_role($form) {
  $header = array(t('Name'), array('data' => t('Operations'), 'colspan' => 2));
  foreach (user_roles() as $rid => $name) {
    $edit_permissions = l(t('edit permissions'), 'admin/user/access/'. $rid);
    if (!in_array($rid, array(DRUPAL_ANONYMOUS_RID, DRUPAL_AUTHENTICATED_RID)))
{
      $rows[] = array($name, l(t('edit role'), 'admin/user/roles/edit/'. $rid),
$edit_permissions);
    }
    else {
      $rows[] = array($name, t('locked'), $edit_permissions);
    }
  }
  $rows[] = array(drupal_render($form['name']), array('data' =>
drupal_render($form['submit']), colspan => 2));

  $output = drupal_render($form);
  $output .= theme('table', $header, $rows);

  return $output;
}

function user_admin_account() {
  $filter = user_build_filter_query();

  $header = array(
    array(),
    array('data' => t('Username'), 'field' => 'u.name'),
    array('data' => t('Status'), 'field' => 'u.status'),
    t('Roles'),
    array('data' => t('Member for'), 'field' => 'u.created', 'sort' => 'desc'),
    array('data' => t('Last access'), 'field' => 'u.access'),
    t('Operations')
  );

  $sql = 'SELECT DISTINCT u.uid, u.name, u.status, u.created, u.access FROM
{users} u LEFT JOIN {users_roles} ur ON u.uid = ur.uid '. $filter['join'] .'
WHERE u.uid != 0 '. $filter['where'];
  $sql .= tablesort_sql($header);
  $result = pager_query($sql, 50, 0, NULL, $filter['args']);
```

```
  $form['options'] = array(
    '#type' => 'fieldset',
    '#title' => t('Update options'),
    '#prefix' => '<div class="container-inline">',
    '#suffix' => '</div>',
  );
  $options = array();
  foreach (module_invoke_all('user_operations') as $operation => $array) {
    $options[$operation] = $array['label'];
  }
  $form['options']['operation'] = array(
    '#type' => 'select',
    '#options' => $options,
    '#default_value' => 'unblock',
  );
  $form['options']['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Update'),
  );

  $destination = drupal_get_destination();

  $status = array(t('blocked'), t('active'));
  $roles = user_roles(1);

  while ($account = db_fetch_object($result)) {
    $accounts[$account->uid] = '';
    $form['name'][$account->uid] = array('#value' => theme('username',
$account));
    $form['status'][$account->uid] =  array('#value' => $status[$account-
>status]);
    $users_roles = array();
    $roles_result = db_query('SELECT rid FROM {users_roles} WHERE uid = %d',
$account->uid);
    while ($user_role = db_fetch_object($roles_result)) {
      $users_roles[] = $roles[$user_role->rid];
    }
    asort($users_roles);
    $form['roles'][$account->uid][0] = array('#value' => theme('item_list',
$users_roles));
    $form['member_for'][$account->uid] = array('#value' =>
format_interval(time() - $account->created));
    $form['last_access'][$account->uid] =  array('#value' => $account->access ?
t('@time ago', array('@time' => format_interval(time() - $account->access))) :
t('never'));
    $form['operations'][$account->uid] = array('#value' => l(t('edit'),
"user/$account->uid/edit", array(), $destination));
  }
  $form['accounts'] = array(
    '#type' => 'checkboxes',
    '#options' => $accounts
  );
  $form['pager'] = array('#value' => theme('pager', NULL, 50, 0));

  return $form;
}

/**
 * Theme user administration overview.
 */
function theme_user_admin_account($form) {
  // Overview table:
```

```
  $header = array(
    theme('table_select_header_cell'),
    array('data' => t('Username'), 'field' => 'u.name'),
    array('data' => t('Status'), 'field' => 'u.status'),
    t('Roles'),
    array('data' => t('Member for'), 'field' => 'u.created', 'sort' => 'desc'),
    array('data' => t('Last access'), 'field' => 'u.access'),
    t('Operations')
  );

  $output = drupal_render($form['options']);
  if (isset($form['name']) && is_array($form['name'])) {
    foreach (element_children($form['name']) as $key) {
      $rows[] = array(
        drupal_render($form['accounts'][$key]),
        drupal_render($form['name'][$key]),
        drupal_render($form['status'][$key]),
        drupal_render($form['roles'][$key]),
        drupal_render($form['member_for'][$key]),
        drupal_render($form['last_access'][$key]),
        drupal_render($form['operations'][$key]),
      );
    }
  }
  else  {
    $rows[] = array(array('data' => t('No users available.'), 'colspan' =>
'7'));
  }

  $output .= theme('table', $header, $rows);
  if ($form['pager']['#value']) {
    $output .= drupal_render($form['pager']);
  }

  $output .= drupal_render($form);

  return $output;
}

/**
 * Submit the user administration update form.
 */
function user_admin_account_submit($form_id, $form_values) {
  $operations = module_invoke_all('user_operations');
  $operation = $operations[$form_values['operation']];
  // Filter out unchecked accounts.
  $accounts = array_filter($form_values['accounts']);
  if ($function = $operation['callback']) {
    // Add in callback arguments if present.
    if (isset($operation['callback arguments'])) {
      $args = array_merge(array($accounts), $operation['callback arguments']);
    }
    else {
      $args = array($accounts);
    }
    call_user_func_array($function, $args);

    cache_clear_all('*', 'cache_menu', TRUE);
    drupal_set_message(t('The update has been performed.'));
  }
}

function user_admin_account_validate($form_id, $form_values) {
```

```
  $form_values['accounts'] = array_filter($form_values['accounts']);
  if (count($form_values['accounts']) == 0) {
    form_set_error('', t('No users selected.'));
  }
}

/**
 * Implementation of hook_user_operations().
 */
function user_user_operations() {
  global $form_values;

  $operations = array(
    'unblock' => array(
      'label' => t('Unblock the selected users'),
      'callback' => 'user_user_operations_unblock',
    ),
    'block' => array(
      'label' => t('Block the selected users'),
      'callback' => 'user_user_operations_block',
    ),
    'delete' => array(
      'label' => t('Delete the selected users'),
    ),
  );

  if (user_access('administer access control')) {
    $roles = user_roles(1);
    unset($roles[DRUPAL_AUTHENTICATED_RID]);  // Can't edit authenticated role.

    $add_roles = array();
    foreach ($roles as $key => $value) {
      $add_roles['add_role-'. $key] = $value;
    }

    $remove_roles = array();
    foreach ($roles as $key => $value) {
      $remove_roles['remove_role-'. $key] = $value;
    }

    if (count($roles)) {
      $role_operations = array(
        t('Add a role to the selected users') => array(
          'label' => $add_roles,
        ),
        t('Remove a role from the selected users') => array(
          'label' => $remove_roles,
        ),
      );

      $operations += $role_operations;
    }
  }

  // If the form has been posted, we need to insert the proper data for role
editing if necessary.
  if ($form_values) {
    $operation_rid = explode('-', $form_values['operation']);
    $operation = $operation_rid[0];
    $rid = $operation_rid[1];
    if ($operation == 'add_role' || $operation == 'remove_role') {
      if (user_access('administer access control')) {
        $operations[$form_values['operation']] = array(
```

```
          'callback' => 'user_multiple_role_edit',
          'callback arguments' => array($operation, $rid),
        );
      }
      else {
        watchdog('security', t('Detected malicious attempt to alter protected
user fields.'), WATCHDOG_WARNING);
        return;
      }
    }
  }

  return $operations;
}

/**
 * Callback function for admin mass unblocking users.
 */
function user_user_operations_unblock($accounts) {
  foreach ($accounts as $uid) {
    $account = user_load(array('uid' => (int)$uid));
    // Skip unblocking user if they are already unblocked.
    if ($account !== FALSE && $account->status == 0) {
      user_save($account, array('status' => 1));
    }
  }
}

/**
 * Callback function for admin mass blocking users.
 */
function user_user_operations_block($accounts) {
  foreach ($accounts as $uid) {
    $account = user_load(array('uid' => (int)$uid));
    // Skip blocking user if they are already blocked.
    if ($account !== FALSE && $account->status == 1) {
      user_save($account, array('status' => 0));
    }
  }
}

/**
 * Callback function for admin mass adding/deleting a user role.
 */
function user_multiple_role_edit($accounts, $operation, $rid) {
  // The role name is not necessary as user_save() will reload the user
  // object, but some modules' hook_user() may look at this first.
  $role_name = db_result(db_query('SELECT name FROM {role} WHERE rid = %d',
$rid));

  switch ($operation) {
    case 'add_role':
      foreach ($accounts as $uid) {
        $account = user_load(array('uid' => (int)$uid));
        // Skip adding the role to the user if they already have it.
        if ($account !== FALSE && !isset($account->roles[$rid])) {
          $roles = $account->roles + array($rid => $role_name);
          user_save($account, array('roles' => $roles));
        }
      }
      break;
    case 'remove_role':
      foreach ($accounts as $uid) {
```

```
        $account = user_load(array('uid' => (int)$uid));
        // Skip removing the role from the user if they already don't have it.
        if ($account !== FALSE && isset($account->roles[$rid])) {
          $roles = array_diff($account->roles, array($rid => $role_name));
          user_save($account, array('roles' => $roles));
        }
      }
      break;
  }
}

function user_multiple_delete_confirm() {
  $edit = $_POST;

  $form['accounts'] = array('#prefix' => '<ul>', '#suffix' => '</ul>', '#tree'
=> TRUE);
  // array_filter returns only elements with TRUE values
  foreach (array_filter($edit['accounts']) as $uid => $value) {
    $user = db_result(db_query('SELECT name FROM {users} WHERE uid = %d',
$uid));
    $form['accounts'][$uid] = array('#type' => 'hidden', '#value' => $uid,
'#prefix' => '<li>', '#suffix' => check_plain($user) ."</li>\n");
  }
  $form['operation'] = array('#type' => 'hidden', '#value' => 'delete');

  return confirm_form($form,
                      t('Are you sure you want to delete these users?'),
                      'admin/user/user', t('This action cannot be undone.'),
                      t('Delete all'), t('Cancel'));
}

function user_multiple_delete_confirm_submit($form_id, $form_values) {
  if ($form_values['confirm']) {
    foreach ($form_values['accounts'] as $uid => $value) {
      user_delete($form_values, $uid);
    }
    drupal_set_message(t('The users have been deleted.'));
  }
  return 'admin/user/user';
}

function user_admin_settings() {
  // User registration settings.
  $form['registration'] = array('#type' => 'fieldset', '#title' => t('User
registration settings'));
  $form['registration']['user_register'] = array('#type' => 'radios', '#title'
=> t('Public registrations'), '#default_value' => variable_get('user_register',
1), '#options' => array(t('Only site administrators can create new user
accounts.'), t('Visitors can create accounts and no administrator approval is
required.'), t('Visitors can create accounts but administrator approval is
required.')));
  $form['registration']['user_email_verification'] = array('#type' =>
'checkbox', '#title' => t('Require e-mail verification when a visitor creates
an account'), '#default_value' => variable_get('user_email_verification',
TRUE), '#description' => t('If this box is checked, new users will be required
to validate their e-mail address prior to logging into to the site, and will be
assigned a system-generated password. With it unchecked, users will be logged
in immediately upon registering, and may select their own passwords during
registration.'));
  $form['registration']['user_registration_help'] = array('#type' =>
'textarea', '#title' => t('User registration guidelines'), '#default_value' =>
variable_get('user_registration_help', ''), '#description' => t("This text is
```

```
displayed at the top of the user registration form. It's useful for helping or
instructing your users."));

  // User e-mail settings.
  $form['email'] = array('#type' => 'fieldset', '#title' => t('User e-mail
settings'));
  $form['email']['user_mail_welcome_subject'] = array('#type' => 'textfield',
'#title' => t('Subject of welcome e-mail'), '#default_value' =>
_user_mail_text('welcome_subject'), '#maxlength' => 180, '#description' =>
t('Customize the subject of your welcome e-mail, which is sent to new members
upon registering.') .' '. t('Available variables are:') .' !username, !site,
!password, !uri, !uri_brief, !mailto, !date, !login_uri, !edit_uri,
!login_url.');
  $form['email']['user_mail_welcome_body'] = array('#type' => 'textarea',
'#title' => t('Body of welcome e-mail'), '#default_value' =>
_user_mail_text('welcome_body'), '#rows' => 15, '#description' => t('Customize
the body of the welcome e-mail, which is sent to new members upon
registering.') .' '. t('Available variables are:') .' !username, !site,
!password, !uri, !uri_brief, !mailto, !login_uri, !edit_uri, !login_url.');
  $form['email']['user_mail_admin_subject'] = array('#type' => 'textfield',
'#title' => t('Subject of welcome e-mail (user created by administrator)'),
'#default_value' => _user_mail_text('admin_subject'), '#maxlength' => 180,
'#description' => t('Customize the subject of your welcome e-mail, which is
sent to new member accounts created by an administrator.') .' '. t('Available
variables are:') .' !username, !site, !password, !uri, !uri_brief, !mailto,
!date, !login_uri, !edit_uri, !login_url.');
  $form['email']['user_mail_admin_body'] = array('#type' => 'textarea',
'#title' => t('Body of welcome e-mail (user created by administrator)'),
'#default_value' => _user_mail_text('admin_body'), '#rows' => 15,
'#description' => t('Customize the body of the welcome e-mail, which is sent to
new member accounts created by an administrator.') .' '. t('Available variables
are:') .' !username, !site, !password, !uri, !uri_brief, !mailto, !login_uri,
!edit_uri, !login_url.');
  $form['email']['user_mail_approval_subject'] = array('#type' => 'textfield',
'#title' => t('Subject of welcome e-mail (awaiting admin approval)'),
'#default_value' => _user_mail_text('approval_subject'), '#maxlength' => 180,
'#description' => t('Customize the subject of your awaiting approval welcome e-
mail, which is sent to new members upon registering.') .' '. t('Available
variables are:') .' !username, !site, !password, !uri, !uri_brief, !mailto,
!date, !login_uri, !edit_uri, !login_url.');
  $form['email']['user_mail_approval_body'] = array('#type' => 'textarea',
'#title' => t('Body of welcome e-mail (awaiting admin approval)'),
'#default_value' => _user_mail_text('approval_body'), '#rows' => 15,
'#description' => t('Customize the body of the awaiting approval welcome e-
mail, which is sent to new members upon registering.') .' '. t('Available
variables are:') .' !username, !site, !password, !uri, !uri_brief, !mailto,
!login_uri, !edit_uri, !login_url.');
  $form['email']['user_mail_pass_subject'] = array('#type' => 'textfield',
'#title' => t('Subject of password recovery e-mail'), '#default_value' =>
_user_mail_text('pass_subject'), '#maxlength' => 180, '#description' =>
t('Customize the subject of your forgotten password e-mail.') .' '.
t('Available variables are:') .' !username, !site, !login_url, !uri,
!uri_brief, !mailto, !date, !login_uri, !edit_uri.');
  $form['email']['user_mail_pass_body'] = array('#type' => 'textarea', '#title'
=> t('Body of password recovery e-mail'), '#default_value' =>
_user_mail_text('pass_body'), '#rows' => 15, '#description' => t('Customize the
body of the forgotten password e-mail.') .' '. t('Available variables are:') .'
!username, !site, !login_url, !uri, !uri_brief, !mailto, !login_uri,
!edit_uri.');

  // If picture support is enabled, check whether the picture directory exists:
  if (variable_get('user_pictures', 0)) {
```

```
    $picture_path = file_create_path(variable_get('user_picture_path',
'pictures'));
    file_check_directory($picture_path, 1, 'user_picture_path');
  }

  $form['pictures'] = array('#type' => 'fieldset', '#title' => t('Pictures'));
  $form['pictures']['user_pictures'] = array('#type' => 'radios', '#title' =>
t('Picture support'), '#default_value' => variable_get('user_pictures', 0),
'#options' => array(t('Disabled'), t('Enabled')), '#description' => t('Enable
picture support.'));
  $form['pictures']['user_picture_path'] = array('#type' => 'textfield',
'#title' => t('Picture image path'), '#default_value' =>
variable_get('user_picture_path', 'pictures'), '#size' => 30, '#maxlength' =>
255, '#description' => t('Subdirectory in the directory %dir where pictures
will be stored.', array('%dir' => file_directory_path() .'/')));
  $form['pictures']['user_picture_default'] = array('#type' => 'textfield',
'#title' => t('Default picture'), '#default_value' =>
variable_get('user_picture_default', ''), '#size' => 30, '#maxlength' => 255,
'#description' => t('URL of picture to display for users with no custom picture
selected. Leave blank for none.'));
  $form['pictures']['user_picture_dimensions'] = array('#type' => 'textfield',
'#title' => t('Picture maximum dimensions'), '#default_value' =>
variable_get('user_picture_dimensions', '85x85'), '#size' => 15, '#maxlength'
=> 10, '#description' => t('Maximum dimensions for pictures, in pixels.'));
  $form['pictures']['user_picture_file_size'] = array('#type' => 'textfield',
'#title' => t('Picture maximum file size'), '#default_value' =>
variable_get('user_picture_file_size', '30'), '#size' => 15, '#maxlength' =>
10, '#description' => t('Maximum file size for pictures, in kB.'));
  $form['pictures']['user_picture_guidelines'] = array('#type' => 'textarea',
'#title' => t('Picture guidelines'), '#default_value' =>
variable_get('user_picture_guidelines', ''), '#description' => t("This text is
displayed at the picture upload form in addition to the default guidelines.
It's useful for helping or instructing your users."));

  return system_settings_form($form);
}

function user_admin($callback_arg = '') {
  $op = isset($_POST['op']) ? $_POST['op'] : $callback_arg;

  switch ($op) {
    case 'search':
    case t('Search'):
      $output = drupal_get_form('search_form', url('admin/user/search'),
$_POST['keys'], 'user') . search_data($_POST['keys'], 'user');
      break;
    case t('Create new account'):
    case 'create':
      $output = drupal_get_form('user_register');
      break;
    default:
      if ($_POST['accounts'] && $_POST['operation'] == 'delete') {
        $output = drupal_get_form('user_multiple_delete_confirm');
      }
      else {
        $output = drupal_get_form('user_filter_form');
        $output .= drupal_get_form('user_admin_account');
      }
  }
  return $output;
}

/**
```

```
 * Implementation of hook_help().
 */
function user_help($section) {
  global $user;

  switch ($section) {
    case 'admin/help#user':
      $output = '<p>'. t('The user module allows users to register, login, and
log out. Users benefit from being able to sign on because it associates content
they create with their account and allows various permissions to be set for
their roles. The user module supports user roles which can setup fine grained
permissions allowing each role to do only what the administrator wants them to.
Each user is assigned to one or more roles. By default there are two roles
<em>anonymous</em> - a user who has not logged in, and <em>authenticated</em> a
user who has signed up and who has been authorized.') .'</p>';
      $output .= '<p>'. t('Users can use their own name or handle and can fine
tune some personal configuration settings through their individual my account
page. Registered users need to authenticate by supplying either a local
username and password, or a remote username and password such as DelphiForums
ID, or one from a Drupal powered website. A visitor accessing your website is
assigned an unique ID, the so-called session ID, which is stored in a cookie.
For security\'s sake, the cookie does not contain personal information but acts
as a key to retrieve the information stored on your server.') .'</p>';
      $output .= '<p>'. t('For more information please read the configuration
and customization handbook <a href="@user">User page</a>.', array('@user' =>
'http://drupal.org/handbook/modules/user/')) .'</p>';
      return $output;
    case 'admin/user/user':
      return '<p>'. t('Drupal allows users to register, login, log out,
maintain user profiles, etc. Users of the site may not use their own names to
post content until they have signed up for a user account.') .'</p>';
    case 'admin/user/user/create':
    case 'admin/user/user/account/create':
      return '<p>'. t('This web page allows the administrators to register a
new users by hand. Note that you cannot have a user where either the e-mail
address or the username match another user in the system.') .'</p>';
    case 'admin/user/rules':
      return '<p>'. t('Set up username and e-mail address access rules for new
<em>and</em> existing accounts (currently logged in accounts will not be logged
out). If a username or e-mail address for an account matches any deny rule, but
not an allow rule, then the account will not be allowed to be created or to log
in. A host rule is effective for every page view, not just registrations.')
.'</p>';
    case 'admin/user/access':
      return '<p>'. t('Permissions let you control what users can do on your
site. Each user role (defined on the <a href="@role">user roles page</a>) has
its own set of permissions. For example, you could give users classified as
"Administrators" permission to "administer nodes" but deny this power to
ordinary, "authenticated" users. You can use permissions to reveal new features
to privileged users (those with subscriptions, for example). Permissions also
allow trusted users to share the administrative burden of running a busy
site.', array('@role' => url('admin/user/roles'))) .'</p>';
    case 'admin/user/roles':
      return t('<p>Roles allow you to fine tune the security and administration
of Drupal. A role defines a group of users that have certain privileges as
defined in <a href="@permissions">user permissions</a>. Examples of roles
include: anonymous user, authenticated user, moderator, administrator and so
on. In this area you will define the <em>role names</em> of the various roles.
To delete a role choose "edit".</p><p>By default, Drupal comes with two user
roles:</p>
      <ul>
      <li>Anonymous user: this role is used for users that don\'t have a user
account or that are not authenticated.</li>
```

```
      <li>Authenticated user: this role is automatically granted to all logged
in users.</li>
      </ul>', array('@permissions' => url('admin/user/access')));
    case 'admin/user/search':
      return '<p>'. t('Enter a simple pattern ("*" may be used as a wildcard
match) to search for a username. For example, one may search for "br" and
Drupal might return "brian", "brad", and "brenda".') .'</p>';
    case 'user/help#user':
      $site = variable_get('site_name', 'Drupal');

      $affiliates = user_auth_help_links();
      if (count($affiliates)) {
        $affiliate_info = implode(', ', user_auth_help_links());
      }
      else {
        $affiliate_info = t('one of our affiliates');
      }

      $output = t('
      <h3>Distributed authentication<a id="da"></a></h3>
      <p>One of the more tedious moments in visiting a new website is filling
out the registration form. Here at @site, you do not have to fill out a
registration form if you are already a member of !affiliate-info. This
capability is called <em>distributed authentication</em>, and <a
href="@drupal">Drupal</a>, the software which powers @site, fully supports
it.</p>
      <p>Distributed authentication enables a new user to input a username and
password into the login box, and immediately be recognized, even if that user
never registered at @site. This works because Drupal knows how to communicate
with external registration databases. For example, lets say that new user
\'Joe\' is already a registered member of <a href="@delphi-forums">Delphi
Forums</a>. Drupal informs Joe on registration and login screens that he may
login with his Delphi ID instead of registering with @site. Joe likes that
idea, and logs in with a username of joe@remote.delphiforums.com and his usual
Delphi password. Drupal then contacts the <em>remote.delphiforums.com</em>
server behind the scenes (usually using <a href="@xml">XML-RPC</a>, <a
href="@http-post">HTTP POST</a>, or <a href="@soap">SOAP</a>) and asks: "Is the
password for user Joe correct?". If Delphi replies yes, then we create a new
@site account for Joe and log him into it. Joe may keep on logging into @site
in the same manner, and he will always be logged into the same account.</p>',
array('!affiliate-info' => $affiliate_info, '@site' => $site, '@drupal' =>
'http://drupal.org', '@delphi-forums' => 'http://www.delphiforums.com', '@xml'
=> 'http://www.xmlrpc.com', '@http-post' => 'http://www.w3.org/Protocols/',
'@soap' => 'http://www.soapware.org'));

        foreach (module_list() as $module) {
          if (module_hook($module, 'auth')) {
            $output .= "<h4><a id=\"$module\"></a>". module_invoke($module,
'info', 'name') .'</h4>';
            $output .= module_invoke($module, 'help', "user/help#$module");
          }
        }

        return $output;
  }

}

/**
 * Menu callback; Prints user-specific help information.
 */
function user_help_page() {
  return user_help('user/help#user');
```

```
}

/**
 * Retrieve a list of all user setting/information categories and sort them by
weight.
 */
function _user_categories($account) {
  $categories = array();

  foreach (module_list() as $module) {
    if ($data = module_invoke($module, 'user', 'categories', NULL, $account,
'')) {
      $categories = array_merge($data, $categories);
    }
  }

  usort($categories, '_user_sort');

  return $categories;
}

function _user_sort($a, $b) {
  return $a['weight'] < $b['weight'] ? -1 : ($a['weight'] > $b['weight'] ? 1 :
($a['title'] < $b['title'] ? -1 : 1));
}

/**
 * Retrieve a list of all form elements for the specified category.
 */
function _user_forms(&$edit, $account, $category, $hook = 'form') {
  $groups = array();
  foreach (module_list() as $module) {
    if ($data = module_invoke($module, 'user', $hook, $edit, $account,
$category)) {
      $groups = array_merge_recursive($data, $groups);
    }
  }
  uasort($groups, '_user_sort');

  return empty($groups) ? FALSE : $groups;
}

/**
 * Retrieve a pipe delimited string of autocomplete suggestions for existing
users
 */
function user_autocomplete($string = '') {
  $matches = array();
  if ($string) {
    $result = db_query_range("SELECT name FROM {users} WHERE LOWER(name) LIKE
LOWER('%s%%')", $string, 0, 10);
    while ($user = db_fetch_object($result)) {
      $matches[$user->name] = check_plain($user->name);
   }
  }
  print drupal_to_js($matches);
  exit();
}

/**
 * List user administration filters that can be applied.
 */
function user_filters() {
```

```
  // Regular filters
  $filters = array();
  $roles = user_roles(1);
  unset($roles[DRUPAL_AUTHENTICATED_RID]); // Don't list authorized role.
  if (count($roles)) {
    $filters['role'] = array('title' => t('role'),

                          'where' => "ur.rid = %d",
                          'options' => $roles,
                        );
  }

  $options = array();
  $t_module = t('module');
  foreach (module_list() as $module) {
    if ($permissions = module_invoke($module, 'perm')) {
      asort($permissions);
      foreach ($permissions as $permission) {
        $options["$module $t_module"][$permission] = t($permission);
      }
    }
  }
  ksort($options);
  $filters['permission'] = array('title' => t('permission'),
                             'join' => 'LEFT JOIN {permission} p ON ur.rid =
p.rid',
                             'where' => " ((p.perm IS NOT NULL AND p.perm LIKE
'%%%s%%') OR u.uid = 1) ",
                             'options' => $options,
                           );

  $filters['status'] = array('title' => t('status'),
                          'where' => 'u.status = %d',
                          'options' => array(1 => t('active'), 0 =>
t('blocked')),
                        );
  return $filters;
}

/**
 * Build query for user administration filters based on session.
 */
function user_build_filter_query() {
  $filters = user_filters();

  // Build query
  $where = $args = $join = array();
  foreach ($_SESSION['user_overview_filter'] as $filter) {
    list($key, $value) = $filter;
    // This checks to see if this permission filter is an enabled permission
for the authenticated role.
    // If so, then all users would be listed, and we can skip adding it to the
filter query.
    if ($key == 'permission') {
      $account = new stdClass();
      $account->uid = 'user_filter';
      $account->roles = array(DRUPAL_AUTHENTICATED_RID => 1);
      if (user_access($value, $account)) {
        continue;
      }
    }
    $where[] = $filters[$key]['where'];
    $args[] = $value;
```

```
    $join[] = $filters[$key]['join'];
  }
  $where = count($where) ? 'AND '. implode(' AND ', $where) : '';
  $join = count($join) ? ' '. implode(' ', array_unique($join)) : '';

  return array('where' => $where,
           'join' => $join,
           'args' => $args,
         );
}

/**
 * Return form for user administration filters.
 */
function user_filter_form() {
  $session = &$_SESSION['user_overview_filter'];
  $session = is_array($session) ? $session : array();
  $filters = user_filters();

  $i = 0;
  $form['filters'] = array('#type' => 'fieldset',
                       '#title' => t('Show only users where'),
                       '#theme' => 'user_filters',
                     );
  foreach ($session as $filter) {
    list($type, $value) = $filter;
    $string = ($i++ ? '<em>and</em> where <strong>%a</strong> is
<strong>%b</strong>' : '<strong>%a</strong> is <strong>%b</strong>');
    // Merge an array of arrays into one if necessary.
    $options = $type == 'permission' ? call_user_func_array('array_merge',
$filters[$type]['options']) : $filters[$type]['options'];
    $form['filters']['current'][] = array('#value' => t($string, array('%a' =>
$filters[$type]['title'] , '%b' => $options[$value])));
  }

  foreach ($filters as $key => $filter) {
    $names[$key] = $filter['title'];
    $form['filters']['status'][$key] = array('#type' => 'select',
                                         '#options' => $filter['options'],
                                       );
  }

  $form['filters']['filter'] = array('#type' => 'radios',
                                 '#options' => $names,
                               );
  $form['filters']['buttons']['submit'] = array('#type' => 'submit',
                                            '#value' => (count($session) ?
t('Refine') : t('Filter'))
                                          );
  if (count($session)) {
    $form['filters']['buttons']['undo'] = array('#type' => 'submit',
                                            '#value' => t('Undo')
                                          );
    $form['filters']['buttons']['reset'] = array('#type' => 'submit',
                                             '#value' => t('Reset')
                                           );
  }

  return $form;
}

/**
 * Theme user administration filter form.
```

```
 */
function theme_user_filter_form($form) {
  $output = '<div id="user-admin-filter">';
  $output .= drupal_render($form['filters']);
  $output .= '</div>';
  $output .= drupal_render($form);
  return $output;
}

/**
 * Theme user administration filter selector.
 */
function theme_user_filters($form) {
  $output = '<ul class="clear-block">';
  if (sizeof($form['current'])) {
    foreach (element_children($form['current']) as $key) {
      $output .= '<li>'. drupal_render($form['current'][$key]) .'</li>';
    }
  }

  $output .= '<li><dl class="multiselect">'. (sizeof($form['current']) ?
'<dt><em>'. t('and') .'</em> '. t('where') .'</dt>' : '') .'<dd class="a">';
  foreach (element_children($form['filter']) as $key) {
    $output .= drupal_render($form['filter'][$key]);
  }
  $output .= '</dd>';

  $output .= '<dt>'. t('is') .'</dt><dd class="b">';

  foreach (element_children($form['status']) as $key) {
    $output .= drupal_render($form['status'][$key]);
  }
  $output .= '</dd>';

  $output .= '</dl>';
  $output .= '<div class="container-inline" id="user-admin-buttons">'.
drupal_render($form['buttons']) .'</div>';
  $output .= '</li></ul>';

  return $output;
}

/**
 * Process result from user administration filter form.
 */
function user_filter_form_submit($form_id, $form_values) {
  $op = $form_values['op'];
  $filters = user_filters();
  switch ($op) {
    case t('Filter'): case t('Refine'):
      if (isset($form_values['filter'])) {
        $filter = $form_values['filter'];
        // Merge an array of arrays into one if necessary.
        $options = $filter == 'permission' ?
call_user_func_array('array_merge', $filters[$filter]['options']) :
$filters[$filter]['options'];
        if (isset($options[$form_values[$filter]])) {
          $_SESSION['user_overview_filter'][] = array($filter,
$form_values[$filter]);
        }
      }
      break;
    case t('Undo'):
```

```
      array_pop($_SESSION['user_overview_filter']);
      break;
    case t('Reset'):
      $_SESSION['user_overview_filter'] = array();
      break;
    case t('Update'):
      return;
  }

  return 'admin/user/user';
}


function user_forms() {
  $forms['user_admin_access_add_form']['callback'] = 'user_admin_access_form';
  $forms['user_admin_access_edit_form']['callback'] = 'user_admin_access_form';
  $forms['user_admin_new_role']['callback'] = 'user_admin_role';
  return $forms;
}
```

USER.CSS

```
/* $Id: user.css,v 1.4 2006/12/30 07:45:31 dries Exp $ */

#permissions td.module {
  font-weight: bold;
}
#permissions td.permission {
  padding-left: 1.5em;
}
#access-rules .access-type, #access-rules .rule-type {
  margin-right: 1em;
  float: left;
}
#access-rules .access-type .form-item, #access-rules .rule-type .form-item {
  margin-top: 0;
}
#access-rules .mask {
  clear: both;
}
#user-login-form {
  text-align: center;
}
#user-admin-filter ul {
  list-style-type: none;
  padding: 0;
  margin: 0;
  width: 100%;
}
#user-admin-buttons {
  float: left;
  margin-left: 0.5em;
  clear: right;
}

/* Generated by user.module but used by profile.module: */
.profile {
  clear: both;
  margin: 1em 0;
}

/*
```

```
.profile .picture {
  float: center;
  margin: 1em 1em 1em 1em;
}
*/
.profile dt {
  margin: 1em 0 0.2em 0;
  font-weight: bold;
}
.profile dd {
  margin:0;
}

#career{
width: 300px;
/*width: 400px;*/
margin: 5px;
padding-left: 5px;
float: left;
}

#personal{
width: 300px;
/*width: 400px;*/
margin: 5px;
float: left;
}

.profile .picture{
width: 300px;
margin: 5px;
padding: 5px;
float: left;
}

#college{
width: 300px;
/*width: 400px;*/
/*padding-left: 300px;
padding-right: 300px;*/
margin: 5px;
float: left;
}
```

PROFILE.MODULE

```
<?php
// $Id: profile.module,v 1.189.2.1 2007/01/23 19:09:58 dries Exp $

/**
 * @file
 * Support for configurable user profiles.
 */

/**
 * Private field, content only available to privileged users.
 */
define('PROFILE_PRIVATE', 1);

/**
 * Public field, content shown on profile page but not used on member
list pages.
```

```
 */
define('PROFILE_PUBLIC', 2);

/**
 * Public field, content shown on profile page and on member list
pages.
 */
define('PROFILE_PUBLIC_LISTINGS', 3);

/**
 * Hidden profile field, only accessible by administrators, modules and
themes.
 */
define('PROFILE_HIDDEN', 4);

/**
 * Implementation of hook_help().
 */
function profile_help($section) {
  switch ($section) {
    case 'admin/help#profile':
      $output = '<p>'. t('The profile module allows you to define
custom fields (such as country, real name, age, ...) in the user
profile. This permits users of a site to share more information about
themselves, and can help community-based sites to organize users around
profile fields.') .'</p>';
      $output .= t('<p>The following types of fields can be added to
the user profile:</p>
<ul>
<li>single-line textfield</li>
<li>multi-line textfield</li>
<li>checkbox</li>
<li>list selection</li>
<li>freeform list</li>
<li>URL</li>
<li>date</li>
</ul>
');
      $output .= '<p>'. t('For more information please read the
configuration and customization handbook <a href="@profile">Profile
page</a>.', array('@profile' =>
'http://drupal.org/handbook/modules/profile/')) .'</p>';
      return $output;
    case 'admin/user/profile':
      return '<p>'. t('Here you can define custom fields that users can
fill in in their user profile (such as <em>country</em>, <em>real
name</em>, <em>age</em>, ...).') .'</p>';
  }
}

/**
 * Implementation of hook_menu().
 */
function profile_menu($may_cache) {
  $items = array();

  if ($may_cache) {
```

```
    $items[] = array('path' => 'profile',
      'title' => t('User list'),
      'callback' => 'profile_browse',
      'access' => user_access('access user profiles'),
      'type' => MENU_SUGGESTED_ITEM);
    $items[] = array('path' => 'admin/user/profile',
      'title' => t('Profiles'),
      'description' => t('Create customizable fields for your users.'),
      'callback' => 'profile_admin_overview');
    $items[] = array('path' => 'admin/user/profile/add',
      'title' => t('Add field'),
      'callback' => 'drupal_get_form',
      'callback arguments' => array('profile_field_form'),
      'type' => MENU_CALLBACK);
    $items[] = array('path' => 'admin/user/profile/autocomplete',
      'title' => t('Profile category autocomplete'),
      'callback' => 'profile_admin_settings_autocomplete',
      'access' => user_access('administer users'),
      'type' => MENU_CALLBACK);
    $items[] = array('path' => 'admin/user/profile/edit',
      'title' => t('Edit field'),
      'callback' => 'drupal_get_form',
      'callback arguments' => array('profile_field_form'),
      'type' => MENU_CALLBACK);
    $items[] = array('path' => 'admin/user/profile/delete',
      'title' => t('Delete field'),
      'callback' => 'drupal_get_form',
      'callback arguments' => array('profile_field_delete'),
      'type' => MENU_CALLBACK);
    $items[] = array('path' => 'profile/autocomplete', 'title' =>
t('Profile autocomplete'),
      'callback' => 'profile_autocomplete',
      'access' => 1,
      'type' => MENU_CALLBACK);
  }

  return $items;
}

/**
 * Implementation of hook_block().
 */
function profile_block($op = 'list', $delta = 0, $edit = array()) {

  if ($op == 'list') {
     $blocks[0]['info'] = t('Author information');

     return $blocks;
  }
  else if ($op == 'configure' && $delta == 0) {
    // Compile a list of fields to show
    $fields = array();
    $result = db_query('SELECT name, title, weight, visibility FROM
{profile_fields} WHERE visibility IN (%d, %d) ORDER BY weight',
PROFILE_PUBLIC, PROFILE_PUBLIC_LISTINGS);
    while ($record = db_fetch_object($result)) {
      $fields[$record->name] = $record->title;
```

```
    }
    $fields['user_profile'] = t('Link to full user profile');
    $form['profile_block_author_fields'] = array('#type' =>
'checkboxes',
      '#title' => t('Profile fields to display'),
      '#default_value' => variable_get('profile_block_author_fields',
NULL),
      '#options' => $fields,
      '#description' => t('Select which profile fields you wish to
display in the block. Only fields designated as public in the <a
href="@profile-admin">profile field configuration</a> are available.',
array('@profile-admin' => url('admin/user/profile'))),
    );
    return $form;
  }
  else if ($op == 'save' && $delta == 0) {
    variable_set('profile_block_author_fields',
$edit['profile_block_author_fields']);
  }
  else if ($op == 'view') {
    if (user_access('access user profiles')) {
      if ((arg(0) == 'node') && is_numeric(arg(1)) && (arg(2) == NULL))
{
        $node = node_load(arg(1));
        $account = user_load(array('uid' => $node->uid));

        if ($use_fields = variable_get('profile_block_author_fields',
array())) {
          // Compile a list of fields to show.
          $fields = array();
          $result = db_query('SELECT name, title, type, visibility,
weight FROM {profile_fields} WHERE visibility IN (%d, %d) ORDER BY
weight', PROFILE_PUBLIC, PROFILE_PUBLIC_LISTINGS);
          while ($record = db_fetch_object($result)) {
            // Ensure that field is displayed only if it is among the
defined block fields and, if it is private, the user has appropriate
permissions.
            if (isset($use_fields[$record->name]) &&
$use_fields[$record->name]) {
              $fields[] = $record;
            }
          }
        }

        if ($fields) {
          $profile = _profile_update_user_fields($fields, $account);
          $output .= theme('profile_block', $account, $profile, TRUE);
        }

        if (isset($use_fields['user_profile']) &&
$use_fields['user_profile']) {
          $output .= '<div>'. l(t('View full user profile'), 'user/'.
$account->uid) .'</div>';
        }
      }

      if ($output) {
```

```
          $block['subject'] = t('About %name', array('%name' =>
$account->name));
          $block['content'] = $output;
          return $block;
      }
    }
  }
}

/**
 * Implementation of hook_user().
 */
function profile_user($type, &$edit, &$user, $category = NULL) {
  switch ($type) {
    case 'load':
      return profile_load_profile($user);
    case 'register':
      return profile_form_profile($edit, $user, $category, TRUE);
    case 'update':
    case 'insert':
      return profile_save_profile($edit, $user, $category);
    case 'view':
      return profile_view_profile($user);
    case 'form':
      return profile_form_profile($edit, $user, $category);
    case 'validate':
      return profile_validate_profile($edit, $category);
    case 'categories':
      return profile_categories();
    case 'delete':
      db_query('DELETE FROM {profile_values} WHERE uid = %d', $user-
>uid);
  }
}

/**
 * Menu callback: Generate a form to add/edit a user profile field.
 */
function profile_field_form($arg = NULL) {
  if (arg(3) == 'edit') {
    if (is_numeric($arg)) {
      $fid = $arg;

      $edit = db_fetch_array(db_query('SELECT * FROM {profile_fields}
WHERE fid = %d', $fid));

      if (!$edit) {
        drupal_not_found();
        return;
      }
      drupal_set_title(t('edit %title', array('%title' =>
$edit['title'])));
      $form['fid'] = array('#type' => 'value',
        '#value' => $fid,
      );
      $type = $edit['type'];
    }
```

```
    else {
      drupal_not_found();
      return;
    }
  }
  else {
    $types = _profile_field_types();
    if (!isset($types[$arg])) {
      drupal_not_found();
      return;
    }
    $type = $arg;
    drupal_set_title(t('add new %type', array('%type' =>
$types[$type])));
    $edit = array('name' => 'profile_');
    $form['type'] = array('#type' => 'value', '#value' => $type);
  }
  $form['fields'] = array('#type' => 'fieldset',
    '#title' => t('Field settings'),
  );
  $form['fields']['category'] = array('#type' => 'textfield',
    '#title' => t('Category'),
    '#default_value' => $edit['category'],
    '#autocomplete_path' => 'admin/user/profile/autocomplete',
    '#description' => t('The category the new field should be part of.
Categories are used to group fields logically. An example category is
"Personal information".'),
    '#required' => TRUE,
  );
  $form['fields']['title'] = array('#type' => 'textfield',
    '#title' => t('Title'),
    '#default_value' => $edit['title'],
    '#description' => t('The title of the new field. The title will be
shown to the user. An example title is "Favorite color".'),
    '#required' => TRUE,
  );
  $form['fields']['name'] = array('#type' => 'textfield',
    '#title' => t('Form name'),
    '#default_value' => $edit['name'],
    '#description' => t('The name of the field. The form name is not
shown to the user but used internally in the HTML code and URLs.
Unless you know what you are doing, it is highly recommended that you
prefix the form name with <code>profile_</code> to avoid name clashes
with other fields. Spaces or any other special characters except dash
(-) and underscore (_) are not allowed. An example name is
"profile_favorite_color" or perhaps just "profile_color".'),
    '#required' => TRUE,
  );
  $form['fields']['explanation'] = array('#type' => 'textarea',
    '#title' => t('Explanation'),
    '#default_value' => $edit['explanation'],
    '#description' => t('An optional explanation to go with the new
field. The explanation will be shown to the user.'),
  );
  if ($type == 'selection') {
    $form['fields']['options'] = array('#type' => 'textarea',
      '#title' => t('Selection options'),
```

```
      '#default_value' => $edit['options'],
      '#description' => t('A list of all options. Put each option on a
separate line. Example options are "red", "blue", "green", etc.'),
    );
  }
  $form['fields']['weight'] = array('#type' => 'weight',
    '#title' => t('Weight'),
    '#default_value' => $edit['weight'],
    '#delta' => 5,
    '#description' => t('The weights define the order in which the form
fields are shown. Lighter fields "float up" towards the top of the
category.'),
  );
  $form['fields']['visibility'] = array('#type' => 'radios',
    '#title' => t('Visibility'),
    '#default_value' => isset($edit['visibility']) ?
$edit['visibility'] : PROFILE_PUBLIC,
    '#options' => array(PROFILE_HIDDEN => t('Hidden profile field, only
accessible by administrators, modules and themes.'), PROFILE_PRIVATE =>
t('Private field, content only available to privileged users.'),
PROFILE_PUBLIC => t('Public field, content shown on profile page but
not used on member list pages.'), PROFILE_PUBLIC_LISTINGS => t('Public
field, content shown on profile page and on member list pages.')),
  );
  if ($type == 'selection' || $type == 'list' || $type == 'textfield')
{
    $form['fields']['page'] = array('#type' => 'textfield',
      '#title' => t('Page title'),
      '#default_value' => $edit['page'],
      '#description' => t('To enable browsing this field by value,
enter a title for the resulting page. The word <code>%value</code> will
be substituted with the corresponding value. An example page title is
"People whose favorite color is %value". This is only applicable for a
public field.'),
    );
  }
  else if ($type == 'checkbox') {
    $form['fields']['page'] = array('#type' => 'textfield',
      '#title' => t('Page title'),
      '#default_value' => $edit['page'],
      '#description' => t('To enable browsing this field by value,
enter a title for the resulting page. An example page title is "People
who are employed". This is only applicable for a public field.'),
    );
  }
  $form['fields']['autocomplete'] = array('#type' => 'checkbox',
    '#title' => t('Form will auto-complete while user is typing.'),
    '#default_value' => $edit['autocomplete'],
  );
  $form['fields']['required'] = array('#type' => 'checkbox',
    '#title' => t('The user must enter a value.'),
    '#default_value' => $edit['required'],
  );
  $form['fields']['register'] = array('#type' => 'checkbox',
    '#title' => t('Visible in user registration form.'),
    '#default_value' => $edit['register'],
  );
```

```
  $form['submit'] = array('#type' => 'submit',
    '#value' => t('Save field'),
  );
  return $form;
}

/**
 * Validate profile_field_form submissions.
 */
function profile_field_form_validate($form_id, $form_values) {
  // Validate the 'field name':
  if (preg_match('/[^a-zA-Z0-9_-]/', $form_values['name'])) {
    form_set_error('name', t('The specified form name contains one or
more illegal characters. Spaces or any other special characters except
dash (-) and underscore (_) are not allowed.'));
  }

  if (in_array($form_values['name'], user_fields())) {
    form_set_error('name', t('The specified form name is reserved for
use by Drupal.'));
  }
  // Validate the category:
  if (!$form_values['category']) {
    form_set_error('category', t('You must enter a category.'));
  }
  if ($form_values['category'] == 'account') {
    form_set_error('category', t('The specified category name is
reserved for use by Drupal.'));
  }
  $args1 = array($form_values['title'], $form_values['category']);
  $args2 = array($form_values['name']);
  $query_suffix = '';

  if (isset($form_values['fid'])) {
    $args1[] = $args2[] = $form_values['fid'];
    $query_suffix = ' AND fid != %d';
  }

  if (db_result(db_query("SELECT fid FROM {profile_fields} WHERE title
= '%s' AND category = '%s'". $query_suffix, $args1))) {
    form_set_error('title', t('The specified title is already in
use.'));
  }
  if (db_result(db_query("SELECT fid FROM {profile_fields} WHERE name =
'%s'". $query_suffix, $args2))) {
    form_set_error('name', t('The specified name is already in use.'));
  }
}

/**
 * Process profile_field_form submissions.
 */
function profile_field_form_submit($form_id, $form_values) {
  if (!isset($form_values['fid'])) {
    db_query("INSERT INTO {profile_fields} (title, name, explanation,
category, type, weight, required, register, visibility, autocomplete,
options, page) VALUES ('%s', '%s', '%s', '%s', '%s', %d, %d, %d, %d,
```

```
%d, '%s', '%s')", $form_values['title'], $form_values['name'],
$form_values['explanation'], $form_values['category'],
$form_values['type'], $form_values['weight'], $form_values['required'],
$form_values['register'], $form_values['visibility'],
$form_values['autocomplete'], $form_values['options'],
$form_values['page']);

    drupal_set_message(t('The field has been created.'));
    watchdog('profile', t('Profile field %field added under category
%category.', array('%field' => $form_values['title'], '%category' =>
$form_values['category'])), WATCHDOG_NOTICE, l(t('view'),
'admin/user/profile'));
  }
  else {
    db_query("UPDATE {profile_fields} SET title = '%s', name = '%s',
explanation = '%s', category = '%s', weight = %d, required = %d,
register = %d, visibility = %d, autocomplete = %d, options = '%s', page
= '%s' WHERE fid = %d", $form_values['title'], $form_values['name'],
$form_values['explanation'], $form_values['category'],
$form_values['weight'], $form_values['required'],
$form_values['register'], $form_values['visibility'],
$form_values['autocomplete'], $form_values['options'],
$form_values['page'], $form_values['fid']);

    drupal_set_message(t('The field has been updated.'));
  }
  cache_clear_all();

  return 'admin/user/profile';
}

/**
 * Menu callback; deletes a field from all user profiles.
 */
function profile_field_delete($fid) {
  $field = db_fetch_object(db_query("SELECT title FROM {profile_fields}
WHERE fid = %d", $fid));
  if (!$field) {
    drupal_not_found();
    return;
  }
  $form['fid'] = array('#type' => 'value', '#value' => $fid);
  $form['title'] = array('#type' => 'value', '#value' => $field-
>title);

  return confirm_form($form,
    t('Are you sure you want to delete the field %field?',
array('%field' => $field->title)), 'admin/user/profile',
    t('This action cannot be undone. If users have entered values into
this field in their profile, these entries will also be deleted. If you
want to keep the user-entered data, instead of deleting the field you
may wish to <a href="@edit-field">edit this field</a> and change it to
a hidden profile field so that it may only be accessed by
administrators.', array('@edit-field' =>
url('admin/user/profile/edit/'. $fid))),
    t('Delete'), t('Cancel'));
}
```

```
/**
 * Process a field delete form submission.
 */
function profile_field_delete_submit($form_id, $form_values) {
  db_query('DELETE FROM {profile_fields} WHERE fid = %d',
$form_values['fid']);
  db_query('DELETE FROM {profile_values} WHERE fid = %d',
$form_values['fid']);

  cache_clear_all();

  drupal_set_message(t('The field %field has been deleted.',
array('%field' => $form_values['title'])));
  watchdog('profile', t('Profile field %field deleted.', array('%field'
=> $form_values['title'])), WATCHDOG_NOTICE, l(t('view'),
'admin/user/profile'));

  return 'admin/user/profile';
}

/**
 * Menu callback; display a listing of all editable profile fields.
 */
function profile_admin_overview() {

  $result = db_query('SELECT * FROM {profile_fields} ORDER BY category,
weight');
  $rows = array();
  while ($field = db_fetch_object($result)) {
    $rows[] = array(check_plain($field->title), $field->name,
_profile_field_types($field->type), $field->category, l(t('edit'),
"admin/user/profile/edit/$field->fid"), l(t('delete'),
"admin/user/profile/delete/$field->fid"));
  }
  if (count($rows) == 0) {
    $rows[] = array(array('data' => t('No fields defined.'), 'colspan'
=> '6'));
  }

  $header = array(t('Title'), t('Name'), t('Type'), t('Category'),
array('data' => t('Operations'), 'colspan' => '2'));

  $output  = theme('table', $header, $rows);
  $output .= '<h2>'. t('Add new field') .'</h2>';
  $output .= '<ul>';
  foreach (_profile_field_types() as $key => $value) {
    $output .= '<li>'. l($value, "admin/user/profile/add/$key")
.'</li>';
  }
  $output .= '</ul>';

  return $output;
}

/**
 * Menu callback; display a list of user information.
```

```
 */
function profile_browse() {
  $name = arg(1);
  list(, , $value) = explode('/', $_GET['q'], 3);

  $field = db_fetch_object(db_query("SELECT DISTINCT(fid), type, title,
page, visibility FROM {profile_fields} WHERE name = '%s'", $name));

  if ($name && $field->fid) {
    // Only allow browsing of fields that have a page title set.
    if (empty($field->page)) {
      drupal_not_found();
      return;
    }
    // Do not allow browsing of private fields by non-admins.
    if (!user_access('administer users') && $field->visibility ==
PROFILE_PRIVATE) {
       drupal_access_denied();
       return;
    }

    // Compile a list of fields to show.
    $fields = array();
    $result = db_query('SELECT name, title, type, weight, page FROM
{profile_fields} WHERE fid != %d AND visibility = %d ORDER BY weight',
$field->fid, PROFILE_PUBLIC_LISTINGS);
    while ($record = db_fetch_object($result)) {
      $fields[] = $record;
    }

    // Determine what query to use:
    $arguments = array($field->fid);
    switch ($field->type) {
      case 'checkbox':
        $query = 'v.value = 1';
        break;
      case 'textfield':
      case 'selection':
        $query = "v.value = '%s'";
        $arguments[] = $value;
        break;
      case 'list':
        $query = "v.value LIKE '%%%s%%'";
        $arguments[] = $value;
        break;
      default:
        drupal_not_found();
        return;
    }

    // Extract the affected users:
    $result = pager_query("SELECT u.uid, u.access FROM {users} u INNER
JOIN {profile_values} v ON u.uid = v.uid WHERE v.fid = %d AND $query
AND u.access != 0 AND u.status != 0 ORDER BY u.name ASC", 20, 0, NULL,
$arguments);

    $output = '<div id="profile">';
```

```
    while ($account = db_fetch_object($result)) {
      $account = user_load(array('uid' => $account->uid));
      $profile = _profile_update_user_fields($fields, $account);
      $output .= theme('profile_listing', $account, $profile);
    }
    $output .= theme('pager', NULL, 20);

    if ($field->type == 'selection' || $field->type == 'list' ||
$field->type == 'textfield') {
      $title = strtr(check_plain($field->page), array('%value' =>
theme('placeholder', $value)));
    }
    else {
      $title = check_plain($field->page);
    }
    $output .= '</div>';

    drupal_set_title($title);
    return $output;
  }
  else if ($name && !$field->fid) {
    drupal_not_found();
  }
  else {
    // Compile a list of fields to show.
    $fields = array();
    $result = db_query('SELECT name, title, type, weight, page FROM
{profile_fields} WHERE visibility = %d ORDER BY category, weight',
PROFILE_PUBLIC_LISTINGS);
    while ($record = db_fetch_object($result)) {
      $fields[] = $record;
    }

    // Extract the affected users:
    //$result = pager_query('SELECT uid, access FROM {users} WHERE uid
> 0 AND status != 0 AND access != 0 ORDER BY access DESC', 20, 0,
NULL);
      $result = pager_query('SELECT uid, access FROM {users} WHERE uid
> 0 AND status != 0 AND access != 0 ORDER BY name ASC', 20, 0, NULL);

    $output = '<div id="profile">';
    while ($account = db_fetch_object($result)) {
      if( $account->uid != 1 ) {
        $account = user_load(array('uid' => $account->uid));
        $profile = _profile_update_user_fields($fields, $account);
        $output .= theme('profile_listing', $account, $profile);
      }
    }
    $output .= '</div>';
    $output .= theme('pager', NULL, 20);

    drupal_set_title(t('user list'));
    return $output;
  }
}

function profile_load_profile(&$user) {
```

```
  $result = db_query('SELECT f.name, f.type, v.value FROM
{profile_fields} f INNER JOIN {profile_values} v ON f.fid = v.fid WHERE
uid = %d', $user->uid);
  while ($field = db_fetch_object($result)) {
    if (empty($user->{$field->name})) {
      $user->{$field->name} = _profile_field_serialize($field->type) ?
unserialize($field->value) : $field->value;
    }
  }
}

function profile_save_profile(&$edit, &$user, $category) {
  $result = _profile_get_fields($category);
  while ($field = db_fetch_object($result)) {
    if (_profile_field_serialize($field->type)) {
       $edit[$field->name] = serialize($edit[$field->name]);
    }
/**
brokenInsert
    db_query("DELETE FROM {profile_values} WHERE fid = %d AND uid =
%d", $field->fid, $user->uid);
    db_query("INSERT INTO {profile_values} (fid, uid, value) VALUES
(%d, %d, '%s')", $field->fid, $user->uid, $edit[$field->name]);

    // Mark field as handled (prevents saving to user->data).
    $edit[$field->name] = NULL;
*/
  }
}

function profile_view_field($user, $field) {
  // Only allow browsing of private fields for admins, if browsing is
enabled,
  // and if a user has permission to view profiles. Note that this
check is
  // necessary because a user may always see their own profile.
  $browse = user_access('access user profiles')
         && (user_access('administer users') || $field->visibility !=
PROFILE_PRIVATE)
         && !empty($field->page);

  if ($value = $user->{$field->name}) {
    switch ($field->type) {
      case 'textarea':
        return check_markup($value);
      case 'textfield':
      case 'selection':
          return $browse ? l($value, 'search/profile/'. $value) :
check_plain($value);
//        return $browse ? l($value, 'profile/'. $field->name .'/'.
$value) : check_plain($value);
      case 'checkbox':
        return $browse ? l($field->title, 'profile/'. $field->name) :
check_plain($field->title);
      case 'url':
        return '<a href="'. check_url($value) .'">'.
check_plain($value) .'</a>';
```

```
      case 'date':
        $format = substr(variable_get('date_format_short', 'm/d/Y -
H:i'), 0, 5);
        // Note: Avoid PHP's date() because it does not handle dates
before
        // 1970 on Windows. This would make the date field useless for
e.g.
        // birthdays.
        $replace = array('d' => sprintf('%02d', $value['day']),
                         'j' => $value['day'],
                         'm' => sprintf('%02d', $value['month']),
                         'M' => map_month($value['month']),
                         'Y' => $value['year'],
                         'H:i' => NULL,
                         'g:ia' => NULL);
        return strtr($format, $replace);
      case 'list':
        $values = split("[,\n\r]", $value);
        $fields = array();
        foreach ($values as $value) {
          if ($value = trim($value)) {
            $fields[] = $browse ? l($value, 'profile/'. $field->name
.'/'. $value) : check_plain($value);
          }
        }
        return implode(', ', $fields);
    }
  }
}

function profile_view_profile($user) {

  profile_load_profile($user);

  // Show private fields to administrators and people viewing their own
account.
  if (user_access('administer users') || $GLOBALS['user']->uid ==
$user->uid) {
    $result = db_query('SELECT * FROM {profile_fields} WHERE visibility
!= %d ORDER BY category, weight', PROFILE_HIDDEN);
  }
  else {
    $result = db_query('SELECT * FROM {profile_fields} WHERE visibility
!= %d AND visibility != %d ORDER BY category, weight', PROFILE_PRIVATE,
PROFILE_HIDDEN);
  }

  while ($field = db_fetch_object($result)) {
    if ($value = profile_view_field($user, $field)) {
      $title = ($field->type != 'checkbox') ? check_plain($field-
>title) : NULL;
      $item = array('title' => $title,
        'value' => $value,
        'class' => $field->name,
      );
      $fields[$field->category][$field->name] = $item;
    }
```

```
  }
  return $fields;
}

function _profile_form_explanation($field) {
  $output = $field->explanation;

  if ($field->type == 'list') {
    $output .= ' '. t('Put each item on a separate line or separate
them by commas. No HTML allowed.');
  }

  if ($field->visibility == PROFILE_PRIVATE) {
    $output .= ' '. t('The content of this field is kept private and
will not be shown publicly.');
  }

  return $output;
}

function profile_form_profile($edit, $user, $category, $register =
FALSE) {
  $result = _profile_get_fields($category, $register);
  $w = 0;
  while ($field = db_fetch_object($result)) {
    $category = $field->category;
    if (!isset($fields[$category])) {
      $fields[$category] = array('#type' => 'fieldset', '#title' =>
$category, '#weight' => $w++);
    }
    switch ($field->type) {
      case 'textfield':
      case 'url':
        $fields[$category][$field->name] = array('#type' =>
'textfield',
          '#title' => check_plain($field->title),
          '#default_value' => $edit[$field->name],
          '#maxlength' => 255,
          '#description' => _profile_form_explanation($field),
          '#required' => $field->required,
        );
        if ($field->autocomplete) {
          $fields[$category][$field->name]['#autocomplete_path'] =
"profile/autocomplete/". $field->fid;
        }
        break;
      case 'textarea':
        $fields[$category][$field->name] = array('#type' => 'textarea',
          '#title' => check_plain($field->title),
          '#default_value' => $edit[$field->name],
          '#description' => _profile_form_explanation($field),
          '#required' => $field->required,
        );
        break;
      case 'list':
        $fields[$category][$field->name] = array('#type' => 'textarea',
          '#title' => check_plain($field->title),
```

```
          '#default_value' => $edit[$field->name],
          '#description' => _profile_form_explanation($field),
          '#required' => $field->required,
        );
        break;
      case 'checkbox':
        $fields[$category][$field->name] = array('#type' => 'checkbox',
          '#title' => check_plain($field->title),
          '#default_value' => $edit[$field->name],
          '#description' => _profile_form_explanation($field),
          '#required' => $field->required,
        );
        break;
      case 'selection':
        $options = $field->required ? array() : array('--');
        $lines = split("[,\n\r]", $field->options);
        foreach ($lines as $line) {
          if ($line = trim($line)) {
            $options[$line] = $line;
          }
        }
        $fields[$category][$field->name] = array('#type' => 'select',
          '#title' => check_plain($field->title),
          '#default_value' => $edit[$field->name],
          '#options' => $options,
          '#description' => _profile_form_explanation($field),
          '#required' => $field->required,
        );
        break;
      case 'date':
        $fields[$category][$field->name] = array('#type' => 'date',
          '#title' => check_plain($field->title),
          '#default_value' => $edit[$field->name],
          '#description' => _profile_form_explanation($field),
          '#required' => $field->required,
        );
        break;
    }
  }
  return $fields;
}

/**
 * Callback to allow autocomplete of profile text fields.
 */
function profile_autocomplete($field, $string) {
  if (db_result(db_query("SELECT COUNT(*) FROM {profile_fields} WHERE
fid = %d AND autocomplete = 1", $field))) {
    $matches = array();
    $result = db_query_range("SELECT value FROM {profile_values} WHERE
fid = %d AND LOWER(value) LIKE LOWER('%s%%') GROUP BY value ORDER BY
value ASC", $field, $string, 0, 10);
    while ($data = db_fetch_object($result)) {
      $matches[$data->value] = check_plain($data->value);
    }

    print drupal_to_js($matches);
```

```
  }
  exit();
}

/**
 * Helper function: update an array of user fields by calling
profile_view_field
 */
function _profile_update_user_fields($fields, $account) {
  foreach ($fields as $key => $field) {
    $fields[$key]->value = profile_view_field($account, $field);
  }
  return $fields;
}

function profile_validate_profile($edit, $category) {
  $result = _profile_get_fields($category);
  while ($field = db_fetch_object($result)) {
    if ($edit[$field->name]) {
      if ($field->type == 'url') {
        if (!valid_url($edit[$field->name], TRUE)) {
          form_set_error($field->name, t('The value provided for %field
is not a valid URL.', array('%field' => $field->title)));
        }
      }
    }
    else if ($field->required && !user_access('administer users')) {
      form_set_error($field->name, t('The field %field is required.',
array('%field' => $field->title)));
    }
  }

  return $edit;
}

function profile_categories() {
  $result = db_query("SELECT DISTINCT(category) FROM
{profile_fields}");
  while ($category = db_fetch_object($result)) {
    $data[] = array('name' => $category->category, 'title' =>
$category->category, 'weight' => 3);
  }
  return $data;
}

function theme_profile_block($account, $fields = array()) {

  $output .= theme('user_picture', $account);

  foreach ($fields as $field) {
    if ($field->value) {
      if ($field->type == 'checkbox') {
        $output .= "<p>$field->value</p>\n";
      }
      else {
        $output .= "<p><strong>$field->title</strong><br />$field-
>value</p>\n";
```

```
      }
    }
  }

  return $output;
}


function theme_profile_listing($account, $fields = array()) {

  $output  = "<div class=\"profile\">\n";
  $output .= ' <div class="name">'. theme('username', $account)
."</div>\n";

  foreach ($fields as $field) {
    if ($field->value) {
      $output .= " <div class=\"field\">$field->value</div>\n";
    }
  }

  $output .= theme('user_picture', $account);

  $output .= '</div><p style="clear:both"> </p><hr />';
  $output .= "\n";

  return $output;
}

/*
function theme_profile_listing($account, $fields = array()) {

  $output  = "<div class=\"profile\">\n";
  $output .= theme('user_picture', $account);
  $output .= ' <div class="name">'. theme('username', $account)
."</div>\n";

  foreach ($fields as $field) {
    if ($field->value) {
      $output .= " <div class=\"field\">$field->value</div>\n";
    }
  }

  $output .= "</div>\n";

  return $output;
}
*/

function _profile_field_types($type = NULL) {
  $types = array('textfield' => t('single-line textfield'),
                 'textarea' => t('multi-line textfield'),
                 'checkbox' => t('checkbox'),
                 'selection' => t('list selection'),
                 'list' => t('freeform list'),
                 'url' => t('URL'),
                 'date' => t('date'));
  return isset($type) ? $types[$type] : $types;
```

```
}

function _profile_field_serialize($type = NULL) {
  return $type == 'date';
}

function _profile_get_fields($category, $register = FALSE) {
  $args = array();
  $sql = 'SELECT * FROM {profile_fields} WHERE ';
  $filters = array();
  if ($register) {
    $filters[] = 'register = 1';
  }
  else {
    // Use LOWER('%s') instead of PHP's strtolower() to avoid UTF-8
conversion issues.
    $filters[] = "LOWER(category) = LOWER('%s')";
    $args[] = $category;
  }
  if (!user_access('administer users')) {
    $filters[] = 'visibility != %d';
    $args[] = PROFILE_HIDDEN;
  }
  $sql .= implode(' AND ', $filters);
  $sql .= ' ORDER BY category, weight';
  return db_query($sql, $args);
}

/**
 * Retrieve a pipe delimited string of autocomplete suggestions for
profile categories
 */
function profile_admin_settings_autocomplete($string) {
  $matches = array();
  $result = db_query_range("SELECT category FROM {profile_fields} WHERE
LOWER(category) LIKE LOWER('%s%%')", $string, 0, 10);
  while ($data = db_fetch_object($result)) {
    $matches[$data->category] = check_plain($data->category);
  }
  print drupal_to_js($matches);
  exit();
}

/**
 * Implementation of hook_search().
 * Programmed by: Chuck Feltes
 */
function profile_search($op = 'search', $keys = NULL) {
 switch ($op) {
      case 'name':
            return t('Profiles');
      case 'search':
            $find = array();
            // Replace wildcards with MySQL / PostgreSQL wildcards
(from user_search)
            $keys = preg_replace('!\*+!', '%', $keys);
            // Build query - check against full_name and grad_year
```

```
            // grad_year stored as a string
            $query = "SELECT * FROM (users) WHERE LOWER(data) LIKE
LOWER('%%%s%%')";
            $result = pager_query($query, 15, 0, NULL, $keys);
            while($info = db_fetch_object($result) )
            {
                  $unserialized_data = unserialize($info->data);
                  $snippet_info = '';
                  if(is_array($unserialized_data)) {
                        if(
$unserialized_data['profile_full_name']!=NULL) {
                              $name =
$unserialized_data['profile_full_name'];
                        }
                        if( $unserialized_data['profile_major']!=NULL)
{
                              $snippet_info .= 'Major: '.
$unserialized_data['profile_major']. '<br />';
                        }
                        if(
$unserialized_data['profile_grad_year']!=NULL) {
                              $snippet_info .= 'Graduation Year: '.
$unserialized_data['profile_grad_year']. '<br />';
                        }
                  }
                  // if the data didn't come out as an array, something
went wrong
                  // so do a default link
                  else {
                        $name = $info->name;
                  }
                  if($info->picture != NULL ) {
                        $snippet_info .= '<img src="'. $info->picture.
'" />';
                  }
                  $find[] = array('title' => $name, 'link' =>
url('user/'. $info->uid, NULL, NULL, TRUE), 'snippet'=>$snippet_info);
/**
                  if($item->picture != NULL ) {
                        $picture = array('<img src="'. $info->picture.
'" />' );
                  }
                  $find[] = array( 'title' => $info->name,
                                    'link' => url('user/'. $info->uid,
NULL, NULL, TRUE),
                                    'snippet' => '<img src="'. $info-
>picture. '" />' );*/
//                                  'snippet' => 'Graduation Year: '.
$account->profile_grad_year. '<br />Major: '. $account->profile_major);
            }
            return $find;
 }
}
```