

```
<?php
// $Id: content_profile.module,v 1.1.2.10 2008/06/20 11:07:06 fago Exp $

/**
 * @file
 * Marks content types as profiles
 */

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

  //Register a path for each content profile type
  foreach (content_profile_get_types('names') as $type => $typename) {
    $items['admin/content/node-type/'. str_replace('_', '-', $type) .'/edit']
= array(
    'title' => 'Edit',
    'type' => MENU_DEFAULT_LOCAL_TASK,
  );
    $items['admin/content/node-type/'. str_replace('_', '-', $type)
.'/profile'] = array(
    'title' => 'Content profile',
    'description' => 'Configure the display and management of this content
profile.',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('content_profile_admin_settings', $type),
    'access callback' => 'user_access',
    'access arguments' => array('administer nodes'),
    'type' => MENU_LOCAL_TASK,
    'weight' => 1,
  );
  }
  return $items;
}

/**
 * Menu callback; content profile settings.
 */
function content_profile_admin_settings(&$form_state, $type) {
  $form_state['type'] = $type;

  $form['weight'] = array(
```

```
'#type' => 'weight',
'#title' => t("Weight"),
'#default_value' => content_profile_get_settings($type, 'weight'),
'#description' => t('The weight of content of this content type where
ever they appear - this applies to the input form integration as well to the
display integration. '),
'#weight' => 5,
);
$form['display'] = array(
  '#type' => 'fieldset',
  '#title' => t('Display settings'),
  '#description' => t('Customize the display of this content profile. '),
  '#collapsible' => TRUE,
);
$form['display']['user_display'] = array(
  '#type' => 'radios',
  '#title' => t("User page display style"),
  '#default_value' => content_profile_get_settings($type, 'user_display'),
  '#options' => array(
    0 => t('Don\'t display this content profile on the user account page'),
    'link' => t('Display it as link to the profile content'),
    'full' => t('Display the full content'),
    'teaser' => t('Display the content\'s teaser'),
  ),
);
$form['display']['edit_link'] = array(
  '#type' => 'checkbox',
  '#title' => t("Include an edit link to the display"),
  '#default_value' => content_profile_get_settings($type, 'edit_link'),
);
$form['display']['add_link'] = array(
  '#type' => 'checkbox',
  '#title' => t("Show a link to the content profile creation page, if there
is no profile."),
  '#default_value' => content_profile_get_settings($type, 'add_link'),
  '#description' => t("If selected and the user has no profile of this type
yet, a link to add one is shown on the user page."),
);
$form['submit'] = array(
  '#type' => 'submit',
  '#value' => t('Submit'),
  '#weight' => 10,
);
return $form;
}
```

```
function content_profile_admin_settings_submit($form, &$form_state) {
  $settings = content_profile_get_settings($form_state['type']);
  foreach (content_profile_settings_info() as $setting => $default) {
    $settings[$setting] = $form_state['values'][$setting];
  }
  content_profile_set_settings($form_state['type'], $settings);
  drupal_set_message('Your changes have been saved.');
```

```
}

/**
 * Determine if a given node is a content_profile.
 * @param $type The node object or the node's type
 */
function is_content_profile($type) {
  if (is_object($type)) {
    $type = $type->type;
  }
  return is_array(variable_get('content_profile_' . $type, FALSE));
}

/**
 * Builds a list of available content types that are marked as
content_profiles,
 * and returns an array of content profile content types in the specified
format.
 *
 * @param $op When set to 'types', content profile content types are returned
 *           as type objects. When set to 'names', only their type names
 *           are returned.
 * @param $setting If set, only content types that have this setting
activated are
 *           returned. Leave it NULL to get all content profile types.
 */
function content_profile_get_types($op = 'types', $setting = NULL) {
  $types = array();

  foreach (node_get_types($op) as $type => $info) {
    if (is_content_profile($type) && (!isset($setting) ||
content_profile_get_setting($type, $setting))) {
      $types[$type] = $info;
    }
  }
  return $types;
}
```

```
/**
 * Implementation of hook_node_type():
 * Rename or delete the settings variable if a type changes.
 */
function content_profile_node_type($op, $info) {
  switch ($op) {
    case 'delete':
      variable_del('content_profile_'. $info->type);
      break;
    case 'update':
      if (!empty($info->old_type) && $info->old_type != $info->type) {
        if (is_content_profile($info->old_type)) {
          $settings = variable_get('content_profile_'. $info->old_type,
array());
          variable_del('content_profile_'. $info->old_type);
          variable_set('content_profile_'. $info->type, $settings);
        }
      }
      break;
  }
}

/**
 * Implementation of hook_form_alter().
 */
function content_profile_form_alter(&$form, $form_state, $form_id) {
  if ($form_id == 'node_type_form') {
    $form['content_profile'] = array(
      '#type' => 'fieldset',
      '#title' => t('Content Profile'),
      '#collapsible' => TRUE,
      '#weight' => 32,
    );
    $form['content_profile']['content_profile'] = array(
      '#type' => 'checkbox',
      '#title' => t('Use this content type as a content profile for users'),
      '#default_value' => is_content_profile($form['#node_type']->type),
      '#return_value' => array(),
    );
  }
}

/**
 * Implementation of hook_user():
```

```

*/
function content_profile_user($op, &$edit, &$account, $category = NULL) {
  global $user;

  switch ($op) {
    case 'view':
      $account->content['content_profile'] =
content_profile_show_profiles($account->uid);
      break;

    case 'delete':
      // Retrieve all profile nodes (in any language) for this user by
issuing an SQL query.
      if ($types = content_profile_get_types()) {
        $condition = array_fill(0, count($types), "type = '%s'");
        $arguments = array_merge(array_keys($types), array($account->uid));

        $result = db_query("SELECT * FROM {node} WHERE (". implode(', ',
$condition) .") AND uid = %d", $arguments);
        while ($node = db_fetch_object($result)) {
          _content_profile_node_delete($node);
        }
      }
      break;
  }
}

/**
 * The original node_delete() function uses node_load() to get the $node
object.
 * Unfortunately, when a hook_user('delete') is called, node_load() doesn't
 * work anymore because the user has already been deleted, and node_load()
 * still expects the user to exist in the {user} table.
 *
 * So this is a modified copy of node_delete() that deletes a node without
 * calling node_load(), taking the full $node object (as retrieved by a
simple
 * "SELECT * FROM {node}" query) instead of just the $nid.
 */
function _content_profile_node_delete($node) {
  // Copied over from node_load(), so that node_invoke('delete') gets
  // the fully extended node object, like modules would expect:

  if ($node->nid) {
    // Call the node specific callback (if any) and piggy-back the

```

```
// results to the node or overwrite some values.
if ($extra = node_invoke($node, 'load')) {
  foreach ($extra as $key => $value) {
    $node->$key = $value;
  }
}
if ($extra = node_invoke_nodeapi($node, 'load')) {
  foreach ($extra as $key => $value) {
    $node->$key = $value;
  }
}
}

// Copied over from node_delete():

db_query('DELETE FROM {node} WHERE nid = %d', $node->nid);
db_query('DELETE FROM {node_revisions} WHERE nid = %d', $node->nid);

// Call the node-specific callback (if any):
node_invoke($node, 'delete');
node_invoke_nodeapi($node, 'delete');

// Clear the cache so an anonymous poster can see the node being deleted.
cache_clear_all();

// Remove this node from the search index if needed.
if (function_exists('search_wipe')) {
  search_wipe($node->nid, 'node');
}
watchdog('content', '@type: deleted %title.', array('@type' => $node->type,
'%title' => $node->title));
drupal_set_message(t('@type %title has been deleted.', array('@type' =>
node_get_types('name', $node), '%title' => $node->title)));
}

/**
 * Implementation of hook_nodeapi().
 */
function content_profile_nodeapi(&$node, $op, $a3 = NULL, $a4 = NULL) {
  global $last_nid;

  if ($op == 'prepare' && is_content_profile($node) && !isset($node->nid) &&
!user_access('administer nodes')) {
    // Check if this nodetype already exists
    if ($nid = content_profile_profile_exists($node, $node->uid)) {
```

```

    // This node already exists, redirect to edit page
    drupal_goto('node/'. $nid .'/edit', 'destination=user/'. $node->uid);
  }
}
else if ($op == 'validate' && is_content_profile($node) &&
user_access('administer nodes') && !isset($node->nid)) {
  //check whether the selected user has already a profile
  $uid = db_result(db_query("SELECT uid FROM {users} WHERE name = '%s'",
$node->name));
  if (content_profile_profile_exists($node, $uid)) {
    form_set_error('name', t('This user has already a content profile of
this type. You can only create one profile per user.'));
  }
}
else if ($op == 'alter' && is_content_profile($node->type) &&
isset($last_nid)) {
  // Index themed node
  $node->body = theme('node', $node, FALSE, TRUE); //TODO: check if still
necessary and if it needs to be ported to 6.x
}
}

```

```

/**
 * Checks whether a node of this type exists already for the author
 *
 * @param $node The node, which is to be created
 * @param $uid The user to check for
 * @return If a node exists, the node id, FALSE else.
 */
function content_profile_profile_exists($node, $uid) {
  $query = "SELECT n.nid AS nid FROM {node} n WHERE n.type = '%s' AND n.uid =
%d ";
  if (module_exists('translation') &&
translation_supported_type($node->type)) {
    $query .= "AND n.language = '%s'";
  }
  return db_result(db_query($query, $node->type, $uid, $node->language));
}

```

```

/**
 * Returns the content_profiles' settings
 * @param $type The content type to return settings for
 * @param $return 'all' or one of the content_profile_available_settings(),
e.g. user_edit, register or weight
 */

```

```
function content_profile_get_settings($type, $return = 'all') {
    $settings = variable_get('content_profile.'. $type, array());
    $settings += content_profile_settings_info();
    if ($return == 'all') {
        return $settings;
    }
    return $settings[$return];
}

/**
 * Saves the content_profile settings of a content type
 */
function content_profile_set_settings($type, $settings) {
    variable_set('content_profile.'. $type, $settings);
}

/**
 * Returns an array, which defines the default value for each
 * available setting
 */
function content_profile_settings_info() {
    return module_invoke_all('content_profile_settings');
}

/**
 * Implementation of hook_content_profile_settings()
 * Defines content profile settings and their default value
 */
function content_profile_content_profile_settings() {
    return array(
        'weight' => 0,
        'user_display' => 'teaser',
        'edit_link' => 1,
        'add_link' => 1,
    );
}

/**
 * Loads the node, like node_load but caches the results
 */
function content_profile_load($type, $uid) {
    static $cache = array();

    if (!isset($cache[$type][$uid])) {
        $cache[$type][$uid] = node_load(array('type' => $type, 'uid' => $uid));
    }
}
```

```
}
return $cache[$type][$uid];
}

/**
 * Returns an array suitable for use with drupal_render,
 * that shows all content_profiles as configured by the admin
 */
function content_profile_show_profiles($uid) {
  global $user;

  $content = array();
  foreach (content_profile_get_types('names') as $type => $type_name) {
    $node = content_profile_load($type, $uid);

    if (($style = content_profile_get_settings($type, 'user_display')) &&
        $node && node_access('view', $node)) {
      $content['content_profile_' . $type] = array(
        '#theme' => ($style == 'link') ? 'content_profile_display_link' :
'content_profile_display_view',
        '#edit_link' => content_profile_get_settings($type, 'edit_link'),
        '#uid' => $uid,
        '#style' => $style,
        '#content_type' => $type,
        '#weight' => content_profile_get_settings($type, 'weight'),
        '#suffix' => '<br />',
      );
    }
    else if ($uid == $user->uid && user_access('create ' . $type . ' content')
        && content_profile_get_settings($type, 'add_link') && !$node) {
      $content['content_profile_' . $type] = array(
        '#theme' => 'content_profile_display_add_link',
        '#uid' => $uid,
        '#content_type' => $type,
        '#weight' => content_profile_get_settings($type, 'weight'),
        '#suffix' => '<br />',
      );
    }
    $content['#prefix'] = '<p id="content-profile-view">';
    $content['#suffix'] = '</p>';
  }
  return $content;
}

/**
```

```
* Implementation of hook_theme()
*/
function content_profile_theme() {
  return array(
    'content_profile_display_view' => array(
      'template' => 'content_profile-display-view',
      'arguments' => array('element' => NULL),
      'file' => 'content_profile.theme.inc',
    ),
    'content_profile_display_add_link' => array(
      'file' => 'content_profile.theme.inc',
    ),
    'content_profile_display_link' => array(
      'file' => 'content_profile.theme.inc',
    ),
    'content_profile_display_tab_view' => array(
      'file' => 'content_profile.theme.inc',
    ),
    'content_profile_display_tab_edit' => array(
      'file' => 'content_profile.theme.inc',
    ),
  );
}

/**
 * Implementation of hook_simpletest()
 */
function content_profile_simpletest() {
  // Scan through content_profile/tests directory for any .test files to tell
  SimpleTest module.
  $tests = file_scan_directory(drupal_get_path('module', 'content_profile')
  . '/tests', '\.test');
  return array_keys($tests);
}
```