

Simpletest Coverage - sites/all/modules/nodequeue/nodequeue.module

```
1  <?php
2  // $Id: nodequeue.module,v 1.82 2009/06/01 16:07:19 ezrag Exp $
3
4  // -----
5  // Drupal Hooks
6
7  /**
8   * Implementation of hook_perm
9   */
10 function nodequeue_perm() {
11   return array('manipulate queues', 'administer nodequeue', 'manipulate all queues');
12 }
13
14 /**
15  * Implementation of hook_init().
16  *
17  * Loads subsidiary includes for other modules.
18  */
19 function nodequeue_init() {
20   include_once drupal_get_path('module', 'nodequeue') . '/includes/nodequeue.actions.inc';
21 }
22
23 /**
24  * Helper function - since hook_menu now takes a function instead of a boolean, this function is used to compute the user's access.
25  *
26  * @return boolean
27  */
28 function _nodequeue_access_admin_or_manipulate() {
29   return user_access('administer nodequeue') || user_access('manipulate queues');
30 }
31
32 /**
33  * Implementation of hook_menu
34  */
35 function nodequeue_menu() {
36   $items = array();
37
38   $admin_access = array('administer nodequeue');
39   $access = array('manipulate queues');
40
41   // administrative items
42   $items['admin/content/nodequeue'] = array(
43     'title' => 'Nodequeue',
44     'access callback' => '_nodequeue_access_admin_or_manipulate',
45     'page callback' => 'nodequeue_view_queues',
46     'description' => 'Create and maintain simple nodequeues.',
47     'type' => MENU_NORMAL_ITEM
48   );
49   $items['admin/content/nodequeue/list'] = array(
50     'title' => 'List',
51     'access callback' => '_nodequeue_access_admin_or_manipulate',
52     'page callback' => 'nodequeue_view_queues',
53     'weight' => -1,
54     'type' => MENU_DEFAULT_LOCAL_TASK
55   );
56   $items['admin/content/nodequeue/settings'] = array(
57     'title' => 'Settings',
58     'access arguments' => $admin_access,
59     'page callback' => 'drupal_get_form',
60     'page arguments' => array('nodequeue_admin_settings'),
61     'type' => MENU_LOCAL_TASK
62   );
63   $items['nodequeue/autocomplete'] = array(
64     'title' => 'Autocomplete',
65     'access arguments' => $access,
66     'page callback' => 'nodequeue_autocomplete',
67     'type' => MENU_CALLBACK
68   );
69   $items['nodequeue/ajax/add'] = array(
70     'title' => 'ajax add',
71     'access arguments' => $access,
72     'page callback' => 'nodequeue_ajax_add',
73     'type' => MENU_CALLBACK
74   );
75   $info = nodequeue_api_info();
76   foreach ($info as $key => $data) {
77     $items['admin/content/nodequeue/add/'. $key] = array(
78       'title' => 'Add @type',
79       'title arguments' => array('@type' => strtolower($data['title'])),
80       'access arguments' => $admin_access,
81       'page callback' => 'drupal_get_form',
82       'page arguments' => array('nodequeue_edit_queue_form', $key),
83       'type' => MENU_LOCAL_TASK
84     );
85   }
86 }
```

```

87 // Note: This path used to set up with a call to nodequeue_load_queues_by_type, passing the result in. Now, that function is called inside of nodequeue_node
88
89 $items['node/%node/nodequeue'] = array(
90   'title' => '@tab',
91   'title arguments' => array('@tab' => variable_get('nodequeue_tab_name', 'Nodequeue')),
92   'access callback' => 'nodequeue_node_tab_access',
93   'access arguments' => array(1),
94   'page callback' => 'nodequeue_node_tab',
95   'page arguments' => array(1),
96   'type' => MENU_LOCAL_TASK,
97   'weight' => 5
98 );
99
100 // Administrative items for an individual queue.
101 $items['admin/content/nodequeue/%nodequeue'] = array(
102   'access arguments' => array(3),
103   'access callback' => 'nodequeue_queue_access',
104   'page callback' => 'nodequeue_admin_view',
105   'page arguments' => array(3),
106   'type' => MENU_CALLBACK
107 );
108 $items['admin/content/nodequeue/%nodequeue/view/%subqueue'] = array(
109   'title' => 'View',
110   'access arguments' => array(3, 5),
111   'access callback' => 'nodequeue_queue_access',
112   'page callback' => 'nodequeue_admin_view',
113   'page arguments' => array(3, 5),
114   'weight' => -10,
115   'type' => MENU_DEFAULT_LOCAL_TASK
116 );
117 // Actual administrative items.
118 $items['admin/content/nodequeue/%nodequeue/edit'] = array(
119   'title' => 'Edit',
120   'access arguments' => $admin_access,
121   'page callback' => 'drupal_get_form',
122   'page arguments' => array('nodequeue_edit_queue_form', 3),
123   'type' => MENU_LOCAL_TASK
124 );
125 $items['admin/content/nodequeue/%nodequeue/delete'] = array(
126   'title' => 'Delete',
127   'access arguments' => $admin_access,
128   'page callback' => 'drupal_get_form',
129   'page arguments' => array('nodequeue_admin_delete', 3),
130   'weight' => 5,
131   'type' => MENU_CALLBACK
132 );
133
134 /*$sqid = arg(5);
135 if (is_numeric($sqid) && $subqueue = nodequeue_load_subqueue($sqid)) {
136   // The following operations require a sqid and a nid.
137   $nid = arg(6);
138   if (in_array(arg(4), array('add', 'remove-node')) &&
139       is_numeric($nid)) {
140     $node = node_load($nid);
141   }*/
142
143 $items["admin/content/nodequeue/%nodequeue/add/%subqueue/%node"] = array(
144   'access arguments' => array(6, 3, 5),
145   'access callback' => 'nodequeue_node_and_queue_access',
146   'page callback' => 'nodequeue_admin_add_node',
147   'page arguments' => array(3, 5, 6),
148   'type' => MENU_CALLBACK
149 );
150 $items["admin/content/nodequeue/%nodequeue/remove-node/%subqueue/%node"] = array(
151   'access arguments' => array(6, 3, 5),
152   'access callback' => 'nodequeue_node_and_queue_access',
153   'page callback' => 'nodequeue_admin_remove_node',
154   'page arguments' => array(3, 5, 6),
155   'type' => MENU_CALLBACK
156 );
157 $items["admin/content/nodequeue/%nodequeue/up/%subqueue/%"] = array(
158   'access arguments' => array(3, 5),
159   'access callback' => 'nodequeue_queue_access',
160   'page callback' => 'nodequeue_admin_up',
161   'page arguments' => array(3, 5, 6),
162   'type' => MENU_CALLBACK
163 );
164 $items["admin/content/nodequeue/%nodequeue/down/%subqueue/%"] = array(
165   'access arguments' => array(3, 5),
166   'access callback' => 'nodequeue_queue_access',
167   'page callback' => 'nodequeue_admin_down',
168   'page arguments' => array(3, 5, 6),
169   'type' => MENU_CALLBACK
170 );
171 $items["admin/content/nodequeue/%nodequeue/front/%subqueue/%"] = array(
172   'access arguments' => array(3, 5),
173   'access callback' => 'nodequeue_queue_access',
174   'page callback' => 'nodequeue_admin_front',

```

```

175     'page arguments' => array(3, 5, 6),
176     'type' => MENU_CALLBACK
177 );
178 $items["admin/content/nodequeue/%nodequeue/back/%subqueue/%"] = array(
179     'access arguments' => array(3, 5),
180     'access callback' => 'nodequeue_queue_access',
181     'page callback' => 'nodequeue_admin_back',
182     'page arguments' => array(3, 5, 6),
183     'type' => MENU_CALLBACK
184 );
185 $items["admin/content/nodequeue/%nodequeue/remove/%subqueue/%"] = array(
186     'access arguments' => array(3, 5),
187     'access callback' => 'nodequeue_queue_access',
188     'page callback' => 'nodequeue_admin_remove',
189     'page arguments' => array(3, 5, 6),
190     'type' => MENU_CALLBACK
191 );
192 $items["admin/content/nodequeue/%nodequeue/clear/%subqueue"] = array(
193     'title' => 'Clear',
194     'access arguments' => array(3, 5),
195     'access callback' => 'nodequeue_queue_access',
196     'page callback' => 'drupal_get_form',
197     'page arguments' => array('nodequeue_clear_confirm', 3, 5),
198     'type' => MENU_CALLBACK
199 );
200
201 return $items;
202 }
203 /**
204  * Implementation of hook_nodeapi
205  */
206 function nodequeue_nodeapi(&$node, $op, $teaser = NULL, $page = NULL) {
207     switch ($op) {
208         case 'delete':
209             // If a node is being deleted, ensure it's also removed from any queues.
210             $result = db_query("SELECT qid, sqid FROM {nodequeue_nodes} WHERE nid = %d", $node->nid);
211             while ($obj = db_fetch_object($result)) {
212                 // If the queue is being tracked by translation set and the node is part
213                 // of a translation set, don't delete the queue record.
214                 // Instead, data will be updated in the 'translation_change' op, below.
215                 $queues = nodequeue_load_queues(array($obj->qid));
216                 $queue = nodequeue_load_queues(array($obj->qid));
217                 if (!$queue->i18n || empty($node->tnid)) {
218                     // This removes by nid, not position, because if we happen to have a
219                     // node in a queue twice, the 2nd position would be wrong.
220                     nodequeue_subqueue_remove_node($obj->sqid, $node->nid);
221                 }
222             }
223             break;
224         case 'translation_change':
225             if (isset($node->translation_change)) {
226                 // If there is only one node remaining, track by nid rather than tnid.
227                 // Otherwise, use the new tnid.
228                 $content_id = $node->translation_change['new_tnid'] == 0 ? $node->translation_change['remaining_nid'] : $node->translation_change['new_tnid'];
229                 db_query("UPDATE {nodequeue_nodes} SET nid = %d WHERE nid = %d", $content_id, $node->translation_change['old_tnid']);
230             }
231     }
232 }
233
234 /**
235  * Implementation of hook_link
236  */
237 function nodequeue_link($type, $node = NULL, $teaser = FALSE) {
238     $links = array();
239
240     if ($type == 'node' &&
241         variable_get('nodequeue_links', FALSE) &&
242         user_access('manipulate queues')) {
243         $queues = nodequeue_load_queues_by_type($node->type, 'links');
244         $subqueues = nodequeue_get_subqueues_by_node($queues, $node);
245         if (empty($subqueues)) {
246             return;
247         }
248
249         // resort the subqueues to retain consistent ordering:
250
251         ksort($subqueues);
252         // Due to caching, we can accidentally get positions leftover
253         // from previous iterations on teaser list pages, so we must
254         // remove any existing positions here.
255         foreach ($subqueues as $id => $subqueue) {
256             unset($subqueues[$id]->position);
257         }
258
259         if(!module_exists('translation')) {
260             nodequeue_set_subqueue_positions($subqueues, $node->nid);
261         }
262     }

```

```

263   foreach ($subqueues as $subqueue) {
264     $queue = $queues[$subqueue->qid];
265     $id = nodequeue_get_content_id($queue, $node);
266     if(module_exists('translation')) {
267       $subqueue = array($subqueue->sqid => $subqueue);
268       nodequeue_set_subqueue_positions($subqueue, $id);
269       $subqueue = array_shift($subqueue);
270     }
271     $query_string = nodequeue_get_query_string($id, TRUE);
272     $class = 'nodequeue-ajax-toggle nodequeue-toggle-q-'. $queue->qid. ' nodequeue-toggle-sq-'. $subqueue->sqid . ' nodequeue-toggle-ref-'. $subqueue->refere
273     if (!isset($subqueue->position)) {
274       $links[$class] = array(
275         'title' => nodequeue_title_substitute($queue->link, $queue, $subqueue),
276         'href' => "admin/content/nodequeue/$queue->qid/add/$subqueue->sqid/$id",
277         'attributes' => array('class' => $class . ' toggle-add'),
278         'query' => $query_string);
279     }
280     else if ($queue->link_remove) {
281       $links[$class] = array(
282         'title' => nodequeue_title_substitute($queue->link_remove, $queue, $subqueue),
283         'href' => "admin/content/nodequeue/$queue->qid/remove-node/$subqueue->sqid/$id",
284         'attributes' => array('class' => $class . ' toggle-remove'),
285         'query' => $query_string);
286     }
287   }
288   drupal_add_js(drupal_get_path('module', 'nodequeue') . '/nodequeue.js');
289   drupal_add_css(drupal_get_path('module', 'nodequeue') . '/nodequeue.css');
290 }
291 return $links;
292 }
293
294 /**
295  * Implementation of hook_views_api()
296  */
297 function nodequeue_views_api() {
298   return array(
299     'api' => 2,
300     'path' => drupal_get_path('module', 'nodequeue') . '/includes/views',
301   );
302 }
303
304 /**
305  * Implementation of hook_form_$form-id_alter().
306  */
307 function nodequeue_form_apachesolr_search_bias_form_alter(&$form, $form_state) {
308   // setup for the form building
309   $weights = drupal_map_assoc(array('21.0', '13.0', '8.0', '5.0', '3.0', '2.0', '1.0', '0.8', '0.5', '0.3', '0.2', '0.1'));
310   $weights['0'] = t('Normal');
311   $queues = nodequeue_load_subqueues_by_queue(array_keys(nodequeue_get_all_qids()));
312
313   // build the form
314   $form['biasing']['nodequeue_boost'] = array(
315     '#type' => 'fieldset',
316     '#title' => t('Nodequeue Biasing'),
317     '#weight' => -5,
318     '#collapsible' => TRUE,
319     '#collapsed' => TRUE,
320   );
321   $form['biasing']['nodequeue_boost']['nodequeue_apachesolr_boost'] = array(
322     '#type' => 'item',
323     '#description' => t("Specify to bias the search result when a node is in a queue. Any value except <em>Normal</em> will increase the score for the given q
324   );
325   foreach ($queues as $sqid => $queue) {
326     $boost = variable_get("nodequeue_apachesolr_boost_$sqid", 0);
327     // add in setting for each queue
328     $form['biasing']['nodequeue_boost']['nodequeue_apachesolr_boost']['nodequeue_apachesolr_boost_$sqid'] = array(
329       '#type' => 'select',
330       '#title' => t('Weight for %title nodequeue', array('%title' => $queue->title)),
331       '#options' => $weights,
332       '#default_value' => $boost,
333     );
334   }
335 }
336
337 /**
338  * Implementation of hook_apachesolr_update_index().
339  */
340 function nodequeue_apachesolr_update_index(&$document, $node) {
341   $queues = nodequeue_load_queues(array_keys(nodequeue_get_all_qids()));
342   $subqueues = nodequeue_get_subqueues_by_node($queues, $node);
343
344   nodequeue_set_subqueue_positions($subqueues, $node->nid);
345   foreach ($subqueues as $sqid => $subqueue) {
346     if (!empty($subqueue->position)) {
347       $key = _nodequeue_solr_qid_key();
348       $document->$key = $sqid;
349     }
350   }

```

```

351 }
352
353 /**
354  * return the apachesolr index key for group id
355  */
356 function _nodequeue_solr_qid_key() {
357     $qid_key = array(
358         'index_type' => 'sint',
359         'multiple' => TRUE,
360         'name' => "nodequeue",
361     );
362
363     return apachesolr_index_key($qid_key);
364 }
365
366 /**
367  * Implementation of hook_apachesolr_modify_query().
368  */
369 function nodequeue_apachesolr_modify_query($query, &$amp;params, $caller) {
370     $queues = nodequeue_load_subqueues_by_queue(array_keys(nodequeue_get_all_qids()));
371     foreach ($queues as $sqid => $queue) {
372         $boost = variable_get("nodequeue_apachesolr_boost_$sqid", 0);
373         if (!empty($boost)) {
374             $params['bq'][] = _nodequeue_solr_qid_key() . "::$sqid^$boost";
375             $params['facet.field'][] = _nodequeue_solr_qid_key();
376         }
377     }
378 }
379
380 // -----
381 // Nodequeue Admin operations
382
383 /**
384  * Print the JSON output for our AJAX calls.
385  */
386 function nodequeue_js_output($label, $href, $count = NULL, $sqid = NULL) {
387     $return = new stdClass();
388     $return->status = 1;
389     $return->label = check_plain($label);
390     $return->href = $href;
391     if (isset($count)) {
392         $return->count = $count;
393     }
394     if (isset($sqid)) {
395         $return->sqid = $sqid;
396     }
397
398     drupal_json($return);
399     exit;
400 }
401
402 /**
403  * Return content id based on il8n settings
404  */
405 function nodequeue_get_content_id($queue, $node) {
406     return ($queue->il8n && !empty($node->tnid)) ? $node->tnid : $node->nid;
407 }
408
409 /**
410  * Page callback to add a node to a queue.
411  */
412 function nodequeue_admin_add_node($queue, $subqueue, $node) {
413     if (!nodequeue_check_token($node->nid)) {
414         return drupal_goto();
415     }
416     $id = nodequeue_get_content_id($queue, $node);
417     nodequeue_subqueue_add($queue, $subqueue, $id);
418
419     // Provide a response if this is javascript.
420     if (!empty($_POST['js'])) {
421         if (isset($_GET['tab'])) {
422             nodequeue_js_output(t('Remove from queue'),
423                 url("admin/content/nodequeue/$queue->qid/remove-node/$subqueue->sqid/$node->nid", array('query' => nodequeue_get_query_string($node->nid, TRUE, array(
424                     nodequeue_subqueue_size_text($queue->size, $queue->size ? min($subqueue->count, $queue->size) : $subqueue->count, FALSE),
425                     $subqueue->sqid);
426                 ),
427             else {
428                 nodequeue_js_output(nodequeue_title_substitute($queue->link_remove, $queue, $subqueue),
429                     url("admin/content/nodequeue/$queue->qid/remove-node/$subqueue->sqid/$node->nid", array('query' => nodequeue_get_query_string($node->nid, TRUE))));
430             }
431         }
432
433         // There should always be a destination set for this, so just goto wherever.
434         drupal_goto();
435     }
436
437 /**
438  * Page callback to remove a node from a queue.

```

```

439 */
440 function nodequeue_admin_remove_node($queue, $subqueue, $node) {
441     if (!nodequeue_check_token($node->nid)) {
442         return drupal_goto();
443     }
444     $id = nodequeue_get_content_id($queue, $node);
445     nodequeue_subqueue_remove_node($subqueue->sqid, $id);
446
447     // Provide a response if this is javascript.
448     if (!empty($_POST['js'])) {
449         if (isset($_GET['tab'])) {
450             nodequeue_js_output(t('Add to queue'),
451                 url("admin/content/nodequeue/$queue->qid/add/$subqueue->sqid/$node->nid", array('query' => nodequeue_get_query_string($node->nid, TRUE, array('tab')))),
452                 nodequeue_subqueue_size_text($queue->size, $subqueue->count - 1, FALSE),
453                 $subqueue->sqid);
454         }
455         else {
456             nodequeue_js_output(nodequeue_title_substitute($queue->link, $queue, $subqueue),
457                 url("admin/content/nodequeue/$queue->qid/add/$subqueue->sqid/$node->nid", array('query' => nodequeue_get_query_string($node->nid, TRUE))));
458         }
459     }
460
461     // There should always be a destination set for this, so just goto wherever.
462     drupal_goto();
463 }
464
465 /**
466  * Display the queue controls for a node.
467  *
468  * @param $node
469  *   The loaded $node; will be loaded by the hook_menu.
470  */
471 function nodequeue_node_tab($node) {
472     $output = '';
473
474     // moved from hook_menu due to architecture change. This function seems to only be called from menu anyway...
475     $queues = nodequeue_load_queues_by_type($node->type, 'tab');
476     if (!$queues) {
477         return FALSE;
478     }
479     $header = array();
480     $header[] = array('data' => t('Title'), 'class' => 'nodequeue-title');
481     if (variable_get('nodequeue_tab_display_max', 1)) {
482         $header[] = array('data' => t('Max nodes'), 'class' => 'nodequeue-max-nodes');
483     }
484     $header[] = array('data' => t('In queue'), 'class' => 'nodequeue-in-queue');
485     $header[] = array('data' => t('Operation'), 'class' => 'nodequeue-operation');
486     $subqueues = nodequeue_get_subqueues_by_node($queues, $node);
487
488     nodequeue_set_subqueue_positions($subqueues, $node->nid);
489
490     $rows = array();
491     foreach ($subqueues as $subqueue) {
492         $queue = $queues[$subqueue->qid];
493         if (empty($subqueue->position)) {
494             $op = l(
495                 t('Add to queue'),
496                 "admin/content/nodequeue/$queue->qid/add/$subqueue->sqid/$node->nid",
497                 array('attributes' => array('class' => 'nodequeue-ajax-toggle'),
498                     'query' => drupal_get_destination() . '&tab&'. nodequeue_get_token($node->nid))
499             );
500         }
501         else {
502             $op = l(
503                 t('Remove from queue'),
504                 "admin/content/nodequeue/$queue->qid/remove-node/$subqueue->sqid/$node->nid",
505                 array('attributes' => array('class' => 'nodequeue-ajax-toggle'),
506                     'query' => drupal_get_destination() . '&tab&'. nodequeue_get_token($node->nid))
507             );
508         }
509         $row = array();
510         $row[] = array(
511             'class' => 'nodequeue-title',
512             'data' => l(nodequeue_title_substitute($queue->subqueue_title, $queue, $subqueue), "admin/content/nodequeue/$queue->qid/view/$subqueue->sqid"),
513         );
514         if (variable_get('nodequeue_tab_display_max', 1)) {
515             $row[] = array('class' => 'nodequeue-max-nodes', 'data' => $queue->size ? $queue->size : t('Infinite'));
516         }
517         $row[] = array(
518             'id' => 'nodequeue-count-' . $subqueue->sqid,
519             'class' => 'nodequeue-in-queue',
520             'data' => nodequeue_subqueue_size_text($queue->size, $subqueue->count, FALSE)
521         );
522         $row[] = array('class' => 'nodequeue-operation', 'data' => $op);
523         $rows[] = $row;
524     }
525
526     $output .= theme('table', $header, $rows, array('class' => 'nodequeue-table'));

```

```

527 drupal_add_js(drupal_get_path('module', 'nodequeue') . '/nodequeue.js');
528 drupal_add_css(drupal_get_path('module', 'nodequeue') . '/nodequeue.css');
529 return $output;
530 }
531
532 /**
533  * Display a list of queues and their status for the administrator.
534  */
535 function nodequeue_view_queues() {
536   $output = theme('advanced_help_topic', 'nodequeue', 'about', 'icon') . ' ' . theme('advanced_help_topic', 'nodequeue', 'about', t('Click here for infor
537   // Fetch all of the queues.
538   $queues = nodequeue_load_queues(nodequeue_get_all_qids(25));
539   foreach ($queues as $queue) {
540     if (!nodequeue_queue_access($queue)) {
541       unset($queues[$queue->qid]);
542     }
543   }
544
545   if (empty($queues)) {
546     return $output . t('No nodequeues exist.');
```

```

615
616     $rows[] = array(
617         array('class' => 'nodequeue-title', 'data' => check_plain($queue->title)),
618         array('class' => 'nodequeue-max-nodes', 'data' => $queue->size == 0 ? t('Infinite') : $queue->size),
619         array('class' => 'nodequeue-subqueues', 'data' => $sub_text),
620         array('class' => 'nodequeue-operation', 'data' => implode(' | ', $operations)),
621     );
622 }
623
624 $output .= theme('table', $header, $rows);
625 $output .= theme('pager', NULL, 25);
626
627 return $output;
628 }
629
630 /**
631  * Display a list of subqueues for a queue and their sizes
632  */
633 function nodequeue_view_subqueues($queue) {
634     // Fetch all of the subqueues.
635     $subqueues = nodequeue_load_subqueues_by_queue($queue->qid);
636
637     $header = array(t('Title'), t('In queue'), t('Operation'));
638
639     $rows = array();
640     foreach ($subqueues as $subqueue) {
641         if (nodequeue_api_subqueue_access($subqueue, NULL, $queue)) {
642             $sub_text = nodequeue_subqueue_size_text($queue->size, $subqueue->count, FALSE);
643             $rows[] = array(
644                 array('class' => 'nodequeue-title', 'data' => check_plain($subqueue->title)),
645                 array('class' => 'nodequeue-subqueues', 'data' => $sub_text),
646                 array('class' => 'nodequeue-operation', 'data' => l(t('View'), "admin/content/nodequeue/$queue->qid/view/$subqueue->sqid"))
647             );
648         }
649     }
650
651     $output = '<div>'. t('Max nodes in queue: @size', array('@size' => $queue->size ? $queue->size : t("Infinite"))) . '</div>';
652     $output .= theme('table', $header, $rows);
653     $output .= theme('pager', NULL, 20);
654
655     return $output;
656 }
657
658 /**
659  * Add or edit a queue.
660  */
661 function nodequeue_edit_queue_form(&$form_state, $queue) {
662     $info = nodequeue_api_info();
663
664     // For adding queues.
665     if (is_string($queue)) {
666         // If the $queue is a string - name of a queue type, basically - then we test that it's a valid queue type.
667         $queue = strtolower($queue);
668         if (!isset($info[$queue])) {
669             return FALSE;
670         }
671         drupal_set_title(t('Add @type', array('@type' => strtolower($info[$queue]['title']))));
672         $queue = new nodequeue_queue($queue);
673     }
674     else {
675         drupal_set_title(t('Nodequeue @title', array('@title' => $queue->title)));
676     }
677
678     $form['description'] = array(
679         '#type' => 'fieldset',
680         '#title' => $info[$queue->owner]['title'],
681         '#description' => $info[$queue->owner]['description'],
682     );
683
684     $form['title'] = array(
685         '#type' => 'textfield',
686         '#title' => t('Title'),
687         '#default_value' => $queue->title,
688         '#size' => 50,
689         '#maxlength' => 64,
690         '#description' => t('Enter the name of the queue'),
691     );
692
693     // This is a value; as the default nodequeue implementation has just one
694     // queue per nodequeue, this field is totally redundant. Plugins can
695     // override this.
696     $form['subqueue_title'] = array(
697         '#type' => 'value',
698         '#value' => $queue->subqueue_title,
699     );
700     // The placeholder is put here so that modifiers have an easy way to put
701     // additional form widgets in a prominent spot but not before the title of
702     // the queue.

```

```

703 $form['placeholder'] = array();
704
705 $form['size'] = array(
706   '#type' => 'textfield',
707   '#title' => t('Queue size'),
708   '#default_value' => $queue->size,
709   '#size' => 2,
710   '#maxlength' => 2,
711   '#description' => t('The maximum number of nodes will appear in the queue. Enter 0 for no limit'),
712 );
713
714 $form['reverse'] = array(
715   '#type' => 'checkbox',
716   '#title' => t('Reverse in admin view'),
717   '#default_value' => $queue->reverse,
718   '#description' => t('Ordinarily queues are arranged with the front of the queue (where items will be removed) on top and the back (where items will be add
719 );
720
721 $form['link'] = array(
722   '#type' => 'textfield',
723   '#title' => t('Link "add to queue" text'),
724   '#default_value' => $queue->link,
725   '#size' => 40,
726   '#maxlength' => 40,
727   '#description' => t('If you want a link to add a node to a queue in the "links" section (next to "add new comment"), enter the text here. If left blank no
728 );
729
730 $form['link_remove'] = array(
731   '#type' => 'textfield',
732   '#title' => t('Link "remove from queue" text'),
733   '#default_value' => $queue->link_remove,
734   '#size' => 40,
735   '#maxlength' => 40,
736   '#description' => t('Enter the text for the corresponding link to remove a node from a queue. This may be blank (in which case no link will appear) but a
737 );
738
739
740 $result = db_query("SELECT r.* FROM {role} r LEFT JOIN {permission} p ON p.rid = r.rid WHERE p.perm LIKE '%manipulate queues%' ORDER BY r.name");
741
742 $roles = array();
743 while ($role = db_fetch_object($result)) {
744   $roles[$role->rid] = $role->name;
745 }
746
747 if (!empty($roles)) {
748   $form['roles'] = array(
749     '#type' => 'checkboxes',
750     '#title' => t('Roles'),
751     '#default_value' => is_array($queue->roles) ? $queue->roles : array(),
752     '#options' => $roles,
753     '#description' => t('Check each role that can add nodes to the queue. Be sure that roles you want to appear here have "manipulate queues" access in the
754 );
755 }
756 else {
757   $form['roles'] = array(
758     '#type' => 'value',
759     '#value' => array(),
760 );
761
762   $form['roles_markup'] = array(
763     '#value' => t('No roles have the "manipulate queues" permission, so only the administrator account will be able to add or remove items from this queue.'
764 );
765 }
766
767 $nodes = node_get_types('names');
768
769 $form['types'] = array(
770   '#type' => 'checkboxes',
771   '#title' => t('Types'),
772   '#default_value' => $queue->types,
773   '#options' => $nodes,
774   '#description' => t('Check each node type that can be added to this queue.'),
775 );
776
777 $form['i18n'] = array(
778   '#type' => 'radios',
779   '#title' => t('Internationalization'),
780   '#options' => array(
781     '1' => t('Treat translation nodes as a single node'),
782     '0' => t('Manually manage translated nodes'),
783 ),
784   '#default_value' => empty($queue->qid) && module_exists('translation_helpers') ? 0 : $queue->i18n,
785   '#description' => t('Treating translations as a single node allows users to add, remove and manipulate a node
786 in the queue regardless of which translation is acted upon; nodequeue will only act on the original node.
787 When manually managing translation nodes, Nodequeue will ignore the relationship between node translations;
788 each translation node must be added to the queue separately and will occupy a separate spot in the queue.
789 Changing this setting will <strong>not</strong> update content that is already in the queue.'),
790   '#access' => module_exists('translation_helpers'),

```

```

791 );
792
793 $form['submit'] = array(
794     '#type' => 'submit',
795     '#value' => t('Submit'),
796 );
797
798 $form['owner'] = array(
799     '#type' => 'value',
800     '#value' => $queue->owner,
801 );
802
803 $form['show_in_links'] = array(
804     '#type' => 'value',
805     '#value' => $queue->show_in_links,
806 );
807
808 $form['show_in_tab'] = array(
809     '#type' => 'value',
810     '#value' => $queue->show_in_tab,
811 );
812
813 $form['show_in_ui'] = array(
814     '#type' => 'value',
815     '#value' => $queue->show_in_ui,
816 );
817
818 $form['reference'] = array(
819     '#type' => 'value',
820     '#value' => $queue->reference,
821 );
822
823 $form['subqueues'] = array(
824     '#type' => 'value',
825     '#value' => $queue->subqueues,
826 );
827
828 if (isset($queue->qid)) {
829     $form[] = array(
830         '#type' => 'submit',
831         '#value' => t('Delete'),
832         '#validate' => array('nodequeue_edit_queue_form_delete_validate'),
833         '#submit' => array('nodequeue_edit_queue_form_delete_submit'),
834     );
835     $form['qid'] = array(
836         '#type' => 'value',
837         '#value' => $queue->qid,
838     );
839     $form['count'] = array(
840         '#type' => 'value',
841         '#value' => $queue->count,
842     );
843 }
844
845 nodequeue_api_queue_form($queue, $form);
846
847 return $form;
848 }
849
850 /**
851  * Validate function for the nodequeue_queue form.
852  */
853 function nodequeue_edit_queue_form_validate($form, &$form_state) {
854     if (empty($form_state['values']['title'])) {
855         form_set_error('title', t('Please enter a title for this queue.'));
856     }
857     $queue = (object) $form_state['values'];
858     // fix checkboxes
859     $queue->roles = array_keys(array_filter($queue->roles));
860     $queue->types = array_keys(array_filter($queue->types));
861
862     if (!isset($queue->qid)) {
863         $queue->new = TRUE;
864     }
865
866     nodequeue_api_queue_form_validate($queue, $form_state, $form);
867 }
868
869 /**
870  * Submit function for the nodequeue_queue form.
871  */
872 function nodequeue_edit_queue_form_submit($formid, &$form_state) {
873     $queue = (object) $form_state['values'];
874     // fix checkboxes
875     $queue->roles = array_keys(array_filter($queue->roles));
876     $queue->types = array_keys(array_filter($queue->types));
877
878     if (!isset($queue->qid)) {

```

```

879     $queue->new = TRUE;
880 }
881
882 // Modify show_in_links based on whether or not links are available.
883 $queue->show_in_links = !empty($queue->link) || !empty($queue->link_remove);
884
885 nodequeue_api_queue_form_submit($queue, $form_state);
886
887 $qid = nodequeue_save($queue); // sets $queue->qid if needed.
888
889 nodequeue_api_queue_form_submit_finish($queue, $form_state);
890
891 nodequeue_check_subqueue_sizes($queue);
892
893 if ($queue->new) {
894     $form_state['values']['qid'] = $qid;
895     drupal_set_message(t('The queue has been created.'));
896 }
897 else {
898     drupal_set_message(t('The queue has been updated.'));
899 }
900 $form_state['redirect'] = 'admin/content/nodequeue';
901 }
902
903 /**
904  * Delete-validate function for the nodequeue_queue form.
905  */
906 function nodequeue_edit_queue_form_delete_validate($form, &$form_state) {
907     // No validation for delete step! But we need to have this so the default validation isn't called.
908 }
909
910 /**
911  * Delete-submit function for the nodequeue_queue form.
912  */
913 function nodequeue_edit_queue_form_delete_submit($formid, &$form_state) {
914     $form_state['redirect'] = "admin/content/nodequeue/". $form_state['values']['qid'] ."/delete";
915 }
916
917 /**
918  * Confirm form to delete a queue
919  */
920 function nodequeue_admin_delete(&$form_state, $queue) {
921     drupal_set_title(t("Nodequeue '@title'", array('@title' => $queue->title)));
922     $form['qid'] = array('#type' => 'value', '#value' => $queue->qid);
923     return confirm_form($form,
924         t('Are you sure you want to delete "%title"?', array('%title' => $queue->title)),
925         $_GET['destination'] ? $_GET['destination'] : 'admin/content/nodequeue',
926         t('This action cannot be undone.'),
927         t('Delete'), t('Cancel')
928     );
929 }
930
931 /**
932  * Submit function for nodequeue delete
933  */
934 function nodequeue_admin_delete_submit($formid, &$form_state) {
935     if ($form_state['values']['confirm']) {
936         nodequeue_delete($form_state['values']['qid']);
937         drupal_set_message(t('The queue has been deleted.'));
938     }
939     $form_state['redirect'] = 'admin/content/nodequeue';
940 }
941
942 /**
943  * Page callback to view a queue.
944  */
945 function nodequeue_admin_view($queue, $subqueue = NULL) {
946     drupal_set_title(t("Nodequeue '@title'", array('@title' => $queue->title)));
947     $qid = $queue->qid;
948
949     // If the queue has just one subqueue, it gets special treatment.
950     if (!$subqueue->sqid) {
951         if ($queue->subqueues == 1) {
952             $subqueues = nodequeue_load_subqueues_by_queue($queue->qid);
953             $subqueue = array_shift($subqueues);
954         }
955         else {
956             // display subqueue list page.
957             return nodequeue_view_subqueues($queue);
958         }
959     }
960     else if ($subqueue->sqid) {
961         if (!nodequeue_api_subqueue_access($subqueue, NULL, $queue)) {
962             return drupal_not_found();
963         }
964         // Adjust properties of the page so our subqueue is in the right
965         // visual place.
966         drupal_set_title(t("Subqueue '@title'",

```

```

967     array('@title' => nodequeue_title_substitute($queue->subqueue_title, $queue, $subqueue))));
968     $breadcrumb = drupal_get_breadcrumb();
969     $breadcrumb[] = l($queue->title, "admin/content/nodequeue/$queue->qid");
970     drupal_set_breadcrumb($breadcrumb);
971 }
972 return nodequeue_arrange_subqueue($queue, $subqueue);
973 }
974
975 function nodequeue_arrange_subqueue_entry($queue, $subqueue, $node) {
976     $qid = $queue->qid;
977     $sqid = $subqueue->sqid;
978     $query_string = nodequeue_get_query_string($node->position, TRUE);
979
980     $buttons = l(
981         theme('image', drupal_get_path('module', 'nodequeue') . '/images/go-up.png', t('Move up')),
982         "admin/content/nodequeue/$qid/up/$sqid/$node->position",
983         array('attributes' => array(
984             'title' => t('Move up'),
985             'class' => 'nodequeue-move-up',
986         )), 'query' => $query_string, 'html' => TRUE));
987     $buttons .= l(
988         theme('image', drupal_get_path('module', 'nodequeue') . '/images/go-down.png', t('Move down')),
989         "admin/content/nodequeue/$qid/down/$sqid/$node->position",
990         array('attributes' => array(
991             'title' => t('Move down'),
992             'class' => 'nodequeue-move-down',
993         )), 'query' => $query_string, 'html' => TRUE));
994     $buttons .= l(
995         theme('image', drupal_get_path('module', 'nodequeue') . '/images/go-top.png', t('Move to front')),
996         "admin/content/nodequeue/$qid/front/$sqid/$node->position",
997         array('attributes' => array(
998             'title' => t('Move to front'),
999             'class' => 'nodequeue-move-front',
1000         )), 'query' => $query_string, 'html' => TRUE));
1001     $buttons .= l(
1002         theme('image', drupal_get_path('module', 'nodequeue') . '/images/go-bottom.png', t('Move to back')),
1003         "admin/content/nodequeue/$qid/back/$sqid/$node->position",
1004         array('attributes' => array(
1005             'title' => t('Move to back'),
1006             'class' => 'nodequeue-move-back',
1007         )), 'query' => $query_string, 'html' => TRUE));
1008     $buttons .= l(
1009         theme('image', drupal_get_path('module', 'nodequeue') . '/images/delete.png', t('Remove from queue')),
1010         "admin/content/nodequeue/$qid/remove/$sqid/$node->position",
1011         array('attributes' => array(
1012             'title' => t('Remove from queue'),
1013             'class' => 'nodequeue-remove',
1014         )), 'query' => $query_string, 'html' => TRUE));
1015
1016     $output = "<tr id='nodequeue-row-'. $node->position .'' class='nodequeue-row ' . ($node->position % 2 ? 'odd' : 'even') .'>";
1017     $output .= "<td>". l($node->title, "node/$node->nid") . "</td>";
1018     $output .= "<td>". theme('username', $node) . "</td>";
1019     $output .= "<td>". format_date($node->created) . "</td>";
1020     $output .= "<td>". $buttons . "</td>";
1021     $output .= "</tr>";
1022     return $output;
1023 }
1024
1025 /**
1026  * View the contents of a subqueue, with links to re-order the queue.
1027  */
1028 function nodequeue_arrange_subqueue($queue, $subqueue) {
1029     $qid = $queue->qid;
1030     $sqid = $subqueue->sqid;
1031     $output = '';
1032
1033     $order = $queue->reverse ? 'DESC' : 'ASC';
1034     $result = db_query("SELECT DISTINCT(n.nid), n.title, n.uid, u.name, n.created, nq.position FROM {node} n LEFT JOIN {users} u on n.uid = u.uid LEFT JOIN {nod
1035
1036     $body = '';
1037
1038     $nids = array();
1039     while ($node = db_fetch_object($result)) {
1040         $nids[$node->position] = $node->nid;
1041         $body .= nodequeue_arrange_subqueue_entry($queue, $subqueue, $node);
1042     }
1043
1044     $output = "<p>". t('Max nodes in queue: @size', array('@size' => $queue->size ? $queue->size : t('Infinite'))) . "</p>";
1045
1046     $output .= "<p class='nodequeue-hide-if-not-js nodequeue-warning'>";
1047     $output .= t('Changes made to the queue order and queue removals will not be active until you click Save, below. If you add more nodes than the queue can ho
1048     $output .= "</p>";
1049
1050     $output .= "<table id='nodequeue-table'>";
1051     $output .= "<thead>";
1052     $output .= "<tr>";
1053     $output .= "<th class='nodequeue-node'>". t('Node') . "</th>";
1054     $output .= "<th class='nodequeue-author'>". t('Author') . "</th>";

```

```

1055 $output .= '<th class="nodequeue-date">'. t('Date Created') . '</th>';
1056 $output .= '<th class="nodequeue-operation">'. t('Operation') . '</th>';
1057 $output .= '</thead>';
1058 $output .= '<tbody>'. $body . '</tbody>';
1059 $output .= '</table>';
1060
1061 $output .= drupal_get_form('nodequeue_arrange_subqueue_form', $queue, $sqid, $nids);
1062 drupal_add_js(drupal_get_path('module', 'nodequeue') . '/nodequeue.js');
1063
1064 return $output;
1065 }
1066
1067 /**
1068 * Form used for arranging a queue
1069 */
1070 //TODD: Form - Revise for D6
1071 function nodequeue_arrange_subqueue_form($form_state, $queue, $sqid, $nids) {
1072   $form['qid'] = array(
1073     '#type' => 'value',
1074     '#value' => $queue->qid,
1075   );
1076
1077   $form['sqid'] = array(
1078     '#type' => 'hidden',
1079     '#value' => $sqid,
1080   );
1081
1082   $form['order'] = array(
1083     '#type' => 'hidden',
1084     '#id' => 'nodequeue-order',
1085     '#default_value' => implode(',', array_keys($nids)),
1086   );
1087
1088   $form['add'] = array(
1089     '#type' => 'textfield',
1090     '#title' => t('Select title to add'),
1091     '#autocomplete_path' => "nodequeue/autocomplete/$sqid",
1092     '#default_value' => '',
1093   );
1094
1095   // For use by the validate handler
1096   $form['nid'] = array(
1097     '#type' => 'value',
1098     '#value' => 0,
1099   );
1100
1101   $form['add_submit'] = array(
1102     '#type' => 'submit',
1103     '#attributes' => array('class' => 'nodequeue-add'),
1104     '#value' => t('Add'),
1105     '#validate' => array('nodequeue_arrange_subqueue_form_add_validate'),
1106     '#submit' => array('nodequeue_arrange_subqueue_form_add_submit'),
1107   );
1108
1109   $form['save'] = array(
1110     '#type' => 'submit',
1111     '#attributes' => array('class' => 'nodequeue-hide-if-not-js-hide nodequeue-save'),
1112     '#value' => t('Save'),
1113     '#validate' => array('nodequeue_arrange_subqueue_form_save_validate'),
1114   );
1115
1116   $form['clear'] = array(
1117     '#type' => 'submit',
1118     '#attributes' => array('class' => 'nodequeue-clear'),
1119     '#value' => t('Clear'),
1120     '#submit' => array('nodequeue_arrange_subqueue_form_clear_submit'),
1121   );
1122
1123   $form['reverse_click'] = array(
1124     '#type' => 'submit',
1125     '#attributes' => array('class' => 'nodequeue-reverse'),
1126     '#value' => t('Reverse'),
1127     '#submit' => array('nodequeue_arrange_subqueue_form_submit'),
1128   );
1129
1130   $form['shuffle'] = array(
1131     '#type' => 'submit',
1132     '#attributes' => array('class' => 'nodequeue-shuffle'),
1133     '#value' => t('Shuffle'),
1134     '#submit' => array('nodequeue_arrange_subqueue_form_shuffle_submit'),
1135   );
1136
1137   // Store the original order.
1138   $form['nids'] = array(
1139     '#type' => 'value',
1140     '#value' => $nids,
1141   );
1142

```

```

1143 $form['added_nids'] = array(
1144   '#type' => 'hidden',
1145   '#default_value' => '',
1146 );
1147
1148 $settings = array(
1149   'nodequeue-table' => array(
1150     // The gadget that stores our the order of items.
1151     'order' => 'input#nodequeue-order',
1152     // The buttons that do stuff.
1153     'up' => 'a.nodequeue-move-up',
1154     'down' => 'a.nodequeue-move-down',
1155     'top' => 'a.nodequeue-move-front',
1156     'bottom' => 'a.nodequeue-move-back',
1157     'remove' => 'a.nodequeue-remove',
1158
1159     // The button that adds an item
1160     'add' => 'input.nodequeue-add',
1161     // The button to clear the queue
1162     'clear_list' => 'input.nodequeue-clear',
1163     // Path for js to shuffle the queue
1164     'shuffle' => 'input.nodequeue-shuffle',
1165     // Path for js to reverse the queue
1166     'reverse' => 'input.nodequeue-reverse',
1167     // Path for ajax on adding an item
1168     'path' => url('nodequeue/ajax/add', array('absolute' => TRUE)),
1169     // Which items to post when adding
1170     'post' => array('#edit-sqid', '#edit-add'),
1171     // Where to get the id of an item
1172     'tr' => 'nodequeue-row-',
1173     'row_class' => 'tr.nodequeue-row',
1174     // Where to put the extra (we're storing a nid)
1175     'extra' => '#edit-added-nids',
1176     // What item to focus on ready
1177     'focus' => '#edit-add',
1178     // What item(s) to clear after add
1179     'clear' => array('#edit-add'),
1180     // What hidden class to show as soon as anything has changed that needs saving.
1181     'hidden' => '.nodequeue-js-hide',
1182     // Do we add to the top or the bottom?
1183     'add_location' => $queue->reverse ? 'top' : 'bottom',
1184   ),
1185 );
1186 drupal_add_js(array('nodequeue' => $settings), 'setting');
1187 drupal_add_css(drupal_get_path('module', 'nodequeue') . '/nodequeue.css');
1188
1189 return $form;
1190 }
1191
1192 /**
1193  * Implements hook_theme.
1194  *
1195  * @return unknown
1196  */
1197 function nodequeue_theme() {
1198   return array(
1199     'nodequeue_arrange_subqueue_form' => array(
1200       'arguments' => array('form'),
1201     ),
1202     'nodequeue_subqueue_empty_text' => array(
1203       'arguments' => array(),
1204     ),
1205     'nodequeue_subqueue_full_text' => array(
1206       'arguments' => array(),
1207     ),
1208     'nodequeue_subqueue_count_text' => array(
1209       'arguments' => array(),
1210     ),
1211   );
1212 }
1213
1214 //TODO: Theme function for Form - do we need to revise?
1215 function theme_nodequeue_arrange_subqueue_form($form) {
1216   $header = array(
1217     check_plain($form['add']['#title']),
1218     '',
1219   );
1220
1221   unset($form['add']['#title']);
1222   $rows = array(
1223     drupal_render($form['add']),
1224     array('data' => drupal_render($form['add_submit']), 'width' => '80%'),
1225   );
1226
1227   $output = theme('table', $header, array($rows));
1228   $output .= drupal_render($form);
1229   return $output;
1230 }

```

```

1231
1232/**
1233 * Validate handler for nodequeue_arrange_subqueue_form
1234 */
1235function nodequeue_arrange_subqueue_form_validate($form, &$form_state) {
1236 // Default Validator - does nothing
1237}
1238
1239/**
1240 * Validate handler for nodequeue_arrange_subqueue_form for 'Add' button.
1241 *
1242 * @param unknown_type $form_id
1243 * @param unknown_type $form_state
1244 * @param unknown_type $form
1245 */
1246function nodequeue_arrange_subqueue_form_add_validate($form, &$form_state) {
1247 $queue = nodequeue_load($form_state['values']['qid']);
1248 $subqueue = nodequeue_load($form_state['values']['sqid']);
1249 $nodes = nodequeue_api_autocomplete($queue, $subqueue, $form_state['values']['add']);
1250 if (empty($nodes) || !is_array($nodes)) {
1251   form_error($form['add'], t('Invalid node'));
1252   return;
1253 }
1254 if (count($nodes) > 1) {
1255   form_error($form['add'], t('That matches too many nodes'));
1256   return;
1257 }
1258 $keys = array_keys($nodes);
1259 $nid = array_pop($keys);
1260 form_set_value($form['nid'], $nid, $form_state);
1261}
1262
1263/**
1264 * Validate handler for nodequeue_arrange_subqueue_form for 'Save' button.
1265 *
1266 * @param unknown_type $form_id
1267 * @param unknown_type $form_state
1268 * @param unknown_type $form
1269 */
1270function nodequeue_arrange_subqueue_form_save_validate($form, &$form_state) {
1271 $nids = $form_state['values']['nids'];
1272
1273 // We can't use array_merge because it'll reset our keys and we can't
1274 // use + because it will overwrite.
1275 if ($form_state['values']['added_nids']) {
1276   foreach (explode(',', $form_state['values']['added_nids']) as $nid) {
1277     if (empty($nids)) {
1278       $nids[1] = $nid;
1279     }
1280     else {
1281       $nids[max(array_keys($nids)) + 1] = $nid;
1282     }
1283   }
1284 }
1285 form_set_value($form['nids'], $nids, $form_state);
1286}
1287
1288/**
1289 * Submit function for nodequeue_arrange_subqueue_form on 'Reverse' button.
1290 *
1291 * Yeah, this just calls the below function with a different parameter, but in D6 we're not supposed to use the $form['ops'].
1292 */
1293function nodequeue_arrange_subqueue_form_reverse_submit($form, &$form_state) {
1294 nodequeue_arrange_subqueue_form_submit($form, $form_state, TRUE, FALSE);
1295}
1296
1297function nodequeue_arrange_subqueue_form_shuffle_submit($form, &$form_state) {
1298 nodequeue_arrange_subqueue_form_submit($form, $form_state, FALSE, TRUE);
1299}
1300
1301/**
1302 * Submit function for nodequeue_arrange_subqueue_form
1303 */
1304//TOD0: Form Submit - Revise for D6
1305function nodequeue_arrange_subqueue_form_submit($form, &$form_state, $reverse=FALSE, $shuffle=FALSE) {
1306 // Add a node to the queue if that's the intention.
1307 $queue = nodequeue_load($form_state['values']['qid']);
1308 $subqueue = nodequeue_load_subqueue($form_state['values']['sqid']);
1309
1310 db_query("DELETE FROM {nodequeue_nodes} WHERE sqid = %d", $form_state['values']['sqid']);
1311 if ($form_state['values']['order']) {
1312   $now = time();
1313   $sql = '';
1314   $args = array();
1315   $nids = $form_state['values']['nids'];
1316   $subqueue->count = 0;
1317   $order = explode(',', $form_state['values']['order']);
1318   if ($queue->reverse xor $reverse) {

```

```

1319     $order = array_reverse($order);
1320 }
1321
1322 foreach ($order as $new_pos => $old_pos) {
1323     if ($sql) {
1324         $sql .= ', ' . ' ';
1325     }
1326     $sql .= ' (%d, %d, %d, %d, %d)';
1327     $args[] = $form_state['values']['sqid'];
1328     $args[] = $form_state['values']['qid'];
1329     $args[] = $nids[$old_pos];
1330
1331     // $new_pos starts from 0 but we start from 1.
1332     $args[] = $new_pos + 1;
1333     $args[] = $now;
1334     $subqueue->count++;
1335 }
1336 $sql = "INSERT INTO {nodequeue_nodes} (sqid, qid, nid, position, timestamp) VALUES $sql";
1337 db_query($sql, $args);
1338 if ($queue->size) {
1339     // 0 means infinity so never do this if false
1340     nodequeue_check_subqueue_size($queue, $subqueue);
1341 }
1342 if ($shuffle) { // $form_values['op'] == t('Shuffle') {
1343     nodequeue_subqueue_shuffle($subqueue);
1344 }
1345 }
1346 drupal_set_message(t('The queue has been updated'));
1347 }
1348
1349 function nodequeue_arrange_subqueue_form_clear_submit($form, &$form_state) {
1350     $form_state['redirect'] = 'admin/content/nodequeue/'. $form_state['values']['qid'] . '/clear/'. $form_state['values']['sqid'];
1351 }
1352
1353 function nodequeue_arrange_subqueue_form_add_submit($form, &$form_state) {
1354     $queue = nodequeue_load($form_state['values']['qid']);
1355     $subqueue = nodequeue_load_subqueue($form_state['values']['sqid']);
1356
1357     nodequeue_subqueue_add($queue, $subqueue, $form_state['values']['nid']);
1358 }
1359
1360 /**
1361  * Page callback to move an item up in a queue. This will be used only if
1362  * javascript is disabled in the client, and is a fallback technique.
1363  */
1364 function nodequeue_admin_up($queue, $subqueue, $pos) {
1365     if (!is_numeric($pos) || !is_numeric($subqueue->sqid) || !nodequeue_check_token($pos)) {
1366         return drupal_goto();
1367     }
1368     $subqueue = nodequeue_load_subqueue($subqueue);
1369     // This function is safe if $pos is out of bounds.
1370     if (!$queue->reverse) {
1371         nodequeue_queue_up($subqueue, $pos);
1372     }
1373     else {
1374         nodequeue_queue_down($subqueue, $pos);
1375     }
1376
1377     drupal_goto();
1378 }
1379
1380 /**
1381  * Page callback to move an item down in a queue. This will be used only if
1382  * javascript is disabled in the client, and is a fallback technique.
1383  */
1384 function nodequeue_admin_down($queue, $subqueue, $pos) {
1385     if (!is_numeric($pos) || !is_numeric($subqueue->sqid) || !nodequeue_check_token($pos)) {
1386         return drupal_goto();
1387     }
1388     $subqueue = nodequeue_load_subqueue($subqueue);
1389     // This function is safe if $pos is out of bounds.
1390     if ($queue->reverse) {
1391         nodequeue_queue_up($subqueue, $pos);
1392     }
1393     else {
1394         nodequeue_queue_down($subqueue, $pos);
1395     }
1396
1397     drupal_goto();
1398 }
1399
1400 /**
1401  * Page callback to move an item to the front of a queue. This will be used
1402  * only if javascript is disabled in the client, and is a fallback technique.
1403  */
1404 function nodequeue_admin_front($queue, $subqueue, $pos) {
1405     if (!is_numeric($pos) || !is_numeric($subqueue->sqid) || !nodequeue_check_token($pos)) {
1406         return drupal_goto();

```

```

1407 }
1408 $subqueue = nodequeue_load_subqueue($subqueue);
1409 // This function is safe if $pos is out of bounds.
1410 if (!$queue->reverse) {
1411     nodequeue_queue_front($subqueue, $pos);
1412 }
1413 else {
1414     nodequeue_queue_back($subqueue, $pos);
1415 }
1416
1417 drupal_goto();
1418 }
1419
1420 /**
1421  * Page callback to move an item to the back of a queue. This will be used
1422  * only if javascript is disabled in the client, and is a fallback technique.
1423  */
1424 function nodequeue_admin_back($queue, $subqueue, $pos) {
1425     if (!is_numeric($pos) || !is_numeric($subqueue->sqid) || !nodequeue_check_token($node->nid)) {
1426         return drupal_goto();
1427     }
1428     $subqueue = nodequeue_load_subqueue($subqueue);
1429     // This function is safe if $pos is out of bounds.
1430     if ($queue->reverse) {
1431         nodequeue_queue_front($subqueue, $pos);
1432     }
1433     else {
1434         nodequeue_queue_back($subqueue, $pos);
1435     }
1436
1437     drupal_goto();
1438 }
1439
1440 /**
1441  * Page callback to remove an item from a queue. This will be used only
1442  * if javascript is disabled in the client, and is a fallback technique.
1443  * This differs from nodequeue_admin_remove_node in that it removes a
1444  * specific position, which is necessary in case a node is in a queue
1445  * multiple times.
1446  */
1447 function nodequeue_admin_remove($queue, $subqueue, $pos) {
1448     if (!is_numeric($pos) || !is_numeric($subqueue->sqid) || !nodequeue_check_token($node->nid)) {
1449         return drupal_goto();
1450     }
1451
1452     nodequeue_subqueue_remove($subqueue, $pos);
1453
1454     drupal_goto();
1455 }
1456
1457 /**
1458  * Confirm form to clear a queue.
1459  */
1460 //TODO: Form - Revise for D6.
1461 function nodequeue_clear_confirm(&$form_state, $queue, $subqueue) {
1462     if (!is_numeric($subqueue)) {
1463         return false;
1464     }
1465     drupal_set_title(t("Nodequeue '@title'", array('@title' => $queue->title)));
1466     $form['sqid'] = array('#type' => 'value', '#value' => $subqueue->sqid);
1467     $form['qid'] = array('#type' => 'value', '#value' => $queue->qid);
1468     return confirm_form($form,
1469         t('Clearing queue "%s" is irreversible. You sure?', array('%s' => nodequeue_title_substitute($queue->subqueue_title, $queue, $subqueue))),
1470         $_GET['destination'] ? $_GET['destination'] : "admin/content/nodequeue/$queue->qid/view/$subqueue->sqid",
1471         t('This action cannot be undone.'),
1472         t('Clear'), t('Cancel')
1473     );
1474 }
1475
1476 /**
1477  * Submit function for nodequeue clear confirm
1478  */
1479 //TODO: Form Submit - Revise for D6
1480 function nodequeue_clear_confirm_submit($form, &$form_state) {
1481     if ($form_state['values']['confirm']) {
1482         nodequeue_queue_clear($form_state['values']['sqid']);
1483         $form_state['redirect'] = "admin/content/nodequeue/" . $form_state['values']['qid'] . "/view/" . $form_state['values']['sqid'];
1484     }
1485 }
1486
1487 /**
1488  * Page callback for autocomplete.
1489  */
1490 function nodequeue_autocomplete($sqid = NULL, $string = NULL) {
1491     $output = _nodequeue_autocomplete($sqid, $string);
1492
1493     drupal_json(drupal_map_assoc($output));
1494     exit;

```

```

1495 }
1496
1497 function _nodequeue_autocomplete($sqid, $string) {
1498     $output = array();
1499
1500     if (!$sqid || !is_numeric($sqid) || !$string) {
1501         return $output;
1502     }
1503
1504     $subqueue = nodequeue_load_subqueue($sqid);
1505     if (!$subqueue) {
1506         return $output;
1507     }
1508
1509     $queue = nodequeue_load($subqueue->qid);
1510     if (!$queue) {
1511         return $output;
1512     }
1513
1514     $nodes = nodequeue_api_autocomplete($queue, $subqueue, $string);
1515     return $nodes;
1516 }
1517
1518 /**
1519  * Page callback to ajaxily add a node.
1520  */
1521 function nodequeue_ajax_add() {
1522     $sqid = $_POST['sqid'];
1523     $position = $_POST['position'];
1524     $string = $_POST['add'];
1525     $output = _nodequeue_ajax_add($sqid, $position, $string);
1526     // let the world know this isn't normal output.
1527     drupal_json($output);
1528     exit;
1529 }
1530
1531 function _nodequeue_ajax_add($sqid, $position, $string) {
1532     if (!$string) {
1533         return array('error' => t('Invalid input'));
1534     }
1535
1536     if (!$sqid || !is_numeric($sqid)) {
1537         return array('error' => t('Invalid sqid'));
1538     }
1539
1540     $subqueue = nodequeue_load_subqueue($sqid);
1541     if (!$subqueue) {
1542         return array('error' => t('Invalid sqid'));
1543     }
1544
1545     $queue = nodequeue_load($subqueue->qid);
1546     if (!$queue) {
1547         return array('error' => t('Invalid sqid'));
1548     }
1549
1550     if (!nodequeue_api_subqueue_access($subqueue)) {
1551         return array('error' => t('Access denied'));
1552     }
1553
1554     $nodes = nodequeue_api_autocomplete($queue, $subqueue, $string);
1555     if (empty($nodes) || !is_array($nodes)) {
1556         return array('error' => t('Invalid node'));
1557     }
1558
1559     if (count($nodes) > 1) {
1560         return array('error' => t('That matches too many nodes'));
1561     }
1562
1563     $keys = array_keys($nodes);
1564     $node = node_load(array_pop($keys));
1565
1566     if (!node_access('view', $node)) {
1567         return array('error' => t('Invalid node'));
1568     }
1569
1570     $node->position = $position;
1571     return array(
1572         'status' => 1,
1573         'extra' => $node->nid,
1574         'max' => $queue->size,
1575         'data' => nodequeue_arrange_subqueue_entry($queue, $subqueue, $node),
1576     );
1577 }
1578
1579 // -----
1580 // Nodequeue manipulation API.
1581
1582 /**

```

```

1583 * @defgroup nodequeue_api
1584 * @{
1585 * Access to the internals of nodequeues are handled primarily through these
1586 * API functions. They allow easy loading of queues for manipulation.
1587 */
1588
1589 /**
1590 * The nodequeue queue class; the constructor makes it so we don't have to
1591 * always check to see if our variables are empty or not.
1592 */
1593 class nodequeue_queue {
1594     var $title = '';
1595     var $size = 0;
1596     var $link = '';
1597     var $link_remove = '';
1598     var $roles = array();
1599     var $types = array();
1600     var $show_in_links = TRUE;
1601     var $show_in_tab = TRUE;
1602     var $show_in_ui = TRUE;
1603     var $reference = 0;
1604     var $i18n = 1;
1605
1606     var $subqueue_title = '';
1607     var $reverse = FALSE;
1608
1609     // runtime
1610     var $subqueues = array();
1611     var $subqueue = NULL;
1612
1613     var $current = NULL;
1614
1615     function nodequeue_queue($type) {
1616         $this->owner = $type;
1617     }
1618 }
1619
1620 /**
1621 * Return TRUE If the specified account has access to manipulate this queue.
1622 */
1623 function nodequeue_queue_access($queue, $subqueue = NULL, $account = NULL) {
1624     if (!$account) {
1625         global $user;
1626         $account = $user;
1627     }
1628
1629     // Automatically true if all queues.
1630     if (user_access('manipulate all queues', $account)) {
1631         return TRUE;
1632     }
1633
1634     // Automatically false if they can't manipulate queues at all.
1635     if (!user_access('manipulate queues', $account) || empty($queue->roles)) {
1636         return FALSE;
1637     }
1638
1639     if ($subqueue) {
1640         return nodequeue_api_subqueue_access($subqueue, $account);
1641     }
1642
1643     if (!nodequeue_api_queue_access($queue, $account)) {
1644         return FALSE;
1645     }
1646
1647     $roles = array_keys((array) $account->roles) + array(DRUPAL_AUTHENTICATED_RID);
1648     return (bool) array_intersect($roles, $queue->roles);
1649 }
1650
1651 /**
1652 * Fetch a list of available queues for a given location. These queues
1653 * will be fully loaded and ready to go.
1654 */
1655 function nodequeue_load_queues_by_type($type, $location = NULL, $account = NULL, $bypass_cache = FALSE) {
1656     $qids = nodequeue_get_qids($type, $account, $bypass_cache);
1657     if ($location) {
1658         nodequeue_filter_qids($qids, $location);
1659     }
1660     return nodequeue_load_queues(array_keys($qids), $bypass_cache);
1661 }
1662
1663 /**
1664 * Used by menu system to determine access to the node and the queue in question.
1665 *
1666 * No, this isn't some odd hook_access implementation.
1667 *
1668 * @param unknown_type $node
1669 * @param unknown_type $queue
1670 * @return unknown

```

```

1671 */
1672 function nodequeue_node_and_queue_access($node, $queue, $subqueue = NULL) {
1673     return nodequeue_node_access($node->type) && nodequeue_queue_access($queue, $subqueue);
1674 }
1675
1676 function nodequeue_node_tab_access($node) {
1677     if (!user_access('manipulate queues')) {
1678         //For performance reasons: If the user can't manipulate queues, there is no reason to run the rest of these queries.
1679         return FALSE;
1680     }
1681     $queues = nodequeue_load_queues_by_type($node->type, 'tab');
1682     $subqueues = nodequeue_get_subqueues_by_node($queues, $node);
1683     if (empty($subqueues)) {
1684         return FALSE;
1685     }
1686     foreach ($subqueues as $subqueue) {
1687         if (!nodequeue_api_subqueue_access($subqueue)) {
1688             unset($subqueues[$subqueue->sqid]);
1689         }
1690     }
1691     return (variable_get('nodequeue_use_tab', 1) && !empty($subqueues));
1692 }
1693 /**
1694  * Return TRUE if $user can queue(s) for this node.
1695  *
1696  * @param $type
1697  *   The node type.
1698  * @param $location
1699  *   Optional argument. May be one of:
1700  *   - 'links': Only check for queues that have node links.
1701  *   - 'tab': Only check for queues that appear on the node tab.
1702  *   - 'ui': Only check for queues that appear in the UI.
1703  */
1704 function nodequeue_node_access($type, $location = NULL, $account = NULL) {
1705     $qids = nodequeue_get_qids($type, $account);
1706     if ($location) {
1707         nodequeue_filter_qids($qids, $location);
1708     }
1709     return !empty($qids);
1710 }
1711
1712
1713 /**
1714  * Filter a list of qids returned by nodequeue_get_qids to a location.
1715  *
1716  * @param $qids
1717  *   An array of $qids from @see nodequeue_get_qids()
1718  * @param $location
1719  *   One of:
1720  *   - 'links': Only check for queues that have node links.
1721  *   - 'tab': Only check for queues that appear on the node tab.
1722  *   - 'ui': Only check for queues that appear in the UI.
1723  */
1724 function nodequeue_filter_qids(&$qids, $location) {
1725     $var = "show_in_$location";
1726     foreach ($qids as $qid => $info) {
1727         if (empty($info->$var)) {
1728             unset($qids[$qid]);
1729         }
1730     }
1731 }
1732
1733 /**
1734  * Get an array of qids applicable to this node type.
1735  *
1736  * @param $type
1737  *   The node type.
1738  * @param $account
1739  *   The account to test against. Defaults to the currently logged in user.
1740  *
1741  * @return $qids
1742  *   An array in the format: @code { array($qid => array('qid' => $qid, 'show_in_tab' => true/false, 'show_in_links' => true/false) }
1743  *
1744  *
1745  * @param $bypass_cache
1746  *   Boolean value indicating whether to bypass the cache or not.
1747  */
1748 function nodequeue_get_qids($type, $account = NULL, $bypass_cache = FALSE) {
1749     if (!isset($account)) {
1750         global $user;
1751         $account = $user;
1752     }
1753
1754     static $cache = array();
1755     if ($bypass_cache || !isset($cache[$type])) {
1756         $roles_join = $roles_where = '';
1757         $roles = array();
1758     }

```

```

1759 // superuser always has access.
1760 if (!user_access('manipulate all queues', $account)) {
1761     $roles_join = "INNER JOIN {nodequeue_roles} nr ON nr.qid = nq.qid ";
1762     $roles = array_keys((array) $account->roles) + array(DRUPAL_AUTHENTICATED_RID);
1763
1764     $roles_where .= "AND nr.rid IN (". db_placeholders($roles, 'int') .")";
1765 }
1766
1767 $sql = 'SELECT nq.qid, nq.show_in_tab, nq.show_in_links, nq.show_in_ui, nq.i18n '.
1768     'FROM {nodequeue_queue} nq '.
1769     'INNER JOIN {nodequeue_types} nt ON nt.qid = nq.qid '. $roles_join .
1770     "WHERE nt.type = '%s' ". $roles_where;
1771 $result = db_query($sql, array_merge(array($type), $roles));
1772
1773 $qids = array();
1774 while ($qid = db_fetch_object($result)) {
1775     $qids[$qid->qid] = $qid;
1776 }
1777 $cache[$type] = $qids;
1778 }
1779 return $cache[$type];
1780 }
1781
1782 /**
1783  * Get an array all qids using the pager query. This administrative list
1784  * does no permission checking, so should only be available to users who
1785  * have passed the 'administer queues' check.
1786  *
1787  * @param $page_size
1788  *   The page size to use. If 0 will be all queues.
1789  * @param $pager_element
1790  *   In the rare event this should use another pager element, set this..
1791  *
1792  * @return $qids
1793  *   An array in the format: @code { array($qid => array('qid' => $qid, 'show_in_tab' '
1794  *   => true/false, 'show_in_links' => true/false }
1795  * @param $bypass_cache
1796  *   Boolean value indicating whether to bypass the cache or not.
1797  */
1798 function nodequeue_get_all_qids($page_size = 25, $pager_element = 0, $bypass_cache = FALSE) {
1799     static $cache = NULL;
1800     if ($bypass_cache || !isset($cache)) {
1801         $sql = 'SELECT nq.qid '.
1802             'FROM {nodequeue_queue} nq '.
1803             'WHERE nq.show_in_ui = 1 ';
1804         $count_sql = 'SELECT COUNT(q.qid) FROM {nodequeue_queue} q WHERE q.show_in_ui = 1 ';
1805         if ($page_size) {
1806             $result = pager_query($sql, $page_size, $pager_element, $count_sql);
1807         }
1808         else {
1809             $result = db_query($sql, $count_sql);
1810         }
1811
1812         $qids = array();
1813         while ($qid = db_fetch_object($result)) {
1814             $qids[$qid->qid] = $qid->qid;
1815         }
1816         $cache = $qids;
1817     }
1818     return $cache;
1819 }
1820
1821 /**
1822  * Load an array of $qids.
1823  *
1824  * This exists to provide a way of loading a bunch of queues with
1825  * the fewest queries. Loading 5 queues results in only 4 queries,
1826  * not 20. This also caches queues so that they don't get loaded
1827  * repeatedly.
1828  *
1829  * @param $qids
1830  *   An array of queue IDs to load.
1831  *
1832  * @param $bypass_cache
1833  *   Boolean value indicating whether to bypass the cache or not.
1834  */
1835 function nodequeue_load_queues($qids = array(), $bypass_cache = FALSE) {
1836     static $cache = array();
1837     $to_load = $loaded = array();
1838
1839     foreach ($qids as $qid) {
1840         if ($bypass_cache || !isset($cache[$qid])) {
1841             $to_load[] = $qid;
1842         }
1843     }
1844
1845     if (!empty($to_load)) {
1846         $placeholders = db_placeholders($to_load, 'int');

```

```

1847 $result = db_query("SELECT q.*, COUNT(s.sqid) AS subqueues FROM {nodequeue_queue} q LEFT JOIN {nodequeue_subqueue} s ON q.qid = s.qid WHERE q.qid IN ($pla
1848 while ($queue = db_fetch_object($result)) {
1849     $loaded[$queue->qid] = $queue;
1850     // ensure valid defaults:
1851     $loaded[$queue->qid]->types = array();
1852     $loaded[$queue->qid]->roles = array();
1853     $loaded[$queue->qid]->count = 0;
1854 }
1855
1856 $result = db_query("SELECT qid, rid FROM {nodequeue_roles} WHERE qid IN ($placeholders)", $to_load);
1857 while ($obj = db_fetch_object($result)) {
1858     $loaded[$obj->qid]->roles[] = $obj->rid;
1859 }
1860
1861 $result = db_query("SELECT qid, type FROM {nodequeue_types} WHERE qid IN ($placeholders)", $to_load);
1862 while ($obj = db_fetch_object($result)) {
1863     $loaded[$obj->qid]->types[] = $obj->type;
1864 }
1865 }
1866
1867 if ($bypass_cache) {
1868     return $loaded;
1869 }
1870 else {
1871     if (!empty($loaded)) {
1872         $cache += $loaded;
1873     }
1874     $queues = array();
1875     foreach ($sqids as $sqid) {
1876         if (isset($cache[$sqid])) {
1877             $queues[$sqid] = $cache[$sqid];
1878         }
1879     }
1880     return $queues;
1881 }
1882 }
1883
1884 /**
1885  * Load a nodequeue.
1886  *
1887  * @param $qid
1888  *   The qid of the queue to load.
1889  */
1890 function nodequeue_load($qid) {
1891     $queues = nodequeue_load_queues(array($qid));
1892     return array_shift($queues);
1893 }
1894
1895 /**
1896  * This function exists so that %subqueue will work in hook_menu.
1897  */
1898 function subqueue_load($sqid) {
1899     if (!$sqid) {
1900         return NULL;
1901     }
1902     return array_shift(nodequeue_load_subqueues(array($sqid)));
1903 }
1904
1905 /**
1906  * Load a list of subqueues
1907  *
1908  * This exists to provide a way of loading a bunch of queues with
1909  * the fewest queries. Loading 5 queues results in only 4 queries,
1910  * not 20. This also caches queues so that they don't get loaded
1911  * repeatedly.
1912  *
1913  * @param $sqids
1914  *   An array of subqueue IDs to load.
1915  * @param $bypass_cache
1916  *   Boolean value indicating whether to bypass the cache or not.
1917  */
1918 function nodequeue_load_subqueues($sqids, $bypass_cache = FALSE) {
1919     static $cache = array();
1920     $to_load = array();
1921
1922     foreach ($sqids as $sqid) {
1923         if ($bypass_cache || !isset($cache[$sqid])) {
1924             $to_load[] = $sqid;
1925         }
1926     }
1927
1928     if (!empty($to_load)) {
1929         $placeholders = db_placeholders($to_load, 'int');
1930
1931         $result = db_query("SELECT s.*, COUNT(n.position) AS count FROM {nodequeue_subqueue} s LEFT JOIN {nodequeue_nodes} n ON n.sqid = s.sqid WHERE s.sqid IN ($
1932         while ($obj = db_fetch_object($result)) {
1933             // Sometimes we want to get to subqueues by reference, sometimes by sqid.
1934             // sqid is always unique, but reference is sometimes more readily available.

```

```

1935     $cache[$obj->sqid] = $obj;
1936 }
1937 }
1938
1939 foreach ($sqids as $sqid) {
1940     if (isset($cache[$sqid])) {
1941         $subqueues[$sqid] = $cache[$sqid];
1942     }
1943 }
1944 return $subqueues;
1945 }
1946
1947 /**
1948  * Load a single subqueue.
1949  *
1950  * @param $sqid
1951  * The subqueue ID to load.
1952  * @param $bypass_cache
1953  * Boolean value indicating whether to bypass the cache or not.
1954  */
1955 function nodequeue_load_subqueue($sqid, $bypass_cache = FALSE) {
1956     $subqueues = nodequeue_load_subqueues(array($sqid), $bypass_cache);
1957     if ($subqueues) {
1958         return array_shift($subqueues);
1959     }
1960 }
1961 }
1962
1963 /**
1964  * Load the entire set of subqueues for a queue.
1965  *
1966  * This will load the entire set of subqueues for a given queue (and can
1967  * respect the pager, if desired). It does NOT cache the subqueues like
1968  * nodequeue_load_subqueues does, so beware of this mixed caching.
1969  *
1970  * @param $qids
1971  * A $qid or array of $qids
1972  * @param $page_size
1973  * If non-zero, use the pager_query and limit the page-size to the parameter.
1974  */
1975 function nodequeue_load_subqueues_by_queue($qids, $page_size = 0) {
1976     if (is_numeric($qids)) {
1977         $qids = array($qids);
1978     }
1979
1980     if (empty($qids)) {
1981         return array();
1982     }
1983
1984     $query = "SELECT s.*, COUNT(n.position) AS count FROM {nodequeue_subqueue} s LEFT JOIN {nodequeue_nodes} n ON n.sqid = s.sqid WHERE s.qid IN (". db_placehol
1985     if ($page_size) {
1986         $result = pager_query($query, $page_size, 0, $qids);
1987     }
1988     else {
1989         $result = db_query($query, $qids);
1990     }
1991
1992     $subqueues = array();
1993
1994     while ($subqueue = db_fetch_object($result)) {
1995         $subqueues[$subqueue->sqid] = $subqueue;
1996     }
1997
1998     return $subqueues;
1999 }
2000
2001 /**
2002  * Load a set of subqueues by reference.
2003  *
2004  * This can be used to load a set of subqueues by reference; it will primarily
2005  * be used by plugins that are managing subqueues.
2006  *
2007  * @param $references
2008  * A keyed array of references to load. The key is the $qid and each value
2009  * is another array of references.
2010  */
2011 function nodequeue_load_subqueues_by_reference($references, $bypass_cache = FALSE) {
2012     static $cache = array();
2013     $subqueues = array();
2014     if ($bypass_cache) {
2015         $cache = array();
2016     }
2017
2018     // build strings for the query based upon the qids and references.
2019     $keys = $values = array();
2020     foreach ($references as $qid => $qid_references) {
2021         $keys[$qid] = array();
2022         $qid_values = array();

```

```

2023     foreach ($qid_references as $reference) {
2024         // If we already have this qid/reference combo cached, don't add it to
2025         // our little list.
2026         if (isset($cache[$qid][$reference])) {
2027             $subqueues[$cache[$qid][$reference]->sqid] = $cache[$qid][$reference];
2028         }
2029         else {
2030             $keys[$qid][] = "%s"; // Substitution strings
2031             $qid_values[] = $reference; // Values to substitute
2032         }
2033     }
2034     if (!empty($keys[$qid])) {
2035         $values = array_merge($values, array($qid), $qid_values);
2036     }
2037     else {
2038         unset($keys[$qid]);
2039     }
2040 }
2041
2042 if (!empty($keys)) {
2043     $where = '';
2044     foreach ($keys as $key_list) {
2045         if ($where) {
2046             $where .= ' OR ';
2047         }
2048         $where .= 's.qid = %d AND s.reference IN ('. implode(' ', $key_list) .)';
2049     }
2050
2051     $result = db_query("SELECT s.*, COUNT(n.position) AS count FROM {nodequeue_subqueue} s LEFT JOIN {nodequeue_nodes} n ON n.sqid = s.sqid WHERE $where GROUP
2052
2053     while ($subqueue = db_fetch_object($result)) {
2054         $cache[$subqueue->qid][$subqueue->reference] = $subqueues[$subqueue->sqid] = $subqueue;
2055     }
2056 }
2057
2058 return $subqueues;
2059 }
2060
2061 /**
2062  * Save a nodequeue. This does not save subqueues; those must be added separately.
2063  */
2064 function nodequeue_save(&$queue) {
2065     if (!isset($queue->qid)) {
2066         db_query("INSERT INTO {nodequeue_queue} (title, subqueue_title, size, link, link_remove, owner, show_in_links, show_in_tab, show_in_ui, i18n, reverse, ref
2067         $queue->qid = db_last_insert_id('nodequeue_queue', 'qid');
2068         if (function_exists('views_invalidate_cache')) {
2069             views_invalidate_cache();
2070         }
2071     }
2072     else {
2073         db_query("UPDATE {nodequeue_queue} set size = %d, title = '%s', subqueue_title = '%s', link = '%s', link_remove = '%s', owner = '%s', show_in_links = %d,
2074         db_query("DELETE FROM {nodequeue_roles} WHERE qid = %d", $queue->qid);
2075         db_query("DELETE FROM {nodequeue_types} WHERE qid = %d", $queue->qid);
2076     }
2077
2078     if (is_array($queue->roles)) {
2079         foreach ($queue->roles as $rid)
2080             db_query("INSERT INTO {nodequeue_roles} (qid, rid) VALUES (%d, %d)", $queue->qid, $rid);
2081     }
2082
2083     if (is_array($queue->types)) {
2084         foreach ($queue->types as $type)
2085             db_query("INSERT INTO {nodequeue_types} (qid, type) VALUES (%d, '%s')", $queue->qid, $type);
2086     }
2087
2088     // set our global that tells us whether or not we need to activate hook_link
2089     if (db_result(db_query("SELECT COUNT(*) FROM {nodequeue_queue} WHERE link <> ''")) {
2090         variable_set('nodequeue_links', TRUE);
2091     }
2092     else {
2093         variable_set('nodequeue_links', FALSE);
2094     }
2095
2096     if (isset($queue->add_subqueue) && is_array($queue->add_subqueue)) {
2097         foreach ($queue->add_subqueue as $reference => $title) {
2098             // If reference is unset it should be set to the qid; this is generally
2099             // used for a single subqueue; setting the reference to the qid makes
2100             // it easy to find that one subqueue.
2101             if ($reference == 0) {
2102                 $reference = $queue->qid;
2103             }
2104             nodequeue_add_subqueue($queue, $title, $reference);
2105         }
2106     }
2107     return $queue->qid;
2108 }
2109
2110 /**

```

```

2111 * Delete a nodequeue.
2112 */
2113 function nodequeue_delete($qid) {
2114     db_query("DELETE FROM {nodequeue_queue} WHERE qid = %d", $qid);
2115     db_query("DELETE FROM {nodequeue_roles} WHERE qid = %d", $qid);
2116     db_query("DELETE FROM {nodequeue_types} WHERE qid = %d", $qid);
2117     db_query("DELETE FROM {nodequeue_nodes} WHERE qid = %d", $qid);
2118     db_query("DELETE FROM {nodequeue_subqueue} WHERE qid = %d", $qid);
2119 }
2120
2121 /**
2122  * Add a new subqueue to a queue.
2123  *
2124  * @param $qid
2125  *   The queue id. This should not be the full queue object.
2126  * @param $reference
2127  *   A reference that uniquely identifies this subqueue. If NULL it will
2128  *   be assigned to the sqid.
2129  */
2130 function nodequeue_add_subqueue(&$queue, $title, $reference = NULL) {
2131     if (empty($reference)) {
2132         $insert_reference = "";
2133     }
2134     else {
2135         $insert_reference = $reference;
2136     }
2137
2138     $subqueue = new stdClass();
2139     $subqueue->reference = $reference;
2140     $subqueue->qid = $queue->qid;
2141     $subqueue->title = $title;
2142
2143     db_query("INSERT INTO {nodequeue_subqueue} (qid, reference, title) VALUES (%d, '%s', '%s')", $queue->qid, $insert_reference, $title);
2144
2145     $subqueue->sqid = db_last_insert_id('nodequeue_subqueue', 'sqid');
2146
2147     // If somehow the $reference is null, here we set it to the sqid.
2148     // We have to do it here, because before the insert we don't know what the sqid will be.
2149     if (empty($reference)) {
2150         db_query("UPDATE {nodequeue_subqueue} SET reference = '%s' WHERE sqid = %d", $subqueue->sqid, $subqueue->sqid);
2151     }
2152
2153     return $subqueue;
2154 }
2155
2156 /**
2157  * Change the title of a subqueue.
2158  *
2159  * Note that only the title of a subqueue is changeable; it can change to
2160  * reflect updates in taxonomy term names, for example.
2161  */
2162 function nodequeue_subqueue_update_title($sqid, $title) {
2163     db_query("UPDATE {nodequeue_subqueue} SET title = '%s' WHERE sqid = %d", $title, $sqid);
2164 }
2165
2166 /**
2167  * Remove a subqueue.
2168  */
2169 function nodequeue_remove_subqueue($sqid) {
2170     nodequeue_queue_clear($sqid);
2171     db_query("DELETE FROM {nodequeue_subqueue} WHERE sqid = %d", $sqid);
2172 }
2173
2174 // -----
2175 // Queue position control
2176
2177 /**
2178  * Add a node to a queue.
2179  *
2180  * @param $queue
2181  *   The parent queue of the subqueue. This is required so that we can
2182  *   pop nodes out if the queue breaks size limits.
2183  * @param $sqid
2184  *   The subqueue ID to add the node to.
2185  * @param $nid
2186  *   The node ID
2187  */
2188 function nodequeue_subqueue_add($queue, &$subqueue, $nid) {
2189     // If adding this would make the queue too big, pop the front node
2190     // (or nodes) out.
2191
2192     if ($queue->size) {
2193         // 0 means infinity so never do this if false
2194         nodequeue_check_subqueue_size($queue, $subqueue, $queue->size - 1);
2195     }
2196
2197     db_query("INSERT INTO {nodequeue_nodes} (sqid, qid, nid, position, timestamp) VALUES (%d, %d, %d, %d, %d)", $subqueue->sqid, $queue->qid, $nid, $subqueue->c
2198     $subqueue->count++;

```

```

2199 if (module_exists('apachesolr')) {
2200     apachesolr_mark_node($nid);
2201 }
2202 }
2203
2204 /**
2205  * Remove a node from the queue. If a node is in the queue more than once,
2206  * only the first (closest to 0 position, or the front of the queue) will
2207  * be removed.
2208  *
2209  * @param $sqid
2210  *   The subqueue to remove nodes from.
2211  * @param $nid
2212  *   The node to remove.
2213  */
2214 function nodequeue_subqueue_remove_node($sqid, $nid) {
2215     if ($pos = nodequeue_get_subqueue_position($sqid, $nid)) {
2216         nodequeue_subqueue_remove($sqid, $pos);
2217         if (module_exists('apachesolr')) {
2218             apachesolr_mark_node($nid);
2219         }
2220     }
2221 }
2222 /**
2223  * Remove a node or node(s) from a nodequeue by position.
2224  *
2225  * If you know the nid but not the position, use
2226  * @see nodequeue_subqueue_remove_node() instead.
2227  *
2228  * @param $sqid
2229  *   The subqueue to remove nodes from.
2230  * @param $start
2231  *   The first position (starting from 1) to remove.
2232  * @param $end
2233  *   The last position to remove. If NULL or equal to $start,
2234  *   only one node will be removed. Thus if $start is 1 and $end is 2,
2235  *   the first and second items will be removed from the queue.
2236  *
2237  */
2238 function nodequeue_subqueue_remove($sqid, $start, $end = NULL) {
2239     if (!isset($end)) {
2240         $end = $start;
2241     }
2242
2243     $diff = $end - $start + 1;
2244     db_query("DELETE FROM {nodequeue_nodes} WHERE sqid = %d AND position >= %d AND position <= %d", $sqid, $start, $end);
2245     db_query("UPDATE {nodequeue_nodes} SET position = position - %d WHERE sqid = %d AND position > %d", $diff, $sqid, $end);
2246 }
2247
2248 /**
2249  * Empty a subqueue.
2250  *
2251  * @param $sqid
2252  *   The sqid to empty.
2253  */
2254 function nodequeue_queue_clear($sqid) {
2255     db_query("DELETE FROM {nodequeue_nodes} WHERE sqid = %d", $sqid);
2256 }
2257
2258 /**
2259  * Guarantee that a subqueue has not gotten too big. It's important to call
2260  * this after an operation that might have reduced a queue's maximum size.
2261  * It stores the count to save a query if this is to be followed by an add
2262  * operation.
2263  *
2264  * @param $queue
2265  *   The queue object.
2266  * @param $reference
2267  *   The subqueue to check.
2268  *
2269  */
2270 function nodequeue_check_subqueue_size($queue, &$subqueue, $size = NULL) {
2271     if (!isset($size)) {
2272         $size = $queue->size;
2273     }
2274
2275     if ($queue->size && $subqueue->count > $size) {
2276         nodequeue_subqueue_remove($subqueue->sqid, 1, $subqueue->count - $size);
2277         $subqueue->count = $size;
2278     }
2279 }
2280
2281 /**
2282  * Guarantee that all subqueues are within the size constraints set
2283  * by $queue->size.
2284  */
2285 function nodequeue_check_subqueue_sizes($queue) {
2286     // Don't check if size is 0, as that means infinite size.

```

```

2287 if (!$queue->size) {
2288     return;
2289 }
2290
2291 $subqueues = nodequeue_load_subqueues_by_queue($queue->qid);
2292 foreach ($subqueues as $subqueue) {
2293     nodequeue_check_subqueue_size($queue, $subqueue);
2294 }
2295 }
2296
2297 /**
2298  * Swap two positions within a subqueue.
2299  */
2300 function nodequeue_queue_swap($subqueue, $pos1, $pos2) {
2301     // Grab the nid off one of the positions so we can more easily swap.
2302     $nid = db_result(db_query("SELECT nid FROM {nodequeue_nodes} WHERE sqid = %d AND position = %d", $subqueue->sqid, $pos1));
2303     if (!$nid) {
2304         return;
2305     }
2306
2307     db_query("UPDATE {nodequeue_nodes} SET position = %d WHERE position = %d AND sqid = %d", $pos1, $pos2, $subqueue->sqid);
2308     db_query("UPDATE {nodequeue_nodes} SET position = %d WHERE nid = %d AND sqid = %d", $pos2, $nid, $subqueue->sqid);
2309 }
2310
2311 /**
2312  * Move a position within a subqueue up by one.
2313  */
2314 function nodequeue_queue_up($subqueue, $position) {
2315     if ($position < 2 || $position > $subqueue->count)
2316         return;
2317     nodequeue_queue_swap($subqueue, $position - 1, $position);
2318 }
2319
2320 /**
2321  * Move a position within a subqueue down by one.
2322  */
2323 function nodequeue_queue_down($subqueue, $position) {
2324     if ($position < 1 || $position >= $subqueue->count)
2325         return;
2326     nodequeue_queue_swap($subqueue, $position + 1, $position);
2327 }
2328
2329 /**
2330  * Move an item to the front of the queue.
2331  */
2332 function nodequeue_queue_front($subqueue, $position) {
2333     if ($position < 2 || $position > $subqueue->count)
2334         return;
2335     $entry = db_fetch_object(db_query("SELECT * FROM {nodequeue_nodes} WHERE sqid= %d AND position = %d", $subqueue->sqid, $position));
2336     db_query("DELETE FROM {nodequeue_nodes} WHERE sqid = %d AND position = %d", $subqueue->sqid, $position);
2337     db_query("UPDATE {nodequeue_nodes} SET position = position + 1 WHERE sqid= %d AND position < %d", $subqueue->sqid, $position);
2338     db_query("INSERT INTO {nodequeue_nodes} (qid, sqid, nid, position, timestamp) VALUES (%d, %d, %d, 1, %d)", $entry->qid, $subqueue->sqid, $entry->nid, $entry
2339 );
2340
2341 /**
2342  * Move an item to the back of the queue.
2343  */
2344 function nodequeue_queue_back($subqueue, $position) {
2345     if ($position < 1 || $position >= $subqueue->count)
2346         return;
2347     $entry = db_fetch_object(db_query("SELECT * FROM {nodequeue_nodes} WHERE sqid = %d AND position = %d", $subqueue->sqid, $position));
2348     db_query("DELETE FROM {nodequeue_nodes} WHERE sqid = %d AND position = %d", $subqueue->sqid, $position);
2349     db_query("UPDATE {nodequeue_nodes} SET position = position - 1 WHERE sqid = %d AND position > %d", $subqueue->sqid, $position);
2350     db_query("INSERT INTO {nodequeue_nodes} (qid, sqid, nid, position, timestamp) VALUES (%d, %d, %d, %d, %d)", $entry->qid, $subqueue->sqid, $entry->nid, $subq
2351 );
2352
2353 /**
2354  * Get the position of a node in a subqueue, or 0 if not found.
2355  */
2356 function nodequeue_get_subqueue_position($sqid, $nid) {
2357     // We use MIN to make sure we always get the closes to the front of the
2358     // queue in case the queue has nodes in it multiple times.
2359     $pos = db_result(db_query("SELECT MIN(position) FROM {nodequeue_nodes} WHERE sqid = %d AND nid = %d", $sqid, $nid));
2360     return $pos;
2361 }
2362
2363 /**
2364  * Get the position of a node in several subqueues.
2365  */
2366 function nodequeue_set_subqueue_positions(&$subqueues, $nid) {
2367     $placeholders = db_placeholders($subqueues, 'int');
2368     $args = array_keys($subqueues);
2369     $args[] = $nid;
2370     $result = db_query("SELECT sqid, MIN(position) AS position FROM {nodequeue_nodes} WHERE sqid IN ($placeholders) AND nid = %d GROUP BY sqid", $args);
2371     while ($obj = db_fetch_object($result)) {
2372         $subqueues[$obj->sqid]->position = $obj->position;
2373     }
2374 }

```

```

2375
2376/**
2377 * Get a list of valid subqueues for a node, along with the position of the node.
2378 *
2379 * @param $queues
2380 *   An array of fully loaded queue objects.
2381 * @param $node
2382 *   A fully loaded node object.
2383 *
2384 */
2385 function nodequeue_get_subqueues_by_node($queues, $node) {
2386     // Determine which subqueues are valid for each queue.
2387     $references= array();
2388     static $last_nid = 0;
2389     foreach ($queues as $queue) {
2390         if ($result = nodequeue_api_subqueues($queue, $node)) {
2391             $references[$queue->qid] = is_array($result) ? $result : array($result);
2392         }
2393     }
2394
2395     if (empty($references)) {
2396         return;
2397     }
2398     // only allow the static cache to be used if the nid is the same as the last
2399     $subqueues = nodequeue_load_subqueues_by_reference($references, ($last_nid != $node->nid));
2400     $last_nid = $node->nid;
2401
2402     return $subqueues;
2403 }
2404
2405/**
2406 * Get a textual representation of a nodequeue's queue size.
2407 */
2408 function nodequeue_subqueue_size_text($max, $count, $long = TRUE) {
2409     if (empty($count)) {
2410         $message = theme('nodequeue_subqueue_empty_text');
2411     }
2412     else if ($count == $max) {
2413         $message = theme('nodequeue_subqueue_full_text');
2414     }
2415     else {
2416         if ($long) {
2417             $message = theme('nodequeue_subqueue_count_text', $count);
2418         }
2419         else {
2420             $message = $count;
2421         }
2422     }
2423     return $message;
2424 }
2425
2426 function theme_nodequeue_subqueue_empty_text() {
2427     return t('Queue empty');
2428 }
2429
2430 function theme_nodequeue_subqueue_full_text() {
2431     return t('Queue full');
2432 }
2433
2434 function theme_nodequeue_subqueue_count_text($count) {
2435     return t('@count in queue', array('@count' => $count));
2436 }
2437
2438/**
2439 * Substitute the subqueue title into some other string.
2440 *
2441 * This function does NOT check_plain the title! The output MUST be checked
2442 * after this is complete.
2443 */
2444 function nodequeue_title_substitute($text, $queue, $subqueue) {
2445     if (empty($text)) {
2446         return $subqueue->title;
2447     }
2448     $text = str_replace('%subqueue', $subqueue->title, $text);
2449     return $text;
2450 }
2451
2452/**
2453 * Shuffle a queue.
2454 *
2455 * @param $subqueue
2456 *   The subqueue to shuffle. May be a sqid or the loaded object.
2457 */
2458 function nodequeue_subqueue_shuffle($subqueue) {
2459     // Load the queue
2460     if (!is_object($subqueue)) {
2461         $subqueue = nodequeue_load_subqueue($subqueue);
2462     }

```

```

2463
2464 if (empty($subqueue)) {
2465     return;
2466 }
2467
2468 $count = $subqueue->count;
2469 // Swap each item with another randomly picked one.
2470 foreach (range(1, $count) as $i) {
2471     nodequeue_queue_swap($subqueue, $i, rand(1, $count));
2472 }
2473}
2474
2475/**
2476 * @} End of defgroup "nodequeue_api"
2477 */
2478
2479// -----
2480// Hooks to implement the default nodequeue type.
2481
2482/**
2483 * Implementation of hook_nodequeue_info()
2484 */
2485function nodequeue_nodequeue_info() {
2486    return array('nodequeue' => array(
2487        'title' => t('Nodequeue'),
2488        'description' => t('Standard nodequeues have just one subqueue. Nodes put into a queue are added to the back of the queue; when a node is added to a full
2489    ));
2490}
2491
2492/**
2493 * Implementation of hook_nodequeue_form_submit()
2494 */
2495function nodequeue_nodequeue_form_submit(&$queue, $form_state) {
2496    // This will add a single subqueue to our new queue.
2497    if (!isset($queue->qid) && !isset($queue->add_subqueue)) {
2498        // A 0 will set the reference to the sqid of the queue.
2499        $queue->add_subqueue = array(0 => $queue->title);
2500    }
2501}
2502
2503// -----
2504// External queue fetching
2505
2506/**
2507 * in general it's preferable to use Views for this functionality.
2508 */
2509function nodequeue_node_titles($sqid, $title = '', $backward = true, $from = 0, $count = 0) {
2510    $orderby = ($backward ? "DESC" : "ASC");
2511    $sql = db_rewrite_sql("SELECT n.nid, n.title FROM {node} n LEFT JOIN {nodequeue_nodes} nn ON n.nid = nn.nid WHERE nn.sqid = %d AND n.status = 1 ORDER BY nn.
2512    if ($count) {
2513        $result = db_query_range($sql, $sqid, $from, $count);
2514    }
2515    else {
2516        $result = db_query($sql, $sqid);
2517    }
2518    return node_title_list($result, $title);
2519}
2520
2521/**
2522 * Get node_view output from a nodequeue
2523 */
2524function nodequeue_view_nodes($sqid, $backward = TRUE, $teaser = TRUE, $links = TRUE, $from = 0, $count = 0) {
2525    $nodes = nodequeue_load_nodes($sqid, $backward, $from, $count);
2526    foreach ($nodes as $node) {
2527        $output .= node_view($node, $teaser, FALSE, $links);
2528    }
2529    return $output;
2530}
2531
2532/**
2533 * Load an array of node objects belonging to a particular nodequeue.
2534 */
2535function nodequeue_load_nodes($sqid, $backward = FALSE, $from = 0, $count = 5) {
2536    $orderby = ($backward ? "DESC" : "ASC");
2537    $sql = db_rewrite_sql("SELECT n.nid FROM {node} n INNER JOIN {nodequeue_nodes} nn ON n.nid = nn.nid WHERE nn.sqid = %d AND n.status = 1 ORDER BY nn.position
2538    if ($count) {
2539        $result = db_query_range($sql, $sqid, $from, $count);
2540    }
2541    else {
2542        $result = db_query($sql, $sqid);
2543    }
2544
2545    while ($nid = db_fetch_object($result)) {
2546        $nodes[] = node_load($nid->nid);
2547    }
2548
2549    return $nodes;
2550}

```

```

2551
2552/**
2553 * Load the first node of a queue
2554 */
2555function nodequeue_load_front($sqid) {
2556     return array_shift(nodequeue_load_nodes($sqid, FALSE, 0, 1));
2557}
2558
2559/**
2560 * Load the last node of a queue
2561 */
2562function nodequeue_load_back($sqid, $teaser = TRUE, $links = TRUE) {
2563     return array_shift(nodequeue_load_nodes($sqid, TRUE, 0, 1));
2564}
2565
2566/**
2567 * View a random node from a queue
2568 */
2569function nodequeue_view_random_node($sqid, $teaser = TRUE, $links = TRUE) {
2570     $count = db_result(db_query(db_rewrite_sql("SELECT COUNT(n.nid) FROM {node} n INNER JOIN {nodequeue_nodes} nn ON n.nid = nn.nid WHERE nn.sqid = %d AND n.sta
2571     return array_shift(nodequeue_view_nodes($sqid, FALSE, $teaser, $links, rand(0, $count - 1), 1));
2572}
2573
2574/**
2575 * Load a random node object from a queue
2576 */
2577function nodequeue_load_random_node($sqid) {
2578     $count = db_result(db_query(db_rewrite_sql("SELECT COUNT(n.nid) FROM {node} n INNER JOIN {nodequeue_nodes} nn ON n.nid = nn.nid WHERE nn.sqid = %d AND n.sta
2579     return array_shift(nodequeue_load_nodes($sqid, TRUE, rand(0, $count - 1), 1));
2580}
2581
2582/**
2583 * Get the position of a node in a subqueue, or FALSE if not found.
2584 */
2585function nodequeue_subqueue_position($sqid, $nid) {
2586     return db_result(db_query("SELECT position FROM {nodequeue_nodes} WHERE sqid = %d AND nid = %d", $sqid, $nid));
2587}
2588
2589/**
2590 * Get the position of a node in a queue; this queue MUST have only one
2591 * subqueue or the results of this function will be unpredictable.
2592 */
2593function nodequeue_queue_position($qid, $nid) {
2594     $sqid = db_result(db_query_range("SELECT sqid FROM {nodequeue_subqueue} WHERE qid = %d", $qid, 0, 1));
2595     return nodequeue_subqueue_position($sqid, $nid);
2596}
2597
2598// -----
2599// API for modules implementing subqueues.
2600
2601/**
2602 * Send the nodequeue edit form to the owning module for modification.
2603 *
2604 * @param $queue
2605 *     The queue being edited.
2606 * @param $form
2607 *     The form. This may be modified.
2608 */
2609//TODO: Form modifying code - Modify for D6?
2610function nodequeue_api_queue_form($queue, &$form) {
2611     $function = $queue->owner . "_nodequeue_form";
2612     if (function_exists($function)) {
2613         $function($queue, $form);
2614     }
2615}
2616
2617/**
2618 * Validate the nodequeue edit form.
2619 *
2620 * @param $queue
2621 *     The queue being edited.
2622 * @param $form_state
2623 *     The form values that were submitted.
2624 * @param $form
2625 *     The actual form object. This may be modified.
2626 */
2627function nodequeue_api_queue_form_validate($queue, &$form_state, &$form) {
2628     $function = $queue->owner . "_nodequeue_form_validate";
2629     if (function_exists($function)) {
2630         $function($queue, $form_state, $form);
2631     }
2632}
2633
2634/**
2635 * Send the nodequeue edit form to the owning module upon submit.
2636 *
2637 * @param $queue
2638 *     The queue being edited. This may be modified prior to being

```

```

2639 * saved.
2640 * @param $form_state
2641 * The form values that were submitted.
2642 */
2643 function nodequeue_api_queue_form_submit(&$queue, &$form_state) {
2644     $function = $queue->owner . "_nodequeue_form_submit";
2645     if (function_exists($function)) {
2646         $function($queue, $form_state);
2647     }
2648 }
2649
2650 /**
2651 * Send the nodequeue edit form to the owning module after the queue
2652 * has been saved.
2653 *
2654 * @param &$queue
2655 * The queue being edited. This may be modified prior to being
2656 * saved.
2657 * @param $form_state
2658 * The form values that were submitted.
2659 */
2660 function nodequeue_api_queue_form_submit_finish($queue, &$form_state) {
2661     $function = $queue->owner . "_nodequeue_form_submit_finish";
2662     if (function_exists($function)) {
2663         $function($queue, $form_state);
2664     }
2665 }
2666
2667 /**
2668 * Fetch a list of subqueues that are valid for this node from
2669 * the owning module.
2670 *
2671 * @param $queue
2672 * The queue being edited.
2673 * @param $node
2674 * The loaded node object being checked.
2675 *
2676 * @return
2677 * An array of subqueues. This will be keyed by $sqid.
2678 */
2679 function nodequeue_api_subqueues(&$queue, $node) {
2680     $function = $queue->owner . "_nodequeue_subqueues";
2681     // This will return an array of references.
2682     if (function_exists($function)) {
2683         return $function($queue, $node);
2684     }
2685     else {
2686         return $queue->qid;
2687     }
2688 }
2689
2690 /**
2691 * Fetch a list of nodes available to a given subqueue
2692 * for autocomplete.
2693 *
2694 * @param $queue
2695 * The queue that owns the subqueue
2696 * @param $subqueue
2697 * The subqueue
2698 * @param $string
2699 * The string being matched.
2700 *
2701 * @return
2702 * An keyed array $nid => $title
2703 */
2704 function nodequeue_api_autocomplete($queue, $subqueue, $string) {
2705     $matches = array();
2706     if (empty($string)) {
2707         return $matches;
2708     }
2709
2710     $where = "n.type IN ('" . db_placeholders($queue->types, 'varchar') . "')";
2711     $where_args = $queue->types;
2712
2713     // Run a match to see if they're specifying by nid.
2714     $preg_matches = array();
2715     $match = preg_match('/\[nid: (\d+)\]/', $string, $preg_matches);
2716     if (!$match) {
2717         $match = preg_match('/^nid: (\d+)/', $string, $preg_matches);
2718     }
2719
2720     if ($match) {
2721         // If it found a nid via specification, reduce our resultset to just that nid.
2722         $where .= " AND n.nid = %d";
2723         array_push($where_args, $preg_matches[1]);
2724     }
2725     else {

```

```

2727 // Build the constant parts of the query.
2728 $where .= " AND LOWER(n.title) LIKE LOWER('%s%')";
2729 array_push($where_args, $string);
2730 }
2731
2732 // Call to the API.
2733 $function = $queue->owner . "_nodequeue_autocomplete";
2734 if (function_exists($function)) {
2735     return $function($queue, $subqueue, $string, $where, $where_args);
2736 }
2737 else {
2738     $result = db_query_range(db_rewrite_sql("SELECT n.nid, n.title FROM {node} n WHERE $where"), $where_args, 0, 10);
2739     while ($node = db_fetch_object($result)) {
2740         $matches[$node->nid] = check_plain($node->title) . " [nid: $node->nid]";
2741     }
2742 }
2743
2744 return $matches;
2745 }
2746
2747 /**
2748  * Collect info about all of the possible nodequeue types from owning
2749  * modules.
2750  */
2751 function nodequeue_api_info() {
2752     return module_invoke_all('nodequeue_info');
2753 }
2754
2755 function nodequeue_api_queue_access($queue, $account = NULL) {
2756     if (!$account) {
2757         global $user;
2758         $account = $user;
2759     }
2760
2761     if ($queue->owner != 'nodequeue') { // Avoids an infinite loop.
2762         $function = $queue->owner . '_queue_access';
2763         if (function_exists($function)) {
2764             $access = $function($queue, $account);
2765         }
2766     }
2767
2768     if (!isset($access)) {
2769         $access = TRUE;
2770     }
2771     return $access;
2772 }
2773
2774 /**
2775  * Allows the owning module of a subqueue to restrict access to viewing and
2776  * manipulating the queue.
2777  */
2778 function nodequeue_api_subqueue_access($subqueue, $account = NULL, $queue = NULL) {
2779     if (!$account) {
2780         global $user;
2781         $account = $user;
2782     }
2783
2784     if (!$queue) {
2785         $queue = nodequeue_load($subqueue->qid);
2786     }
2787
2788     $function = $queue->owner . '_subqueue_access';
2789     if (function_exists($function)) {
2790         $access = $function($subqueue, $account, $queue);
2791     }
2792
2793     if (!isset($access)) {
2794         $access = TRUE;
2795     }
2796
2797     return $access;
2798 }
2799 /**
2800  * Form builder for the nodequeue settings tab.
2801  */
2802 function nodequeue_admin_settings() {
2803     $form = array();
2804     $form['nodequeue_use_tab'] = array(
2805         '#type' => 'checkbox',
2806         '#title' => t('Create a menu tab for each node that could belong to any queues'),
2807         '#default_value' => variable_get('nodequeue_use_tab', 1),
2808     );
2809     $form['nodequeue_tab_display_max'] = array(
2810         '#type' => 'checkbox',
2811         '#title' => t('Include a column on the nodequeue tab for the maximum number of nodes in each queue'),
2812         '#default_value' => variable_get('nodequeue_tab_display_max', 1),
2813     );
2814     $form['nodequeue_tab_name'] = array(

```

```

2815     '#type' => 'textfield',
2816     '#title' => t('Nodequeue tab label'),
2817     '#default_value' => variable_get('nodequeue_tab_name', t('Nodequeue')),
2818     '#description' => t('If nodes will have a menu tab for manipulating related nodequeues, what should that tab be called?'),
2819 );
2820 return system_settings_form($form);
2821 }
2822
2823 /**
2824  * Generate a query string to use on nodequeue's private links.
2825  *
2826  * @param $seed
2827  *   The seed to use when generating a token. If NULL no token will
2828  *   be generated.
2829  * @param $destination
2830  *   The destination to use. If FALSE one won't be used; if TRUE
2831  *   one will be generated from drupal_get_destination().
2832  * @param $query
2833  *   An array of additional items to add to the query.
2834  *
2835  * @return
2836  *   The query string suitable for use in the l() function.
2837  */
2838 function nodequeue_get_query_string($seed, $destination = FALSE, $query = array()) {
2839   if ($dest = drupal_get_destination()) {
2840     $query[] = $dest;
2841   }
2842
2843   if (isset($seed)) {
2844     $query[] = nodequeue_get_token($seed);
2845   }
2846
2847   return implode('&', $query);
2848 }
2849
2850 /**
2851  * Get a private token used to protect nodequeue's links from spoofing.
2852  */
2853 function nodequeue_get_token($nid) {
2854   return 'token=', drupal_get_token($nid);
2855 }
2856
2857 /**
2858  * Check to see if the token generated from seed matches.
2859  */
2860 function nodequeue_check_token($seed) {
2861   return drupal_get_token($seed) == $_GET['token'];
2862 }
2863

```

Legend

Missed

lines code that **were not** excersized during program execution.

Covered

lines code **were** excersized during program execution.

Comment/non executable

Comment or non-executable line of code.

Dead

lines of code that according to xdebug could not be executed. This is counted as coverage code because in almost all cases it is code that runnable.