Drupal Reference Manual

4.6.2

Generated by Doxygen 1.3.5

Sun Jul 24 00:02:15 2005

Contents

Chapter 1

Drupal Module Index

1.1 Drupal Modules

Here is a list of all modules:

Chapter 2

Drupal File Index

2.1 Drupal File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Drupal Module Documentation

3.1 Input validation

Functions

- valid_email_address ($mail)
- valid_url ($url, $absolute=FALSE)
- valid_input_data ($data)

3.1.1 Detailed Description

Functions to validate user input.

3.1.2 Function Documentation

3.1.2.1 valid_email_address ($ *mail*)

Verify the syntax of the given e-mail address.

Empty e-mail addresses are allowed. See RFC 2822 for details.

Parameters:

$mail A string containing an email address.

Returns:

TRUE if the address is in a valid format.

Definition at line 573 of file common.inc.

```
573                                     {
574   $user = '[a-zA-Z0-9_\-\.\+\^!#\$%&*+\/\=\?\`\|\{\}~\']+';
575   $domain = '(?:(?:[a-zA-Z0-9]|[a-zA-Z0-9][a-zA-Z0-9\-]*[a-zA-Z0-9])\.?)+';
576   $ipv4 = '[0-9]{1,3}(\.[0-9]{1,3}){3}';
577   $ipv6 = '[0-9a-fA-F]{1,4}(\:[0-9a-fA-F]{1,4}){7}';
578
579   return preg_match("/^$user@($domain|(\[($ipv4|$ipv6)\]))$/", $mail);
580 }
```

3.1.2.2 valid_input_data ($ *data*)

Validate data input by a user.

Ensures that user data cannot be used to perform attacks on the site.

Parameters:

$data The input to check.

Returns:

TRUE if the input data is acceptable.

Definition at line 612 of file common.inc.

References theme(), and watchdog().

Referenced by file_save_data(), and file_save_upload().

```
612                                     {
613   if (is_array($data) || is_object($data)) {
614     // Form data can contain a number of nested arrays.
615     foreach ($data as $key => $value) {
616       if (!valid_input_data($key) || !valid_input_data($value)) {
617         return FALSE;
618       }
619     }
620   }
621   else if (isset($data)) {
622     // Detect dangerous input data.
623
624     // Decode all normal character entities.
625     $data = decode_entities($data, array('<', '&', '"'));
626
627     // Check strings:
628     $match  = preg_match('/\Wjavascript\s*:/i', $data);
629     $match += preg_match('/\Wexpression\s*\(/i', $data);
630     $match += preg_match('/\Walert\s*\(/i', $data);
631
632     // Check attributes:
633     $match += preg_match("/\W(dynsrc|datasrc|data|lowsrc|on[a-z]+)\s*=[^>]+?>/i", $data);
634
635     // Check tags:
636     $match += preg_match("/<\s*(applet|script|object|style|embed|form|blink|meta|html|frame|iframe|lay
637
638     if ($match) {
639       watchdog('security', t('Terminated request because of suspicious input data: %data.', array('%da
640       return FALSE;
641     }
642   }
643
644   return TRUE;
645 }
```

3.1.2.3 valid_url ($ *url*, $ *absolute* = FALSE)

Verify the syntax of the given URL.

Parameters:

$url The URL to verify.

$absolute Whether the URL is absolute (beginning with a scheme such as "http:").

Returns:

TRUE if the URL is in a valid format.

Definition at line 592 of file common.inc.

References url().

```
592                                             {
593   $allowed_characters = '[a-z0-9\/:_\-_\.\?\$,~=#&%\+]';
594   if ($absolute) {
595     return preg_match("/^(http|https|ftp):\/\/". $allowed_characters ."+$/i", $url);
596   }
597   else {
598     return preg_match("/^". $allowed_characters ."+$/i", $url);
599   }
600 }
```

3.2 Formatting

Functions

- format_rss_channel ($title, $link, $description, $items, $language= 'en', $args=array())
- format_rss_item ($title, $link, $description, $args=array())
- format_plural ($count, $singular, $plural)
- format_size ($size)
- format_interval ($timestamp, $granularity=2)
- format_date ($timestamp, $type= 'medium', $format= '', $timezone=NULL)
- format_name ($object)

3.2.1 Detailed Description

Functions to format numbers, strings, dates, etc.

3.2.2 Function Documentation

3.2.2.1 format_date ($ *timestamp*, $ *type* = 'medium', $ *format* = '', $ *timezone* = NULL)

Format a date with the given configured format or a custom format string.

Drupal allows administrators to select formatting strings for 'small', 'medium' and 'large' date formats. This function can handle these formats, as well as any custom format.

Parameters:

$timestamp The exact date to format, as a UNIX timestamp.

$type The format to use. Can be "small", "medium" or "large" for the preconfigured date formats. If "custom" is specified, then $format is required as well.

$format A PHP date format string as required by date(). A backslash should be used before a character to avoid interpreting the character as part of a date format.

$timezone Time zone offset in seconds; if omitted, the user's time zone is used.

Returns:

A translated date string in the requested format.

Definition at line 852 of file common.inc.

Referenced by theme_aggregator_page_item().

```
852                                                                                 {
853   if ($timezone === NULL) {
854     global $user;
855     if (variable_get('configurable_timezones', 1) && $user->uid && strlen($user->timezone)) {
856       $timezone = $user->timezone;
857     }
```

```
858     else {
859       $timezone = variable_get('date_default_timezone', 0);
860     }
861   }
862
863   $timestamp += $timezone;
864
865   switch ($type) {
866     case 'small':
867       $format = variable_get('date_format_short', 'm/d/Y - H:i');
868       break;
869     case 'large':
870       $format = variable_get('date_format_long', 'l, F j, Y - H:i');
871       break;
872     case 'custom':
873       // No change to format
874       break;
875     case 'medium':
876     default:
877       $format = variable_get('date_format_medium', 'D, m/d/Y - H:i');
878   }
879
880   $max = strlen($format);
881   $date = '';
882   for ($i = 0; $i < $max; $i++) {
883     $c = $format{$i};
884     if (strpos('AaDFlM', $c) !== false) {
885       $date .= t(gmdate($c, $timestamp));
886     }
887     else if (strpos('BdgGhHiIjLmnsStTUwWYyz', $c) !== false) {
888       $date .= gmdate($c, $timestamp);
889     }
890     else if ($c == 'r') {
891       $date .= format_date($timestamp - $timezone, 'custom', 'D, d M Y H:i:s O', $timezone);
892     }
893     else if ($c == 'O') {
894       $date .= sprintf('%s%02d%02d', ($timezone < 0 ? '-' : '+'), abs($timezone / 3600), abs($timezone
895     }
896     else if ($c == 'Z') {
897       $date .= $timezone;
898     }
899     else if ($c == '\\') {
900       $date .= $format[++$i];
901     }
902     else {
903       $date .= $c;
904     }
905   }
906
907   return $date;
908 }
```

3.2.2.2 format_interval ($ *timestamp*, $ *granularity* = 2)

Format a time interval with the requested granularity.

Parameters:

$timestamp The length of the interval in seconds.

$granularity How many different units to display in the string.

Returns:

A translated string representation of the interval.

Definition at line 813 of file common.inc.

Referenced by theme_aggregator_feed(), and theme_aggregator_summary_item().

```
813                                                     {
814   $units = array('1 year|%count years' => 31536000, '1 week|%count weeks' => 604800, '1 day|%count day
815   $output = '';
816   foreach ($units as $key => $value) {
817     $key = explode('|', $key);
818     if ($timestamp >= $value) {
819       $output .= ($output ? ' ' : '') . format_plural(floor($timestamp / $value), $key[0], $key[1]);
820       $timestamp %= $value;
821       $granularity--;
822     }
823
824     if ($granularity == 0) {
825       break;
826     }
827   }
828   return $output ? $output : t('0 sec');
829 }
```

3.2.2.3 format_name ($ *object*)

Format a username.

Parameters:

$object The user object to format, usually returned from user_load().

Returns:

A string containing an HTML link to the user's page if the passed object suggests that this is a site user. Otherwise, only the username is returned.

Definition at line 919 of file common.inc.

References l(), and variable_get().

Referenced by theme_node().

```
919                                {
920
921   if ($object->uid && $object->name) {
922     // Shorten the name when it is too long or it will break many tables.
923     if (strlen($object->name) > 20) {
924       $name = truncate_utf8($object->name, 15) .'...';
925     }
926     else {
927       $name = $object->name;
928     }
929
930     if (user_access('access user profiles')) {
931       $output = l($name, 'user/'. $object->uid, array('title' => t('View user profile.')));
932     }
933     else {
934       $output = $name;
935     }
936   }
937   else if ($object->name) {
938     // Sometimes modules display content composed by people who are
```

```
939     // not registered members of the site (e.g. mailing list or news
940     // aggregator modules). This clause enables modules to display
941     // the true author of the content.
942     if ($object->homepage) {
943       $output = '<a href="'. $object->homepage .'">'. $object->name .'</a>';
944     }
945     else {
946       $output = $object->name;
947     }
948
949     $output .= ' ('. t('not verified') .')';
950   }
951   else {
952     $output = variable_get('anonymous', 'Anonymous');
953   }
954
955   return $output;
956 }
```

3.2.2.4 format_plural ($ *count*, $ *singular*, $ *plural*)

Format a string containing a count of items.

This function ensures that the string is pluralized correctly. Since t() is called by this function, make sure not to pass already-localized strings to it.

Parameters:

$count The item count to display.

$singular The string for the singular case. Please make sure it is clear this is singular, to ease translation (e.g. use "1 new comment" instead of "1 new").

$plural The string for the plural case. Please make sure it is clear this is plural, to ease translation. Use count in place of the item count, as in "%count new comments".

Returns:

A translated string.

Definition at line 762 of file common.inc.

```
762                                                 {
763   if ($count == 1) return t($singular, array("%count" => $count));
764
765   // get the plural index through the gettext formula
766   $index = (function_exists('locale')) ? locale_get_plural($count) : -1;
767   if ($index < 0) { // backward compatibility
768     return t($plural, array("%count" => $count));
769   }
770   else {
771     switch ($index) {
772       case "0":
773         return t($singular, array("%count" => $count));
774       case "1":
775         return t($plural, array("%count" => $count));
776       default:
777         return t(strtr($plural, array("%count" => '%count['. $index .']')), array('%count['. $index .'
778     }
779   }
780 }
```

3.2.2.5 format_rss_channel ($ *title*, $ *link*, $ *description*, $ *items*, $ *language* = 'en', $ *args* = array())

Formats an RSS channel.

Arbitrary elements may be added using the $args associative array.

Definition at line 692 of file common.inc.

```
692                                                                                                                 {
693   // arbitrary elements may be added using the $args associative array
694
695   $output = "<channel>\n";
696   $output .= ' <title>'. check_plain($title) ."</title>\n";
697   $output .= ' <link>'. check_url($link) ."</link>\n";
698   $output .= ' <description>'. check_plain($description) ."</description>\n";
699   $output .= ' <language>'. check_plain($language) ."</language>\n";
700   foreach ($args as $key => $value) {
701     $output .= ' <'. $key .'>'. check_plain($value) ."</$key>\n";
702   }
703   $output .= $items;
704   $output .= "</channel>\n";
705
706   return $output;
707 }
```

3.2.2.6 format_rss_item ($ *title*, $ *link*, $ *description*, $ *args* = array())

Format a single RSS item.

Arbitrary elements may be added using the $args associative array.

Definition at line 714 of file common.inc.

```
714                                                                                                                 {
715   $output = "<item>\n";
716   $output .= ' <title>'. check_plain($title) ."</title>\n";
717   $output .= ' <link>'. check_url($link) ."</link>\n";
718   $output .= ' <description>'. check_plain($description) ."</description>\n";
719   foreach ($args as $key => $value) {
720     if (is_array($value)) {
721       if ($value['key']) {
722         $output .= ' <'. $value['key'];
723         if (is_array($value['attributes'])) {
724           $output .= drupal_attributes($value['attributes']);
725         }
726
727         if ($value['value']) {
728           $output .= '>'. $value['value'] .'</'. $value['key'] .">\n";
729         }
730         else {
731           $output .= " />\n";
732         }
733       }
734     }
735     else {
736       $output .= ' <'. $key .'>'. check_plain($value) ."</$key>\n";
737     }
738   }
739   $output .= "</item>\n";
740
741   return $output;
742 }
```

3.2.2.7 format_size ($ *size*)

Generate a string representation for the given byte count.

Parameters:

$size The size in bytes.

Returns:

A translated string representation of the size.

Definition at line 790 of file common.inc.

```
790                             {
791   $suffix = t('bytes');
792   if ($size > 1024) {
793     $size = round($size / 1024, 2);
794     $suffix = t('KB');
795   }
796   if ($size > 1024) {
797     $size = round($size / 1024, 2);
798     $suffix = t('MB');
799   }
800   return t('%size %suffix', array('%size' => $size, '%suffix' => $suffix));
801 }
```

3.3 Form generation

Functions

- form ($form, $method= 'post', $action=NULL, $attributes=NULL)
- form_set_error ($name, $message)
- form_get_errors ()
- _form_get_error ($name)
- **_form_get_class** ($name, $required, $error)
- form_item ($title, $value, $description=NULL, $id=NULL, $required=FALSE, $error=FALSE)
- form_group ($legend, $group, $description=NULL)
- form_radio ($title, $name, $value=1, $checked=FALSE, $description=NULL, $attributes=NULL, $required=FALSE)
- form_radios ($title, $name, $value, $options, $description=NULL, $required=FALSE, $attributes=NULL)
- form_checkbox ($title, $name, $value=1, $checked=FALSE, $description=NULL, $attributes=NULL, $required=FALSE)
- form_checkboxes ($title, $name, $values, $options, $description=NULL, $attributes=NULL, $required=FALSE)
- form_textfield ($title, $name, $value, $size, $maxlength, $description=NULL, $attributes=NULL, $required=FALSE)
- form_password ($title, $name, $value, $size, $maxlength, $description=NULL, $attributes=NULL, $required=FALSE)
- form_textarea ($title, $name, $value, $cols, $rows, $description=NULL, $attributes=NULL, $required=FALSE)
- form_select ($title, $name, $value, $options, $description=NULL, $extra=0, $multiple=FALSE, $required=FALSE)
- form_file ($title, $name, $size, $description=NULL, $required=FALSE)
- form_hidden ($name, $value)
- form_button ($value, $name= 'op', $type= 'submit', $attributes=NULL)
- form_submit ($value, $name= 'op', $attributes=NULL)
- form_weight ($title=NULL, $name= 'weight', $value=0, $delta=10, $description=NULL, $extra=0)

3.3.1 Detailed Description

Functions to enable output of HTML forms and form elements.

Drupal uses these functions to achieve consistency in its form presentation, while at the same time simplifying code and reducing the amount of HTML that must be explicitly generated by modules.

3.3.2 Function Documentation

3.3.2.1 _form_get_error ($ *name*)

Return the error message filed against the form with the specified name.

Definition at line 1012 of file common.inc.

References form().

Referenced by form_checkbox(), form_checkboxes(), form_file(), form_password(), form_radio(), form_radios(), form_select(), form_textarea(), and form_textfield().

```
1012                                {
1013   if (array_key_exists('form', $GLOBALS)) {
1014     return $GLOBALS['form'][$name];
1015   }
1016 }
```

3.3.2.2 form ($ *form*, $ *method* = 'post', $ *action* = NULL, $ *attributes* = NULL)

Generate a form from a set of form elements.

Parameters:

$form An HTML string containing one or more form elements.

$method The query method to use ("post" or "get").

$action The URL to send the form contents to, if not the current page.

$attributes An associative array of attributes to add to the form tag.

Returns:

An HTML string with the contents of $form wrapped in a form tag.

Definition at line 985 of file common.inc.

References drupal_attributes(), form(), and request_uri().

Referenced by _form_get_error(), _locale_admin_export_screen(), _locale_admin_import_screen(), _locale_admin_manage_add_screen(), _locale_admin_manage_screen(), _locale_string_seek_form(), form(), form_get_errors(), form_set_error(), search_form(), and theme_confirm().

```
985                                                                                                 {
986   if (!$action) {
987     $action = request_uri();
988   }
989   return '<form action="'. check_url($action) .'" method="'. $method .'"'. drupal_attributes($attribut
990 }
```

3.3.2.3 form_button ($ *value*, $ *name* = 'op', $ *type* = 'submit', $ *attributes* = NULL)

Format an action button.

Parameters:

$value Both the label for the button, and the value passed to the target page when this button is clicked.

$name The internal name used to refer to the button.

$type What type to pass to the HTML input tag.

$attributes An associative array of HTML attributes to add to the form item.

Returns:
A themed HTML string representing the button.

Definition at line 1379 of file common.inc.

References drupal_attributes().

Referenced by form_submit().

```
1379                                                                                {
1380   return '<input type="'. $type .'" class="form-'. $type .'" name="'. $name .'" value="'. check_plain
1381 }
```

3.3.2.4 form_checkbox ($ *title*, $ *name*, $ *value* = 1, $ *checked* = FALSE, $ *description* = NULL, $ *attributes* = NULL, $ *required* = FALSE)

Format a checkbox.

Parameters:
$title The label for the checkbox.

$name The internal name used to refer to the button.

$value The value that the form element takes on when selected.

$checked Whether the button will be initially selected when the page is rendered.

$description Explanatory text to display after the form item.

$attributes An associative array of HTML attributes to add to the button.

$required Whether the user must check this box before submitting the form.

Returns:
A themed HTML string representing the checkbox.

Definition at line 1139 of file common.inc.

References _form_get_error(), drupal_attributes(), form_hidden(), and theme().

Referenced by _locale_admin_manage_screen().

```
1139
1140   $element = '<input type="checkbox" class="'. _form_get_class('form-checkbox', $required, _form_get_
1141   if (!is_null($title)) {
1142     $element = '<label class="option">'. $element .' '. $title .'</label>';
1143   }
1144   // Note: because unchecked boxes are not included in the POST data, we include
1145   // a form_hidden() which will be overwritten for a checked box.
1146   return form_hidden($name, 0) . theme('form_element', NULL, $element, $description, $name, $required
1147 }
```

3.3.2.5 form_checkboxes ($ *title*, $ *name*, $ *values*, $ *options*, $ *description* = NULL, $ *attributes* = NULL, $ *required* = FALSE)

Format a set of checkboxes.

Parameters:

$title The label for the checkboxes as a group.

$name The internal name used to refer to the buttons.

$values A linear array of keys of the initially checked boxes.

$options An associative array of buttons to display. The keys in this array are button values, while the values are the labels to display for each button.

$description Explanatory text to display after the form item.

$attributes An associative array of HTML attributes to add to each button.

$required Whether the user must check a box before submitting the form.

Returns:

A themed HTML string representing the radio button set.

Definition at line 1170 of file common.inc.

References _form_get_error(), drupal_attributes(), form_hidden(), and theme().

```
1170
1171   if (count($options) > 0) {
1172     if (!isset($values) || $values == 0) {
1173       $values = array();
1174     }
1175     $choices = '';
1176     foreach ($options as $key => $choice) {
1177       $choices .= '<label class="option"><input type="checkbox" class="form-checkbox" name="edit['. $
1178     }
1179     // Note: because unchecked boxes are not included in the POST data, we
1180     // include a form_hidden() which will be overwritten as soon as there is at
1181     // least one checked box.
1182     return form_hidden($name, 0) . theme('form_element', $title, $choices, $description, NULL, $requi
1183   }
1184 }
```

3.3.2.6 form_file ($ *title*, $ *name*, $ *size*, $ *description* = NULL, $ *required* = FALSE)

Format a file upload field.

Parameters:

$title The label for the file upload field.

$name The internal name used to refer to the field.

$size A measure of the visible size of the field (passed directly to HTML).

$description Explanatory text to display after the form item.

$required Whether the user must upload a file to the field.

Returns:

A themed HTML string representing the field.

For assistance with handling the uploaded file correctly, see the API provided by file.inc.

Definition at line 1342 of file common.inc.

References _form_get_error(), and theme().

```
1342                                                                                {
1343   return theme('form_element', $title, '<input type="file" class="'. _form_get_class('form-file', $re
1344 }
```

3.3.2.7 form_get_errors ()

Return an associative array of all errors.

Definition at line 1003 of file common.inc.

References form().

```
1003                             {
1004   if (array_key_exists('form', $GLOBALS)) {
1005     return $GLOBALS['form'];
1006   }
1007 }
```

3.3.2.8 form_group ($ *legend*, $ *group*, $ *description* = NULL)

Format a group of form items.

Parameters:

$legend The label for the form item group.

$group The form items within the group, as an HTML string.

$description Explanatory text to display after the form item group.

Returns:

A themed HTML string representing the form item group.

Definition at line 1056 of file common.inc.

Referenced by _locale_string_seek_form().

```
1056                                                        {
1057   return '<fieldset>' . ($legend ? '<legend>'. $legend .'</legend>' : '') . $group . ($description ?
1058 }
```

3.3.2.9 form_hidden ($ *name*, $ *value*)

Store data in a hidden form field.

Parameters:

$name The internal name used to refer to the field.

$value The stored data.

Returns:

A themed HTML string representing the hidden field.

This function can be useful in retaining information between page requests, but be sure to validate the data on the receiving page as it is possible for an attacker to change the value before it is submitted.

Definition at line 1360 of file common.inc.

Referenced by form_checkbox(), form_checkboxes(), and theme_confirm().

```
1360                                    {
1361   return '<input type="hidden" name="edit['. $name .']" value="'. check_plain($value) ."\" />\n";
1362 }
```

3.3.2.10 form_item ($ *title*, $ *value*, $ *description* = NULL, $ *id* = NULL, $ *required* = FALSE, $ *error* = FALSE)

Format a general form item.

Parameters:

$title The label for the form item.

$value The contents of the form item.

$description Explanatory text to display after the form item.

$id A unique identifier for the form item.

$required Whether the user must fill in this form element before submitting the form.

$error An error message to display alongside the form element.

Returns:

A themed HTML string representing the form item.

Definition at line 1040 of file common.inc.

References theme().

Referenced by file_check_directory().

```
1040
1041   return theme('form_element', $title, $value, $description, $id, $required, $error);
1042 }
```

3.3.2.11 form_password ($ *title*, $ *name*, $ *value*, $ *size*, $ *maxlength*, $ *description* = NULL, $ *attributes* = NULL, $ *required* = FALSE)

Format a single-line text field that does not display its contents visibly.

Parameters:

$title The label for the text field.

$name The internal name used to refer to the field.

$value The initial value for the field at page load time.

$size A measure of the visible size of the field (passed directly to HTML).

$maxlength The maximum number of characters that may be entered in the field.

$description Explanatory text to display after the form item.

$attributes An associative array of HTML attributes to add to the form item.

$required Whether the user must enter some text in the field.

Returns:

A themed HTML string representing the field.

Definition at line 1235 of file common.inc.

References _form_get_error(), drupal_attributes(), and theme().

```
1235
1236   $size = $size ? ' size="'. $size .'"' : '';
1237   return theme('form_element', $title, '<input type="password" class="'. _form_get_class('form-passwo
1238 }
```

3.3.2.12 form_radio ($ *title*, $ *name*, $ *value* = 1, $ *checked* = FALSE, $ *description* = NULL, $ *attributes* = NULL, $ *required* = FALSE)

Format a radio button.

Parameters:

$title The label for the radio button.

$name The internal name used to refer to the button.

$value The value that the form element takes on when selected.

$checked Whether the button will be initially selected when the page is rendered.

$description Explanatory text to display after the form item.

$attributes An associative array of HTML attributes to add to the button.

$required Whether the user must select this radio button before submitting the form.

Returns:

A themed HTML string representing the radio button.

Definition at line 1080 of file common.inc.

References _form_get_error(), drupal_attributes(), and theme().

Referenced by _locale_admin_manage_screen().

```
1080
1081   $element = '<input type="radio" class="'. _form_get_class('form-radio', $required, _form_get_error(
1082   if (!is_null($title)) {
1083     $element = '<label class="option">'. $element .' '. $title .'</label>';
1084   }
1085   return theme('form_element', NULL, $element, $description, $name, $required, _form_get_error($name)
1086 }
```

3.3.2.13 form_radios ($ *title*, $ *name*, $ *value*, $ *options*, $ *description* = NULL, $ *required* = FALSE, $ *attributes* = NULL)

Format a set of radio buttons.

Parameters:

$title The label for the radio buttons as a group.

$name The internal name used to refer to the buttons.

$value The currently selected radio button's key.

$options An associative array of buttons to display. The keys in this array are button values, while the values are the labels to display for each button.

$description Explanatory text to display after the form item.

$required Whether the user must select a radio button before submitting the form.

$attributes An associative array of HTML attributes to add to each button.

Returns:

A themed HTML string representing the radio button set.

Definition at line 1109 of file common.inc.

References _form_get_error(), drupal_attributes(), and theme().

Referenced by _locale_admin_import_screen(), and _locale_string_seek_form().

```
1109
1110   if (count($options) > 0) {
1111     $choices = '';
1112     foreach ($options as $key => $choice) {
1113       $choices .= '<label class="option"><input type="radio" class="form-radio" name="edit['. $name .
1114     }
1115     return theme('form_element', $title, $choices, $description, NULL, $required, _form_get_error($na
1116   }
1117 }
```

3.3.2.14 form_select ($ *title*, $ *name*, $ *value*, $ *options*, $ *description* = NULL, $ *extra* = 0, $ *multiple* = FALSE, $ *required* = FALSE)

Format a dropdown menu or scrolling selection box.

Parameters:

$title The label for the form element.

$name The internal name used to refer to the form element.

$value The key of the currently selected item, or a linear array of keys of all the currently selected items if multiple selections are allowed.

$options An associative array of buttons to display. The keys in this array are button values, while the values are the labels to display for each button.

$description Explanatory text to display after the form item.

$extra Additional HTML to inject into the select element tag.

$multiple Whether the user may select more than one item.

$required Whether the user must select a value before submitting the form.

Returns:
A themed HTML string representing the form element.

It is possible to group options together; to do this, change the format of $options to an associative array in which the keys are group labels, and the values are associative arrays in the normal $options format.

Definition at line 1306 of file common.inc.

References _form_get_error(), check_plain(), and theme().

Referenced by _locale_admin_export_screen(), _locale_admin_import_screen(), _locale_admin_-manage_add_screen(), and form_weight().

```
1306
1307   $select = '';
1308   foreach ($options as $key => $choice) {
1309     if (is_array($choice)) {
1310       $select .= '<optgroup label="'. $key .'">';
1311       foreach ($choice as $key => $choice) {
1312         $select .= '<option value="'. $key .'"'. (is_array($value) ? (in_array($key, $value) ? ' sele
1313       }
1314       $select .= '</optgroup>';
1315     }
1316     else {
1317       $select .= '<option value="'. $key .'"'. (is_array($value) ? (in_array($key, $value) ? ' select
1318     }
1319   }
1320   return theme('form_element', $title, '<select name="edit['. $name .']'. ($multiple ? '[]' : '') .'"
1321 }
```

3.3.2.15 form_set_error ($ *name*, $ *message*)

File an error against the form element with the specified name.

Definition at line 995 of file common.inc.

References drupal_set_message(), and form().

Referenced by file_check_directory().

```
995                                          {
996   $GLOBALS['form'][$name] = $message;
997   drupal_set_message($message, 'error');
998 }
```

3.3.2.16 form_submit ($ *value*, $ *name* = 'op', $ *attributes* = NULL)

Format a form submit button.

Parameters:
$value Both the label for the button, and the value passed to the target page when this button is clicked.

$name The internal name used to refer to the button.

$attributes An associative array of HTML attributes to add to the form item.

Returns:
A themed HTML string representing the button.

Definition at line 1396 of file common.inc.

References form_button().

Referenced by _locale_admin_export_screen(), _locale_admin_manage_add_screen(), _locale_admin_-manage_screen(), _locale_string_seek_form(), and theme_confirm().

```
1396                                                                    {
1397   return form_button($value, $name, 'submit', $attributes);
1398 }
```

3.3.2.17 form_textarea ($ *title*, $ *name*, $ *value*, $ *cols*, $ *rows*, $ *description* = NULL, $ *attributes* = NULL, $ *required* = FALSE)

Format a multiple-line text field.

Parameters:
$title The label for the text field.
$name The internal name used to refer to the field.
$value The initial value for the field at page load time.
$cols The width of the field, in columns of text.
$rows The height of the field, in rows of text.
$description Explanatory text to display after the form item.
$attributes An associative array of HTML attributes to add to the form item.
$required Whether the user must enter some text in the field.

Returns:
A themed HTML string representing the field.

Definition at line 1262 of file common.inc.

References _form_get_error(), check_plain(), drupal_attributes(), module_hook(), module_invoke(), module_list(), and theme().

```
1262
1263   $cols = $cols ? ' cols="'. $cols .'"' : '';
1264   $pre = '';
1265   $post = '';
1266
1267   // optionally plug in a WYSIWYG editor
1268   foreach (module_list() as $module_name) {
1269     if (module_hook($module_name, 'textarea')) {
1270       $pre  .= module_invoke($module_name, 'textarea', 'pre', $name);
1271       $post .= module_invoke($module_name, 'textarea', 'post', $name);
1272     }
1273   }
1274
1275   return theme('form_element', $title, $pre .'<textarea wrap="virtual"'. $cols .' rows="'. $rows .'"
1276 }
```

3.3.2.18 form_textfield ($ *title*, $ *name*, $ *value*, $ *size*, $ *maxlength*, $ *description* = NULL, $ *attributes* = NULL, $ *required* = FALSE)

Format a single-line text field.

Parameters:

$title The label for the text field.

$name The internal name used to refer to the field.

$value The initial value for the field at page load time.

$size A measure of the visible size of the field (passed directly to HTML).

$maxlength The maximum number of characters that may be entered in the field.

$description Explanatory text to display after the form item.

$attributes An associative array of HTML attributes to add to the form item.

$required Whether the user must enter some text in the field.

Returns:

A themed HTML string representing the field.

Definition at line 1208 of file common.inc.

References _form_get_error(), drupal_attributes(), and theme().

Referenced by _locale_admin_manage_add_screen(), _locale_admin_manage_screen(), and _locale_-string_seek_form().

```
1208
1209   $size = $size ? ' size="'. $size .'"' : '';
1210   return theme('form_element', $title, '<input type="text" maxlength="'. $maxlength .'" class="'. _fo
1211 }
```

3.3.2.19 form_weight ($ *title* = NULL, $ *name* = 'weight', $ *value* = 0, $ *delta* = 10, $ *description* = NULL, $ *extra* = 0)

Format a weight selection menu.

Parameters:

$title The label for the form element.

$name The internal name used to refer to the form element.

$value The selected weight value at page load time.

$delta The largest in absolute value the weight can be. For example, if set to 10, weights could range from -10 to 10 inclusive.

$description Explanatory text to display after the form item.

$extra Additional HTML to inject into the select element tag.

Returns:

A themed HTML string representing the form element.

Definition at line 1419 of file common.inc.

References form_select().

```
1419
1420   for ($n = (-1 * $delta); $n <= $delta; $n++) {
1421     $weights[$n] = $n;
1422   }
1423
1424   return form_select($title, $name, $value, $weights, $description, $extra);
1425 }
```

3.4 Database abstraction layer

Functions

- db_prefix_tables ($sql)
- db_set_active ($name= 'default')
- db_query ($query)
- db_queryd ($query)
- _db_rewrite_sql ($query= '', $primary_table= 'n', $primary_field= 'nid', $args=array())
- db_rewrite_sql ($query, $primary_table= 'n', $primary_field= 'nid', $args=array())
- db_connect ($url)
- pager_query ($query, $limit=10, $element=0, $count_query=NULL)
- tablesort_sql ($header, $before= '')

3.4.1 Detailed Description

Allow the use of different database servers using the same code base.

Drupal provides a slim database abstraction layer to provide developers with the ability to support multiple database servers easily. The intent of this layer is to preserve the syntax and power of SQL as much as possible, while letting Drupal control the pieces of queries that need to be written differently for different servers and provide basic security checks.

Most Drupal database queries are performed by a call to db_query() or db_query_range(). Module authors should also consider using pager_query() for queries that return results that need to be presented on multiple pages, and tablesort_sql() for generating appropriate queries for sortable tables.

For example, one might wish to return a list of the most recent 10 nodes authored by a given user. Instead of directly issuing the SQL query

```
SELECT n.title, n.body, n.created FROM node n WHERE n.uid = $uid LIMIT 0, 10;
```

one would instead call the Drupal functions:

```
$result = db_query_range('SELECT n.title, n.body, n.created
  FROM {node} n WHERE n.uid = %d', $uid, 0, 10);
while ($node = db_fetch_object($result)) {
  // Perform operations on $node->body, etc. here.
}
```

Curly braces are used around "node" to provide table prefixing via db_prefix_tables(). The explicit use of a user ID is pulled out into an argument passed to db_query() so that SQL injection attacks from user input can be caught and nullified. The LIMIT syntax varies between database servers, so that is abstracted into db_query_range() arguments. Finally, note the common pattern of iterating over the result set using db_fetch_object().

3.4.2 Function Documentation

3.4.2.1 _db_rewrite_sql ($ *query* = ”, $ *primary_table* = ’n’, $ *primary_field* = ’nid’, $ *args* = array())

Helper function for db_rewrite_sql.

Collects JOIN and WHERE statements via hook_sql. Decides whether to select primary_key or DISTINCT(primary_key)

Parameters:

$query Query to be rewritten.

$primary_table Name or alias of the table which has the primary key field for this query. Possible values are: comments, forum, node, term_data, vocabulary.

$primary_field Name of the primary field.

$args Array of additional arguments.

Returns:

An array: join statements, where statements, field or DISTINCT(field).

Definition at line 195 of file database.inc.

```
195 {
196   $where = array();
197   $join = array();
198   $distinct = FALSE;
199   foreach (module_implements(’db_rewrite_sql’) as $module) {
200     $result = module_invoke($module, ’db_rewrite_sql’, $query, $primary_table, $primary_field, $args);
201     if (is_array($result)) {
202       if (isset($result[’where’])) {
203         $where[] .= $result[’where’];
204       }
205       if (isset($result[’join’])) {
206         $join[] .= $result[’join’];
207       }
208       if (isset($result[’distinct’]) && $result[’distinct’]) {
209         $distinct = TRUE;
210       }
211     }
212     elseif (isset($result)) {
213       $where[] .= $result;
214     }
215   }
216
217   $where = empty($where) ? ’’ : ’(’. implode(’) AND (’, $where) .’)’;
218   $join = empty($join) ? ’’ : implode(’ ’, $join);
219
220   return array($join, $where, $distinct);
221 }
```

3.4.2.2 db_connect ($ *url*)

Initialize a database connection.

Note that you can change the mysql_connect() call to mysql_pconnect() if you want to use persistent connections. This is not recommended on shared hosts, and might require additional database/webserver tuning. It can increase performance, however, when the overhead to connect to your database is high (e.g. your database and web server live on different machines).

Definition at line 23 of file database.mysql.inc.

References url().

Referenced by db_set_active().

```
23                          {
24   $url = parse_url($url);
25
26   // Allow for non-standard MySQL port.
27   if (isset($url['port'])) {
28      $url['host'] = $url['host'] .':'. $url['port'];
29   }
30
31   $connection = mysql_connect($url['host'], $url['user'], $url['pass'], TRUE) or die(mysql_error());
32   mysql_select_db(substr($url['path'], 1)) or die('unable to select database');
33
34   return $connection;
35 }
```

3.4.2.3 db_prefix_tables ($ *sql*)

Append a database prefix to all tables in a query.

Queries sent to Drupal should wrap all table names in curly brackets. This function searches for this syntax and adds Drupal's table prefix to all tables, allowing Drupal to coexist with other systems in the same database if necessary.

Parameters:

$sql A string containing a partial or entire SQL query.

Returns:

The properly-prefixed string.

Definition at line 59 of file database.inc.

Referenced by db_next_id(), and db_query_range().

```
59                                {
60   global $db_prefix;
61
62   if (is_array($db_prefix)) {
63     if (array_key_exists('default', $db_prefix)) {
64       $tmp = $db_prefix;
65       unset($tmp['default']);
66       foreach ($tmp as $key => $val) {
67         $sql = strtr($sql, array('{'. $key. '}' => $val. $key));
68       }
69       return strtr($sql, array('{' => $db_prefix['default'], '}' => ''));
70     }
71     else {
72       foreach ($db_prefix as $key => $val) {
73         $sql = strtr($sql, array('{'. $key. '}' => $val. $key));
74       }
75       return strtr($sql, array('{' => '', '}' => ''));
76     }
77   }
78   else {
79     return strtr($sql, array('{' => $db_prefix, '}' => ''));
80   }
81 }
```

3.4.2.4 db_query ($ *query*)

Runs a basic query in the active database.

User-supplied arguments to the query should be passed in as separate parameters so that they can be properly escaped to avoid SQL injection attacks.

Parameters:

$query A string containing an SQL query.

... A variable number of arguments which are substituted into the query using printf() syntax. Instead of a variable number of query arguments, you may also pass a single array containing the query arguments.

Returns:

A database query result resource, or FALSE if the query was not executed correctly.

Definition at line 145 of file database.inc.

Referenced by _locale_add_language(), _locale_admin_manage_screen(), _locale_export_po(), _locale_string_edit(), _locale_string_save(), _menu_build(), cache_clear_all(), cache_get(), cache_set(), db_next_id(), drupal_get_filename(), drupal_get_path_map(), list_theme_engines(), list_themes(), menu_rebuild(), module_list(), node_access(), pager_query(), theme_aggregator_page_item(), update_101(), update_97(), update_98(), variable_del(), variable_init(), variable_set(), and watchdog().

```
145                           {
146   $args = func_get_args();
147   $query = db_prefix_tables($query);
148   if (count($args) > 1) {
149     if (is_array($args[1])) {
150       $args = array_merge(array($query), $args[1]);
151     }
152     $args = array_map('db_escape_string', $args);
153     $args[0] = $query;
154     $query = call_user_func_array('sprintf', $args);
155   }
156   return _db_query($query);
157 }
```

3.4.2.5 db_queryd ($ *query*)

Debugging version of db_query().

Echoes the query to the browser.

Definition at line 164 of file database.inc.

```
164                            {
165   $args = func_get_args();
166   $query = db_prefix_tables($query);
167   if (count($args) > 1) {
168     if (is_array($args[1])) {
169       $args = array_merge(array($query), $args[1]);
170     }
171     $args = array_map('db_escape_string', $args);
172     $args[0] = $query;
173     $query = call_user_func_array('sprintf', $args);
174   }
175   return _db_query($query, 1);
176 }
```

3.4.2.6 db_rewrite_sql ($ *query*, $ *primary_table* = 'n', $ *primary_field* = 'nid', $ *args* = array())

Rewrites node queries.

Parameters:

$query Query to be rewritten.

$primary_table Name or alias of the table which has the primary key field for this query. Possible values are: comments, forum, node, term_data, vocabulary.

$primary_field Name of the primary field.

$args An array of arguments, passed to the implementations of hook_db_rewrite_sql.

Returns:

The original query with JOIN and WHERE statements inserted from hook_db_rewrite_sql implementations. nid is rewritten if needed.

Definition at line 237 of file database.inc.

```
237                                                                                                                 {
238   list($join, $where, $distinct) = _db_rewrite_sql($query, $primary_table, $primary_field, $args);
239
240   if ($distinct) {
241     $field_to_select = 'DISTINCT('. $primary_table .'.'. $primary_field .')';
242     // (?<!text) is a negative look-behind (no need to rewrite queries that already use DISTINCT).
243     $query = preg_replace('/(SELECT.*)('. $primary_table .'\.)?(?<!DISTINCT\()(?<!DISTINCT\('. $primar
244   }
245
246   if (!empty($join)) {
247     $query = preg_replace('|FROM[^[:upper:]/,]+|','\0 '. $join .' ', $query);
248   }
249
250   if (!empty($where)) {
251     if (strpos($query, 'WHERE')) {
252       $replace = 'WHERE';
253       $add = 'AND';
254     }
255     elseif (strpos($query, 'GROUP')) {
256       $replace = 'GROUP';
257       $add = 'GROUP';
258     }
259     elseif (strpos($query, 'ORDER')) {
260       $replace = 'ORDER';
261       $add = 'ORDER';
262     }
263     elseif (strpos($query, 'LIMIT')) {
264       $replace = 'LIMIT';
265       $add = 'LIMIT';
266     }
267     else {
268       $query .= ' WHERE '. $where;
269     }
270     if (isset($replace)) {
271       $query = str_replace($replace, 'WHERE  '. $where .' '. $add .' ', $query);
272     }
273   }
274
275   return $query;
276 }
```

3.4.2.7 db_set_active ($ *name* = 'default')

Activate a database for future queries.

If it is necessary to use external databases in a project, this function can be used to change where database queries are sent. If the database has not yet been used, it is initialized using the URL specified for that name in Drupal's configuration file. If this name is not defined, a duplicate of the default connection is made instead.

Be sure to change the connection back to the default when done with custom code.

Parameters:

$name The name assigned to the newly active database connection. If omitted, the default connection will be made active.

Definition at line 99 of file database.inc.

References $db_url, and db_connect().

```
99                                          {
100   global $db_url, $db_type, $active_db;
101   static $db_conns;
102
103   if (!isset($db_conns[$name])) {
104     // Initiate a new connection, using the named DB URL specified.
105     if (is_array($db_url)) {
106       $connect_url = array_key_exists($name, $db_url) ? $db_url[$name] : $db_url['default'];
107     }
108     else {
109       $connect_url = $db_url;
110     }
111
112     $db_type = substr($connect_url, 0, strpos($connect_url, '://'));
113     $handler = "includes/database.$db_type.inc";
114
115     if (is_file($handler)) {
116       include_once($handler);
117     }
118     else {
119       die('Unsupported database type');
120     }
121
122     $db_conns[$name] = db_connect($connect_url);
123
124   }
125   // Set the active connection.
126   $active_db = $db_conns[$name];
127 }
```

3.4.2.8 pager_query ($ *query*, $ *limit* = 10, $ *element* = 0, $ *count_query* = NULL)

Perform a paged database query.

Use this function when doing select queries you wish to be able to page. The pager uses LIMIT-based queries to fetch only the records required to render a certain page. However, it has to learn the total number of records returned by the query to compute the number of pages (the number of records / records per page). This is done by inserting "COUNT(∗)" in the original query. For example, the query "SELECT nid, type FROM node WHERE status = '1' ORDER BY sticky DESC, created DESC" would be rewritten to

read "SELECT COUNT(∗) FROM node WHERE status = ’1’ ORDER BY sticky DESC, created DESC". Rewriting the query is accomplished using a regular expression.

Unfortunately, the rewrite rule does not always work as intended for queries that already have a "COUNT(∗)" or a "GROUP BY" clause, and possibly for other complex queries. In those cases, you can optionally pass a query that will be used to count the records.

For example, if you want to page the query "SELECT COUNT(∗), TYPE FROM node GROUP BY TYPE", pager_query() would invoke the incorrect query "SELECT COUNT(∗) FROM node GROUP BY TYPE". So instead, you should pass "SELECT COUNT(DISTINCT(TYPE)) FROM node" as the optional $count_query parameter.

Parameters:

$query The SQL query that needs paging.

$limit The number of query results to display per page.

$element An optional integer to distinguish between multiple pagers on one page.

$count_query An SQL query used to count matching records.

... A variable number of arguments which are substituted into the query (and the count query) using printf() syntax. Instead of a variable number of query arguments, you may also pass a single array containing the query arguments.

Returns:

A database query result resource, or FALSE if the query was not executed correctly.

Definition at line 51 of file pager.inc.

References db_query(), db_query_range(), and db_result().

Referenced by _locale_string_seek().

```
51                                                                                {
52   global $pager_from_array, $pager_total;
53   $from = $_GET['from'];
54
55   // Substitute in query arguments.
56   $args = func_get_args();
57   $args = array_slice($args, 4);
58   // Alternative syntax for '...'
59   if (is_array($args[0])) {
60     $args = $args[0];
61   }
62
63   // Construct a count query if none was given.
64   if (!isset($count_query)) {
65     $count_query = preg_replace(array('/SELECT.*?FROM/As', '/ORDER BY .*/'), array('SELECT COUNT(*) FRO
66   }
67
68   // Convert comma-separated $from to an array, used by other functions.
69   $pager_from_array = explode(',', $from);
70
71   if (count($args)) {
72     $pager_total[$element] = db_result(db_query($count_query, $args));
73     return db_query_range($query, $args, (int)$pager_from_array[$element], (int)$limit);
74   }
75   else {
76     $pager_total[$element] = db_result(db_query($count_query));
77     return db_query_range($query, (int)$pager_from_array[$element], (int)$limit);
78   }
79 }
```

3.4.2.9 tablesort_sql ($ *header*, $ *before* = ”)

Create an SQL sort clause.

This function produces the ORDER BY clause to insert in your SQL queries, assuring that the returned database table rows match the sort order chosen by the user.

Parameters:

$header An array of column headers in the format described in theme_table().

$before An SQL string to insert after ORDER BY and before the table sorting code. Useful for sorting by important attributes like "sticky" first.

Returns:

An SQL string to append to the end of a query.

Definition at line 51 of file tablesort.inc.

References db_escape_string(), and tablesort_init().

```
51                                                {
52   $ts = tablesort_init($header);
53   if ($ts['sql']) {
54     $sql = db_escape_string($ts['sql']);
55     $sort = strtoupper(db_escape_string($ts['sort']));
56     return " ORDER BY $before $sql $sort";
57   }
58 }
```

3.5 File interface

Enumerations

- enum **IS_WINDOWS**
- enum **FILE_DOWNLOADS_PUBLIC**
- enum **FILE_DOWNLOADS_PRIVATE**
- enum **FILE_CREATE_DIRECTORY**
- enum **FILE_MODIFY_PERMISSIONS**
- enum **FILE_DIRECTORY_TEMP**
- enum **FILE_EXISTS_RENAME**
- enum **FILE_EXISTS_REPLACE**
- enum **FILE_EXISTS_ERROR**

Functions

- file_create_url ($path)
- file_create_path ($dest=0)
- file_check_directory (&$directory, $mode=0, $form_item=NULL)
- file_check_path (&$path)
- file_check_upload ($source)
- file_check_location ($source, $directory=0)
- file_copy (&$source, $dest=0, $replace=FILE_EXISTS_RENAME)
- file_move (&$source, $dest=0, $replace=FILE_EXISTS_RENAME)
- **file_create_filename** ($basename, $directory)
- **file_delete** ($path)
- file_save_upload ($source, $dest=0, $replace=FILE_EXISTS_RENAME)
- file_save_data ($data, $dest, $replace=FILE_EXISTS_RENAME)
- file_transfer ($source, $headers)
- file_download ()
- file_scan_directory ($dir, $mask, $nomask=array('.', '..', 'CVS'), $callback=0, $recurse=TRUE, $key= 'filename', $min_depth=0, $depth=0)

3.5.1 Detailed Description

Common file handling functions.

3.5.2 Function Documentation

3.5.2.1 file_check_directory (&$ *directory*, $ *mode* = 0, $ *form_item* = NULL)

Check that directory exists and is writable.

Parameters:

$directory Path to extract and verify directory for.

$mode Try to create the directory if it does not exist.

$form_item Optional name for a field item to attach potential errors to.

Returns:

False when directory not found, or true when directory exists.

Definition at line 81 of file file.inc.

References drupal_set_message(), form_item(), form_set_error(), and theme().

Referenced by file_check_path().

```
81                                                                  {
82   $directory = rtrim($directory, '/\\');
83
84   // Check if directory exists.
85   if (!is_dir($directory)) {
86     if (($mode & FILE_CREATE_DIRECTORY) && @mkdir($directory, 0760)) {
87       drupal_set_message(t('Created directory %directory.', array('%directory' => theme('placeholder',
88     }
89     else {
90       if ($form_item) {
91         form_set_error($form_item, t('The directory %directory does not exist.', array('%directory' =>
92       }
93       return false;
94     }
95   }
96
97   // Check to see if the directory is writable.
98   if (!is_writable($directory)) {
99     if (($mode & FILE_MODIFY_PERMISSIONS) && @chmod($directory, 0760)) {
100       drupal_set_message(t('Modified permissions on directory %directory.', array('%directory' => them
101     }
102     else {
103       form_set_error($form_item, t('The directory %directory is not writable.', array('%directory' =>
104       return false;
105     }
106   }
107
108   return true;
109 }
```

3.5.2.2 file_check_location ($ *source*, $ *directory* = 0)

Check if a file is really located inside $directory. Should be used to make sure a file specified is really located within the directory to prevent exploits.

```
// Returns false:
file_check_location('/www/example.com/files/../../../etc/passwd', '/www/example.com/files');
```

Parameters:

$source A string set to the file to check.

$directory A string where the file should be located.

Returns:
0 for invalid path or the real path of the source.

Definition at line 175 of file file.inc.

Referenced by file_create_path().

```
175                                                     {
176   $check = realpath($source);
177   if ($check) {
178     $source = $check;
179   }
180   else {
181     // This file does not yet exist
182     $source = realpath(dirname($source)) .'/'. basename($source);
183   }
184   $directory = realpath($directory);
185   if ($directory && strpos($source, $directory) !== 0) {
186     return 0;
187   }
188   return $source;
189 }
```

3.5.2.3 file_check_path (&$ *path*)

Checks path to see if it is a directory, or a dir/file.

Parameters:
$path

Definition at line 116 of file file.inc.

References file_check_directory().

Referenced by file_copy().

```
116                                 {
117   // Check if path is a directory.
118   if (file_check_directory($path)) {
119     return '';
120   }
121
122   // Check if path is a possible dir/file.
123   $filename = basename($path);
124   $path = dirname($path);
125   if (file_check_directory($path)) {
126     return $filename;
127   }
128
129   return false;
130 }
```

3.5.2.4 file_check_upload ($ *source*)

Check if $source is a valid file upload.

Parameters:

$source

Definition at line 137 of file file.inc.

Referenced by file_save_upload().

```
137                                 {
138   if (is_object($source)) {
139     if (is_file($source->filepath)) {
140       return $source;
141     }
142   }
143   elseif ($_FILES["edit"]["name"][$source] && is_uploaded_file($_FILES["edit"]["tmp_name"][$source])) {
144     $file = new StdClass();
145     $file->filename = trim(basename($_FILES["edit"]["name"][$source]), '.');
146     $file->filemime = $_FILES["edit"]["type"][$source];
147     $file->filepath = $_FILES["edit"]["tmp_name"][$source];
148     $file->error = $_FILES["edit"]["error"][$source];
149     $file->filesize = $_FILES["edit"]["size"][$source];
150     $file->source = $source;
151     return $file;
152   }
153   else {
154     // In case of previews return previous file object.
155     if (file_exists($_SESSION['file_uploads'][$source]->filepath)) {
156       return $_SESSION['file_uploads'][$source];
157     }
158   }
159 }
```

3.5.2.5 file_copy (&$ *source*, $ *dest* = 0, $ *replace* = FILE_EXISTS_RENAME)

Copies a file to a new location. This is a powerful function that in many ways performs like an advanced version of copy().

- Checks if $source and $dest are valid and readable/writable.
- Performs a file copy if $source is not equal to $dest.
- If file already exists in $dest either the call will error out, replace the file or rename the file based on the $replace parameter.

Parameters:

$source A string specifying the file location of the original file. This parameter will contain the resulting destination filename in case of success.

$dest A string containing the directory $source should be copied to.

$replace Replace behavior when the destination file already exists.

- FILE_EXISTS_REPLACE - Replace the existing file
- FILE_EXISTS_RENAME - Append _{incrementing number} until the filename is unique
- FILE_EXISTS_ERROR - Do nothing and return false.

Returns:

True for success, false for failure.

Definition at line 209 of file file.inc.

References drupal_set_message(), file_check_path(), and file_create_path().

Referenced by file_move().

```
209                                                                         {
210   $dest = file_create_path($dest);
211
212   $directory = $dest;
213   $basename = file_check_path($directory);
214
215   // Make sure we at least have a valid directory.
216   if ($basename === false) {
217     drupal_set_message(t('File copy failed: no directory configured, or it could not be accessed.'), '
218     return 0;
219   }
220
221   // Process a file upload object.
222   if (is_object($source)) {
223     $file = $source;
224     $source = $file->filepath;
225     if (!$basename) {
226       $basename = $file->filename;
227     }
228   }
229
230   $source = realpath($source);
231   if (!file_exists($source)) {
232     drupal_set_message(t('File copy failed: source file does not exist.'), 'error');
233     return 0;
234   }
235
236   // If destination file is not specified then use filename of source file.
237   $basename = $basename ? $basename : basename($source);
238   $dest = $directory .'/'. $basename;
239
240   // Make sure source and destination filenames are not the same, makes no sense
241   // to copy it if they are. In fact copying the file will most likely result in
242   // a 0 byte file. Which is bad. Real bad.
243   if ($source != realpath($dest)) {
244     if (file_exists($dest)) {
245       switch ($replace) {
246         case FILE_EXISTS_RENAME:
247           // Destination file already exists and we can't replace is so we try and
248           // and find a new filename.
249           if ($pos = strrpos($basename, '.')) {
250             $name = substr($basename, 0, $pos);
251             $ext = substr($basename, $pos);
252           }
253           else {
254             $name = $basename;
255           }
256
257           $counter = 0;
258           do {
259             $dest = $directory .'/'. $name .'_'. $counter++ . $ext;
260           } while (file_exists($dest));
261           break;
262
263         case FILE_EXISTS_ERROR:
264           drupal_set_message(t('File copy failed. File already exists.'), 'error');
265           return 0;
266
```

```
267         case FILE_EXISTS_REPLACE:
268           // Leave $dest where it is for replace.
269       }
270     }
271
272     if (!@copy($source, $dest)) {
273       drupal_set_message(t('File copy failed.'), 'error');
274       return 0;
275     }
276   }
277
278   if (is_object($file)) {
279     $file->filename = $basename;
280     $file->filepath = $dest;
281     $source = $file;
282   }
283   else {
284     $source = $dest;
285   }
286
287   return 1; // Everything went ok.
288 }
```

3.5.2.6 file_create_path ($ *dest* = 0)

Make sure the destination is a complete path and resides in the file system directory, if it is not prepend the file system directory.

Parameters:

$dest Path to verify

Returns:

Path to file with file system directory appended if necessary. Returns FALSE if the path is invalid (i.e. outside the configured 'files'-directory).

Definition at line 52 of file file.inc.

References file_check_location(), and variable_get().

Referenced by file_copy(), file_download(), and file_transfer().

```
52                                      {
53   $file_path = variable_get('file_directory_path', 'files');
54   if (!$dest) {
55     return $file_path;
56   }
57   // file_check_location() checks whether the destination is inside the Drupal files directory.
58   if (file_check_location($dest, $file_path)) {
59     return $dest;
60   }
61   // check if the destination is instead inside the Drupal temporary files directory.
62   else if (file_check_location($dest, variable_get('file_directory_temp', FILE_DIRECTORY_TEMP))) {
63     return $dest;
64   }
65   // Not found, try again with prefixed dirctory path.
66   else if (file_check_location($file_path . '/' . $dest, $file_path)) {
67     return $file_path . '/' . $dest;
68   }
69   // File not found.
70   return FALSE;
71 }
```

3.5.2.7 file_create_url ($ *path*)

Create the download path to a file.

Parameters:

$path Path to the file to generate URL for

Returns:

URL pointing to the file

Definition at line 31 of file file.inc.

References url(), and variable_get().

```
31                                {
32   if (strpos($path, variable_get('file_directory_path', 'files')) !== false) {
33     $path = trim(substr($path, strlen(variable_get('file_directory_path', 'files'))), '\\/');
34   }
35   switch (variable_get('file_downloads', FILE_DOWNLOADS_PUBLIC)) {
36     case FILE_DOWNLOADS_PUBLIC:
37       return $GLOBALS['base_url'] .'/'. variable_get('file_directory_path', 'files') .'/'. str_replace(
38     case FILE_DOWNLOADS_PRIVATE:
39       return url('system/files', 'file='. $path);
40   }
41 }
```

3.5.2.8 file_download ()

Call modules to find out if a file is accessible for a given user.

Definition at line 473 of file file.inc.

References drupal_access_denied(), drupal_not_found(), file_create_path(), file_download(), file_-transfer(), module_invoke(), and module_list().

Referenced by file_download().

```
473                           {
474   $file = $_GET['file'];
475   if (file_exists(file_create_path($file))) {
476     $list = module_list();
477     foreach ($list as $module) {
478       $headers = module_invoke($module, 'file_download', $file);
479       if ($headers === -1) {
480         drupal_access_denied();
481       }
482       elseif (is_array($headers)) {
483         file_transfer($file, $headers);
484       }
485     }
486   }
487   drupal_not_found();
488 }
```

3.5.2.9 file_move (&$ *source*, $ *dest* = 0, $ *replace* = FILE_EXISTS_RENAME)

Moves a file to a new location.

- Checks if $source and $dest are valid and readable/writable.
- Performs a file move if $source is not equal to $dest.
- If file already exists in $dest either the call will error out, replace the file or rename the file based on the $replace parameter.

Parameters:

$source A string specifying the file location of the original file. This parameter will contain the resulting destination filename in case of success.

$dest A string containing the directory $source should be copied to.

$replace Replace behavior when the destination file already exists.

- FILE_EXISTS_REPLACE - Replace the existing file
- FILE_EXISTS_RENAME - Append _{incrementing number} until the filename is unique
- FILE_EXISTS_ERROR - Do nothing and return false.

Returns:

True for success, false for failure.

Definition at line 307 of file file.inc.

References drupal_set_message(), and file_copy().

Referenced by file_save_data(), and file_save_upload().

```
307                                                                     {
308
309   $path_original = is_object($source) ? $source->filepath : $source;
310
311   if (file_copy($source, $dest, $replace)) {
312     $path_current = is_object($source) ? $source->filepath : $source;
313
314     if ($path_original == $path_current || file_delete($path_original)) {
315       return 1;
316     }
317     drupal_set_message(t('Removing original file failed.'), 'error');
318   }
319   return 0;
320 }
```

3.5.2.10 file_save_data ($ *data*, $ *dest*, $ *replace* = FILE_EXISTS_RENAME)

Save a string to the specified destination

Parameters:

$data A string containing the contents of the file

$dest A string containing the destination location

Returns:

A string containing the resulting filename or 0 on error

Definition at line 418 of file file.inc.

References drupal_set_message(), file_move(), valid_input_data(), variable_get(), and watchdog().

```
418                                                                     {
419   if (!user_access('bypass input data check') && !valid_input_data($data)) {
420     watchdog('security', t('Possible exploit abuse: invalid data.'), WATCHDOG_WARNING);
421     drupal_set_message(t('File upload failed: invalid data.'), 'error');
422     return 0;
423   }
424
425   $temp = variable_get('file_directory_temp', FILE_DIRECTORY_TEMP);
426   $file = tempnam($temp, 'file');
427   if (!$fp = fopen($file, 'wb')) {
428     drupal_set_message(t('Unable to create file.'), 'error');
429     return 0;
430   }
431   fwrite($fp, $data);
432   fclose($fp);
433
434   if (!file_move($file, $dest, $replace)) {
435     return 0;
436   }
437
438   return $file;
439 }
```

3.5.2.11 file_save_upload ($ *source*, $ *dest* = 0, $ *replace* = FILE_EXISTS_RENAME)

Saves a file upload to a new location. The source file is validated as a proper upload and handled as such.

Parameters:

$source A string specifying the name of the upload field to save. This parameter will contain the resulting destination filename in case of success.

$dest A string containing the directory $source should be copied to, will use the temporary directory in case no other value is set.

$replace A boolean, set to true if the destination should be replaced when in use, but when false append a _X to the filename.

Returns:

An object containing file info or 0 in case of error.

Definition at line 363 of file file.inc.

References drupal_set_message(), file_check_upload(), file_move(), valid_input_data(), variable_get(), and watchdog().

```
363                                                                     {
364   // Make sure $source exists in $_FILES.
365   if ($file = file_check_upload($source)) {
366     if (!$dest) {
367       $dest = variable_get('file_directory_temp', FILE_DIRECTORY_TEMP);
368       $temporary = 1;
369       if (is_file($file->filepath)) {
370         // If this file was uploaded by this user before replace the temporary copy.
371         $replace = 1;
372       }
```

```
373     }
374
375     if (!user_access('bypass input data check') && !valid_input_data($file)) {
376       watchdog('security', t('Possible exploit abuse: invalid data.'), WATCHDOG_WARNING);
377       drupal_set_message(t('File upload failed: invalid data.'), 'error');
378       return 0;
379     }
380
381     // Check for file upload errors.
382     switch ($file->error) {
383       case 0: // UPLOAD_ERR_OK
384         break;
385       case 1: // UPLOAD_ERR_INI_SIZE
386       case 2: // UPLOAD_ERR_FORM_SIZE
387         drupal_set_message(t('File upload failed: file size too big.'), 'error');
388         return 0;
389       case 3: // UPLOAD_ERR_PARTIAL
390       case 4: // UPLOAD_ERR_NO_FILE
391         drupal_set_message(t('File upload failed: incomplete upload.'), 'error');
392         return 0;
393       default: // Unknown error
394         drupal_set_message(t('File upload failed: unknown error.'), 'error');
395         return 0;
396     }
397
398     unset($_SESSION['file_uploads'][is_object($source) ? $source->source : $source]);
399     if (file_move($file, $dest, $replace)) {
400       if ($temporary) {
401         $_SESSION['file_uploads'][is_object($source) ? $source->source : $source] = $file;
402       }
403       return $file;
404     }
405     return 0;
406   }
407   return 0;
408 }
```

3.5.2.12 file_scan_directory ($ *dir*, $ *mask*, $ *nomask* = array('.', '..', 'CVS'), $ *callback* = 0, $ *recurse* = TRUE, $ *key* = 'filename', $ *min_depth* = 0, $ *depth* = 0)

Finds all files that match a given mask in a given directory.

Parameters:

$dir The base directory for the scan.

$mask The regular expression of the files to find.

$nomask An array of files/directories to ignore.

$callback The callback function to call for each match.

$recurse When TRUE, the directory scan will recurse the entire tree starting at the provided directory.

$key The key to be used for the returned array of files. Possible values are "filename", for the path starting with $dir, "basename", for the basename of the file, and "name" for the name of the file without an extension.

$min_depth Minimum depth of directories to return files from.

$depth Current depth of recursion. This parameter is only used internally and should not be passed.

Returns:

An associative array (keyed on the provided key) of objects with "path", "basename", and "name" members corresponding to the matching files.

Definition at line 520 of file file.inc.

```
520
521   $key = (in_array($key, array('filename', 'basename', 'name')) ? $key : 'filename');
522   $files = array();
523
524   if (is_dir($dir) && $handle = opendir($dir)) {
525     while ($file = readdir($handle)) {
526       if (!in_array($file, $nomask)) {
527         if (is_dir("$dir/$file") && $recurse) {
528           $files = array_merge($files, file_scan_directory("$dir/$file", $mask, $nomask, $callback, $r
529         }
530         elseif ($depth >= $min_depth && ereg($mask, $file)) {
531           $filename = "$dir/$file";
532           $basename = basename($file);
533           $name = substr($basename, 0, strrpos($basename, '.'));
534           $files[$$key] = new stdClass();
535           $files[$$key]->filename = $filename;
536           $files[$$key]->basename = $basename;
537           $files[$$key]->name = $name;
538           if ($callback) {
539             $callback($filename);
540           }
541         }
542       }
543     }
544
545     closedir($handle);
546   }
547
548   return $files;
549 }
```

3.5.2.13 file_transfer ($ *source*, $ *headers*)

Transfer file using http to client. Pipes a file through Drupal to the client.

Parameters:

$source File to transfer.

$headers An array of http headers to send along with file.

Definition at line 448 of file file.inc.

References drupal_not_found(), and file_create_path().

Referenced by file_download().

```
448                                           {
449   ob_end_clean();
450
451   foreach ($headers as $header) {
452     header($header);
453   }
454
455   $source = file_create_path($source);
456
457   // Transfer file in 1024 byte chunks to save memory usage.
458   if ($fd = fopen($source, 'rb')) {
459     while (!feof($fd)) {
460       print fread($fd, 1024);
```

```
461     }
462     fclose($fd);
463   }
464   else {
465     drupal_not_found();
466   }
467   exit();
468 }
```

3.6 Menu system

Menu flags

Flags for use in the "type" attribute of menu items.

- enum **MENU_IS_ROOT**
- enum **MENU_VISIBLE_IN_TREE**
- enum **MENU_VISIBLE_IN_BREADCRUMB**
- enum **MENU_VISIBLE_IF_HAS_CHILDREN**
- enum **MENU_MODIFIABLE_BY_ADMIN**
- enum **MENU_MODIFIED_BY_ADMIN**
- enum **MENU_CREATED_BY_ADMIN**
- enum **MENU_IS_LOCAL_TASK**
- enum **MENU_EXPANDED**
- enum **MENU_LINKS_TO_PARENT**

Menu item types

Menu item definitions provide one of these constants, which are shortcuts for combinations of the above flags.

- enum MENU_NORMAL_ITEM
- enum MENU_ITEM_GROUPING
- enum MENU_CALLBACK
- enum MENU_DYNAMIC_ITEM
- enum MENU_SUGGESTED_ITEM
- enum MENU_LOCAL_TASK
- enum MENU_DEFAULT_LOCAL_TASK
- enum MENU_CUSTOM_ITEM
- enum MENU_CUSTOM_MENU

Menu status codes

Status codes for menu callbacks.

- enum **MENU_FOUND**
- enum **MENU_NOT_FOUND**
- enum **MENU_ACCESS_DENIED**

Functions

- menu_get_menu ()
- menu_get_local_tasks ()
- menu_set_location ($location)

- menu_execute_active_handler ()
- menu_get_active_item ()
- menu_set_active_item ($path=NULL)
- menu_get_active_nontask_item ()
- menu_get_active_title ()
- menu_get_active_help ()
- menu_get_active_breadcrumb ()
- menu_in_active_trail ($mid)
- menu_rebuild ()
- menu_tree ($pid=1)
- menu_item_link ($mid)
- menu_primary_local_tasks ()
- menu_secondary_local_tasks ()

3.6.1 Detailed Description

Define the navigation menus, and route page requests to code based on URLs.

The Drupal menu system drives both the navigation system from a user perspective and the callback system that Drupal uses to respond to URLs passed from the browser. For this reason, a good understanding of the menu system is fundamental to the creation of complex modules.

Drupal's menu system follows a simple hierarchy defined by paths. Implementations of hook_menu() define menu items and assign them to paths (which should be unique). The menu system aggregates these items and determines the menu hierarchy from the paths. For example, if the paths defined were a, a/b, e, a/b/c/d, f/g, and a/b/h, the menu system would form the structure:

- a
  - a/b
    - a/b/c/d
    - a/b/h
- e
- f/g Note that the number of elements in the path does not necessarily determine the depth of the menu item in the tree.

When responding to a page request, the menu system looks to see if the path requested by the browser is registered as a menu item with a callback. If not, the system searches up the menu tree for the most complete match with a callback it can find. If the path a/b/i is requested in the tree above, the callback for a/b would be used.

The found callback function is called with any arguments specified in the "callback arguments" attribute of its menu item. After these arguments, any remaining components of the path are appended as further arguments. In this way, the callback for a/b above could respond to a request for a/b/i differently than a request for a/b/j.

For an illustration of this process, see page_example.module.

Access to the callback functions is also protected by the menu system. The "access" attribute of each menu item is checked as the search for a callback proceeds. If this attribute is TRUE, then access is granted; if FALSE, then access is denied. The first found "access" attribute determines the accessibility of the target. Menu items may omit this attribute to use the value provided by an ancestor item.

In the default Drupal interface, you will notice many links rendered as tabs. These are known in the menu system as "local tasks", and they are rendered as tabs by default, though other presentations are possible. Local tasks function just as other menu items in most respects. It is convention that the names of these tasks should be short verbs if possible. In addition, a "default" local task should be provided for each set. When visiting a local task's parent menu item, the default local task will be rendered as if it is selected; this provides for a normal tab user experience. This default task is special in that it links not to its provided path, but to its parent item's path instead. The default task's path is only used to place it appropriately in the menu hierarchy.

3.6.2 Enumeration Type Documentation

3.6.2.1 enum MENU_CALLBACK

Callbacks simply register a path so that the correct function is fired when the URL is accessed. They are not shown in the menu.

Definition at line 115 of file menu.inc.

3.6.2.2 enum MENU_CUSTOM_ITEM

Custom items are those defined by the administrator. Reserved for internal use; do not return from hook_-menu() implementations.

Definition at line 146 of file menu.inc.

3.6.2.3 enum MENU_CUSTOM_MENU

Custom menus are those defined by the administrator. Reserved for internal use; do not return from hook_-menu() implementations.

Definition at line 152 of file menu.inc.

3.6.2.4 enum MENU_DEFAULT_LOCAL_TASK

Every set of local tasks should provide one "default" task, that links to the same path as its parent when clicked.

Definition at line 140 of file menu.inc.

3.6.2.5 enum MENU_DYNAMIC_ITEM

Dynamic menu items change frequently, and so should not be stored in the database for administrative customization.

Definition at line 121 of file menu.inc.

3.6.2.6 enum MENU_ITEM_GROUPING

Item groupings are used for pages like "node/add" that simply list subpages to visit. They are distinguished from other pages in that they will disappear from the menu if no subpages exist.

Definition at line 109 of file menu.inc.

3.6.2.7 enum MENU_LOCAL_TASK

Local tasks are rendered as tabs by default. Use this for menu items that describe actions to be performed on their parent item. An example is the path "node/52/edit", which performs the "edit" task on "node/52".

Definition at line 134 of file menu.inc.

3.6.2.8 enum MENU_NORMAL_ITEM

Normal menu items show up in the menu tree and can be moved/hidden by the administrator. Use this for most menu items. It is the default value if no menu item type is specified.

Definition at line 102 of file menu.inc.

Referenced by _menu_append_contextual_items(), and _menu_build().

3.6.2.9 enum MENU_SUGGESTED_ITEM

Modules may "suggest" menu items that the administrator may enable. They act just as callbacks do until enabled, at which time they act like normal items.

Definition at line 127 of file menu.inc.

3.6.3 Function Documentation

3.6.3.1 menu_execute_active_handler ()

Execute the handler associated with the active menu item.

This is called early in the page request. The active menu item is at this point determined exclusively by the URL. The handler that is called here may, as a side effect, change the active menu item so that later menu functions (that display the menus and breadcrumbs, for example) act as if the user were in a different location on the site.

Definition at line 326 of file menu.inc.

References _menu_item_is_accessible(), menu_get_active_item(), and menu_get_menu().

Referenced by drupal_access_denied(), and drupal_not_found().

```
326                                     {
327   $menu = menu_get_menu();
328
329   // Determine the menu item containing the callback.
330   $path = $_GET['q'];
331   while ($path && (!array_key_exists($path, $menu['path index']) || empty($menu['items'][$menu['path i
332     $path = substr($path, 0, strrpos($path, '/'));
```

```
333   }
334   if (!array_key_exists($path, $menu['path index'])) {
335     return MENU_NOT_FOUND;
336   }
337   $mid = $menu['path index'][$path];
338
339   if (empty($menu['items'][$mid]['callback'])) {
340     return MENU_NOT_FOUND;
341   }
342
343   if (!_menu_item_is_accessible(menu_get_active_item())) {
344     return MENU_ACCESS_DENIED;
345   }
346
347   // We found one, and are allowed to execute it.
348   $arguments = array_key_exists('callback arguments', $menu['items'][$mid]) ? $menu['items'][$mid]['ca
349   $arg = substr($_GET['q'], strlen($menu['items'][$mid]['path']) + 1);
350   if (strlen($arg)) {
351     $arguments = array_merge($arguments, explode('/', $arg));
352   }
353
354   call_user_func_array($menu['items'][$mid]['callback'], $arguments);
355   return MENU_FOUND;
356 }
```

3.6.3.2 menu_get_active_breadcrumb ()

Returns an array of rendered menu items in the active breadcrumb trail.

Definition at line 464 of file menu.inc.

References _menu_get_active_trail(), l(), menu_get_menu(), and menu_item_link().

Referenced by drupal_get_breadcrumb().

```
464                                       {
465   $menu = menu_get_menu();
466
467   $links[] = l(t('Home'), '');
468
469   $trail = _menu_get_active_trail();
470   foreach ($trail as $mid) {
471     if ($menu['items'][$mid]['type'] & MENU_VISIBLE_IN_BREADCRUMB) {
472       $links[] = menu_item_link($mid);
473     }
474   }
475
476   // The last item in the trail is the page title; don't display it here.
477   array_pop($links);
478
479   return $links;
480 }
```

3.6.3.3 menu_get_active_help ()

Returns the help associated with the active menu item.

Definition at line 435 of file menu.inc.

References _menu_item_is_accessible(), menu_get_active_item(), module_hook(), module_invoke(), module_list(), theme(), and url().

```
435                                 {
436   $path = $_GET['q'];
437   $output = '';
438
439   if (!_menu_item_is_accessible(menu_get_active_item())) {
440     // Don't return help text for areas the user cannot access.
441     return;
442   }
443
444   foreach (module_list() as $name) {
445     if (module_hook($name, 'help')) {
446       if ($temp = module_invoke($name, 'help', $path)) {
447         $output .= $temp . "\n";
448       }
449       if (module_hook('help', 'page')) {
450         if (substr($path, 0, 6) == "admin/") {
451           if (module_invoke($name, 'help', 'admin/help#' . substr($path, 6))) {
452             $output .= theme("more_help_link", url('admin/help/' . substr($path, 6)));
453           }
454         }
455       }
456     }
457   }
458   return $output;
459 }
```

3.6.3.4 menu_get_active_item ()

Returns the ID of the active menu item.

Definition at line 361 of file menu.inc.

References menu_set_active_item().

Referenced by _menu_get_active_trail(), menu_execute_active_handler(), menu_get_active_help(), and menu_get_active_nontask_item().

```
361                                 {
362   return menu_set_active_item();
363 }
```

3.6.3.5 menu_get_active_nontask_item ()

Returns the ID of the current menu item or, if the current item is a local task, the menu item to which this task is attached.

Definition at line 407 of file menu.inc.

References menu_get_active_item(), and menu_get_menu().

Referenced by menu_get_active_title(), menu_get_local_tasks(), menu_primary_local_tasks(), and menu_secondary_local_tasks().

```
407                                     {
408   $menu = menu_get_menu();
409   $mid = menu_get_active_item();
410
411   // Find the first non-task item:
412   while ($mid && ($menu['items'][$mid]['type'] & MENU_IS_LOCAL_TASK)) {
413     $mid = $menu['items'][$mid]['pid'];
414   }
415
416   if ($mid) {
417     return $mid;
418   }
419 }
```

3.6.3.6 menu_get_active_title ()

Returns the title of the active menu item.

Definition at line 424 of file menu.inc.

References menu_get_active_nontask_item(), and menu_get_menu().

Referenced by drupal_get_title().

```
424                                 {
425   $menu = menu_get_menu();
426
427   if ($mid = menu_get_active_nontask_item()) {
428     return $menu['items'][$mid]['title'];
429   }
430 }
```

3.6.3.7 menu_get_local_tasks ()

Return the local task tree.

Unlike the rest of the menu structure, the local task tree cannot be cached nor determined too early in the page request, because the user's current location may be changed by a menu_set_location() call, and the tasks shown (just as the breadcrumb trail) need to reflect the changed location.

Definition at line 232 of file menu.inc.

References _menu_build_local_tasks(), and menu_get_active_nontask_item().

Referenced by menu_primary_local_tasks(), and menu_secondary_local_tasks().

```
232                                 {
233   global $_menu;
234
235   // Don't cache the local task tree, as it varies by location and tasks are
236   // allowed to be dynamically determined.
237   if (!isset($_menu['local tasks'])) {
238     // _menu_build_local_tasks() may indirectly call this function, so prevent
239     // infinite loops.
240     $_menu['local tasks'] = array();
241     $pid = menu_get_active_nontask_item();
242     if (!_menu_build_local_tasks($pid)) {
```

```
243       // If the build returned FALSE, the tasks need not be displayed.
244       $_menu['local tasks'][$pid]['children'] = array();
245     }
246   }
247
248   return $_menu['local tasks'];
249 }
```

3.6.3.8 menu_get_menu ()

Return the menu data structure.

The returned structure contains much information that is useful only internally in the menu system. External modules are likely to need only the ['visible'] element of the returned array. All menu items that are accessible to the current user and not hidden will be present here, so modules and themes can use this structure to build their own representations of the menu.

$menu['visible'] will contain an associative array, the keys of which are menu IDs. The values of this array are themselves associative arrays, with the following key-value pairs defined:

- 'title' - The displayed title of the menu or menu item. It will already have been translated by the locale system.
- 'description' - The description (link title attribute) of the menu item. It will already have been translated by the locale system.
- 'path' - The Drupal path to the menu item. A link to a particular item can thus be constructed with l($item['title'], $item['path'], array('title' => $item['description'])).
- 'children' - A linear list of the menu ID's of this item's children.

Menu ID 0 is the "root" of the menu. The children of this item are the menus themselves (they will have no associated path). Menu ID 1 will always be one of these children; it is the default "Navigation" menu.

Definition at line 198 of file menu.inc.

References _menu_append_contextual_items(), _menu_build(), cache_get(), and cache_set().

Referenced by _menu_get_active_trail(), _menu_item_is_accessible(), _menu_sort(), menu_execute_active_handler(), menu_get_active_breadcrumb(), menu_get_active_nontask_item(), menu_get_active_title(), menu_item_link(), menu_rebuild(), menu_set_active_item(), and menu_tree().

```
198                         {
199   global $_menu;
200   global $user;
201   global $locale;
202
203   if (!isset($_menu['items'])) {
204     // _menu_build() may indirectly call this function, so prevent infinite loops.
205     $_menu['items'] = array();
206
207     $cid = "menu:$user->uid:$locale";
208     if ($cached = cache_get($cid)) {
209       $_menu = unserialize($cached->data);
210     }
211     else {
212       _menu_build();
213       // Cache the menu structure for this user, to expire after one day.
214       cache_set($cid, serialize($_menu), time() + (60 * 60 * 24));
```

```
215     }
216
217     // Make sure items that cannot be cached are added.
218     _menu_append_contextual_items();
219   }
220
221   return $_menu;
222 }
```

3.6.3.9 menu_in_active_trail ($ *mid*)

Returns true when the menu item is in the active trail.

Definition at line 485 of file menu.inc.

References _menu_get_active_trail().

Referenced by menu_primary_local_tasks(), menu_secondary_local_tasks(), and menu_tree().

```
485                                     {
486   $trail = _menu_get_active_trail();
487
488   return in_array($mid, $trail);
489 }
```

3.6.3.10 menu_item_link ($ *mid*)

Returns the rendered link to a menu item.

Parameters:

$mid The menu item id to render.

Definition at line 617 of file menu.inc.

References menu_get_menu(), menu_item_link(), and theme().

Referenced by menu_get_active_breadcrumb(), menu_item_link(), theme_menu_item(), and theme_-menu_local_task().

```
617                               {
618   $menu = menu_get_menu();
619
620   $link_mid = $mid;
621   while ($menu['items'][$link_mid]['type'] & MENU_LINKS_TO_PARENT) {
622     $link_mid = $menu['items'][$link_mid]['pid'];
623   }
624
625   return theme('menu_item_link', $menu['items'][$mid], $menu['items'][$link_mid]);
626 }
```

3.6.3.11 menu_primary_local_tasks ()

Returns the rendered HTML of the primary local tasks.

Definition at line 650 of file menu.inc.

References menu_get_active_nontask_item(), menu_get_local_tasks(), menu_in_active_trail(), and theme().

Referenced by theme_menu_local_tasks().

```
650                                     {
651   $local_tasks = menu_get_local_tasks();
652   $pid = menu_get_active_nontask_item();
653   $output = '';
654
655   if (count($local_tasks[$pid]['children'])) {
656     foreach ($local_tasks[$pid]['children'] as $mid) {
657       $output .= theme('menu_local_task', $mid, menu_in_active_trail($mid), TRUE);
658     }
659   }
660
661   return $output;
662 }
```

3.6.3.12 menu_rebuild ()

Populate the database representation of the menu.

This need only be called at the start of pages that modify the menu.

Definition at line 496 of file menu.inc.

References _menu_build(), cache_clear_all(), db_next_id(), db_query(), menu_get_menu(), and module_-exist().

Referenced by _locale_string_save().

```
496                         {
497   // Clear the page cache, so that changed menus are reflected for anonymous users.
498   cache_clear_all();
499   // Also clear the menu cache.
500   cache_clear_all('menu:', TRUE);
501
502   _menu_build();
503
504   if (module_exist('menu')) {
505     $menu = menu_get_menu();
506
507     $new_items = array();
508     foreach ($menu['items'] as $mid => $item) {
509       if ($mid < 0 && ($item['type'] & MENU_MODIFIABLE_BY_ADMIN)) {
510         $new_mid = db_next_id('{menu}_mid');
511         // Check explicitly for mid 1. If the database was improperly prefixed,
512         // this would cause a nasty infinite loop.
513         // TODO: have automatic prefixing through an installer to prevent this.
514         if ($new_mid == 1) {
515           $new_mid = db_next_id('{menu}_mid');
516         }
517         if (isset($new_items[$item['pid']])) {
518           $new_pid = $new_items[$item['pid']]['mid'];
519         }
```

```
520         else {
521           $new_pid = $item['pid'];
522         }
523
524         // Fix parent IDs for menu items already added.
525         if ($item['children']) {
526           foreach ($item['children'] as $child) {
527             if (isset($new_items[$child])) {
528               $new_items[$child]['pid'] = $new_mid;
529             }
530           }
531         }
532
533         $new_items[$mid] = array('mid' => $new_mid, 'pid' => $new_pid, 'path' => $item['path'], 'title
534       }
535     }
536
537     if (count($new_items)) {
538       foreach ($new_items as $item) {
539         db_query('INSERT INTO {menu} (mid, pid, path, title, description, weight, type) VALUES (%d, %d
540       }
541
542       // Rebuild the menu to account for the changes.
543       _menu_build();
544     }
545   }
546 }
```

3.6.3.13 menu_secondary_local_tasks ()

Returns the rendered HTML of the secondary local tasks.

Definition at line 667 of file menu.inc.

References menu_get_active_nontask_item(), menu_get_local_tasks(), menu_in_active_trail(), and theme().

Referenced by theme_menu_local_tasks().

```
667                                     {
668   $local_tasks = menu_get_local_tasks();
669   $pid = menu_get_active_nontask_item();
670   $output = '';
671
672   if (count($local_tasks[$pid]['children'])) {
673     foreach ($local_tasks[$pid]['children'] as $mid) {
674       if (menu_in_active_trail($mid) && count($local_tasks[$mid]['children']) > 1) {
675         foreach ($local_tasks[$mid]['children'] as $cid) {
676           $output .= theme('menu_local_task', $cid, menu_in_active_trail($cid), FALSE);
677         }
678       }
679     }
680   }
681
682   return $output;
683 }
```

3.6.3.14 menu_set_active_item ($ *path* = NULL)

Sets the path of the active menu item.

Definition at line 368 of file menu.inc.

References menu_get_menu().

Referenced by drupal_access_denied(), drupal_not_found(), menu_get_active_item(), and menu_set_-location().

```
368                                             {
369   static $stored_mid;
370   $menu = menu_get_menu();
371
372   if (is_null($stored_mid) || !empty($path)) {
373     if (empty($path)) {
374       $path = $_GET['q'];
375     }
376     else {
377       $_GET['q'] = $path;
378     }
379
380     while ($path && !array_key_exists($path, $menu['path index'])) {
381       $path = substr($path, 0, strrpos($path, '/'));
382     }
383     $stored_mid = array_key_exists($path, $menu['path index']) ? $menu['path index'][$path] : 0;
384
385     // Search for default local tasks to activate instead of this item.
386     $continue = TRUE;
387     while ($continue) {
388       $continue = FALSE;
389       if (array_key_exists('children', $menu['items'][$stored_mid])) {
390         foreach ($menu['items'][$stored_mid]['children'] as $cid) {
391           if ($menu['items'][$cid]['type'] & MENU_LINKS_TO_PARENT) {
392             $stored_mid = $cid;
393             $continue = TRUE;
394           }
395         }
396       }
397     }
398   }
399
400   return $stored_mid;
401 }
```

3.6.3.15 menu_set_location ($ *location*)

Change the current menu location of the user.

Frequently, modules may want to make a page or node act as if it were in the menu tree somewhere, even though it was not registered in a hook_menu() implementation. If the administrator has rearranged the menu, the newly set location should respect this in the breadcrumb trail and expanded/collapsed status of menu items in the tree. This function allows this behavior.

Parameters:

$location An array specifying a complete or partial breadcrumb trail for the new location, in the same format as the return value of hook_menu(). The last element of this array should be the new location itself.

This function will set the new breadcrumb trail to the passed-in value, but if any elements of this trail are visible in the site tree, the trail will be "spliced in" to the existing site navigation at that point.

Definition at line 270 of file menu.inc.

References menu_set_active_item().

Referenced by theme_forum_display().

```
270                                     {
271   global $_menu;
272   $temp_id = min(array_keys($_menu['items'])) - 1;
273   $prev_id = 0;
274
275   foreach (array_reverse($location) as $item) {
276     if (isset($_menu['path index'][$item['path']])) {
277       $mid = $_menu['path index'][$item['path']];
278       if (isset ($_menu['visible'][$mid])) {
279         // Splice in the breadcrumb at this location.
280         if ($prev_id) {
281           $_menu['items'][$prev_id]['pid'] = $mid;
282         }
283         $prev_id = 0;
284         break;
285       }
286       else {
287         // A hidden item; show it, but only temporarily.
288         $_menu['items'][$mid]['type'] |= MENU_VISIBLE_IN_BREADCRUMB;
289         if ($prev_id) {
290           $_menu['items'][$prev_id]['pid'] = $mid;
291         }
292         $prev_id = $mid;
293       }
294     }
295     else {
296       $item['type'] |= MENU_VISIBLE_IN_BREADCRUMB;
297       if ($prev_id) {
298         $_menu['items'][$prev_id]['pid'] = $temp_id;
299       }
300       $_menu['items'][$temp_id] = $item;
301       $_menu['path index'][$item['path']] = $temp_id;
302
303       $prev_id = $temp_id;
304       $temp_id--;
305     }
306   }
307
308   if ($prev_id) {
309     // Didn't find a home, so attach this to the main navigation menu.
310     $_menu['items'][$prev_id]['pid'] = 1;
311   }
312
313   $final_item = array_pop($location);
314   menu_set_active_item($final_item['path']);
315 }
```

3.6.3.16 menu_tree ($*pid* = 1)

Returns a rendered menu tree.

Parameters:

$pid The parent id of the menu.

Definition at line 568 of file menu.inc.

References menu_get_menu(), menu_in_active_trail(), menu_tree(), and theme().

Referenced by menu_tree(), and theme_menu_tree().

```
568                               {
569   $menu = menu_get_menu();
570   $output = '';
571
572   if (isset($menu['visible'][$pid]) && $menu['visible'][$pid]['children']) {
573     foreach ($menu['visible'][$pid]['children'] as $mid) {
574       $output .= theme('menu_item', $mid, menu_in_active_trail($mid) || ($menu['visible'][$mid]['type'
575     }
576   }
577
578   return $output;
579 }
```

3.7 Hooks

Functions

- module_hook ($module, $hook)
- module_implements ($hook)
- module_invoke ()
- module_invoke_all ()

3.7.1 Detailed Description

Allow modules to interact with the Drupal core.

Drupal's module system is based on the concept of "hooks". A hook is a PHP function that is named foo_-bar(), where "foo" is the name of the module (whose filename is thus foo.module) and "bar" is the name of the hook. Each hook has a defined set of parameters and a specified result type.

To extend Drupal, a module need simply implement a hook. When Drupal wishes to allow intervention from modules, it determines which modules implement a hook and call that hook in all enabled modules that implement it.

The available hooks to implement are explained here in the Hooks section of the developer documentation. The string "hook" is used as a placeholder for the module name is the hook definitions. For example, if the module file is called example.module, then hook_help() as implemented by that module would be defined as example_help().

3.7.2 Function Documentation

3.7.2.1 module_hook ($ *module*, $ *hook*)

Determine whether a module implements a hook.

Parameters:

$module The name of the module (without the .module extension).

$hook The name of the hook (e.g. "help" or "menu").

Returns:

TRUE if the module is both installed and enabled, and the hook is implemented in that module.

Definition at line 136 of file module.inc.

Referenced by form_textarea(), menu_get_active_help(), module_implements(), and module_invoke().

```
136                                     {
137   return function_exists($module .'_'. $hook);
138 }
```

3.7.2.2 module_implements ($ *hook*)

Determine which modules are implementing a hook.

Parameters:

$hook The name of the hook (e.g. "help" or "menu").

Returns:

An array with the names of the modules which are implementing this hook.

Definition at line 148 of file module.inc.

References module_hook(), and module_list().

Referenced by module_invoke_all().

```
148                                 {
149   static $implementations;
150
151   if (!isset($implementations[$hook])) {
152     $implementations[$hook] = array();
153     $list = module_list();
154     foreach ($list as $module) {
155       if (module_hook($module, $hook)) {
156         $implementations[$hook][] = $module;
157       }
158     }
159   }
160
161   return $implementations[$hook];
162 }
```

3.7.2.3 module_invoke ()

Invoke a hook in a particular module.

Parameters:

$module The name of the module (without the .module extension).

$hook The name of the hook to invoke.

... Arguments to pass to the hook implementation.

Returns:

The return value of the hook implementation.

Definition at line 176 of file module.inc.

References module_hook().

Referenced by bootstrap_invoke_all(), file_download(), form_textarea(), menu_get_active_help(), node_access(), and theme_blocks().

```
176                         {
177   $args = func_get_args();
178   $module = array_shift($args);
179   $hook = array_shift($args);
```

```
180   $function = $module .'_'. $hook;
181   if (module_hook($module, $hook)) {
182     return call_user_func_array($function, $args);
183   }
184 }
```

3.7.2.4 module_invoke_all ()

Invoke a hook in all enabled modules that implement it.

Parameters:

$hook The name of the hook to invoke.

... Arguments to pass to the hook.

Returns:

An array of return values of the hook implementations. If modules return arrays from their implementations, those are merged into one array.

Definition at line 196 of file module.inc.

References module_implements().

Referenced by _menu_append_contextual_items(), _menu_build(), drupal_goto(), drupal_page_footer(), module_init(), theme_closure(), and theme_onload_attribute().

```
196                             {
197   $args = func_get_args();
198   $hook = array_shift($args);
199   $return = array();
200   foreach (module_implements($hook) as $module) {
201     $function = $module .'_'. $hook;
202     $result = call_user_func_array($function, $args);
203     if (is_array($result)) {
204       $return = array_merge($return, $result);
205     }
206     else if (isset($result)) {
207       $return[] = $result;
208     }
209   }
210
211   return $return;
212 }
```

3.8 Themeable functions

Pager pieces

Use these pieces to construct your own custom pagers in your theme. Note that you should NOT modify this file to customize your pager.

- theme_pager_first ($text, $limit, $element=0, $attributes=array())
- theme_pager_previous ($text, $limit, $element=0, $interval=1, $attributes=array())
- theme_pager_next ($text, $limit, $element=0, $interval=1, $attributes=array())
- theme_pager_last ($text, $limit, $element=0, $attributes=array())
- theme_pager_detail ($limit, $element=0, $format= '%d through%d of%d.')
- theme_pager_list ($limit, $element=0, $quantity=5, $text= '', $attributes=array())

Functions

- theme_menu_tree ($pid=1)
- theme_menu_item ($mid, $children= '', $leaf=TRUE)
- theme_menu_item_link ($item, $link_item)
- theme_menu_local_tasks ()
- theme_menu_local_task ($mid, $active, $primary)
- theme_pager ($tags=array(), $limit=10, $element=0, $attributes=array())
- theme_placeholder ($text)
- theme_page ($content)
- theme_status_messages ()
- theme_links ($links, $delimiter= '| ')
- theme_image ($path, $alt= '', $title= '', $attr= '', $getsize=true)
- theme_breadcrumb ($breadcrumb)
- theme_node ($node, $teaser=FALSE, $page=FALSE)
- theme_form_element ($title, $value, $description=NULL, $id=NULL, $required=FALSE, $error=FALSE)
- theme_submenu ($links)
- theme_table ($header, $rows, $attributes=NULL)
- theme_box ($title, $content, $region= 'main')
- theme_block ($block)
- theme_mark ($type=MARK_NEW)
- theme_stylesheet_import ($stylesheet, $media= 'all')
- theme_item_list ($items=array(), $title=NULL)
- theme_error ($message)
- **theme_more_help_link** ($url)
- theme_xml_icon ($url)
- theme_closure ($main=0)
- theme_onload_attribute ($theme_onloads=array())
- theme_blocks ($region)
- theme_confirm ($question, $path, $description=NULL, $yes=NULL, $no=NULL, $extra=NULL, $name= 'confirm')
- function theme_aggregator_feed ($feed)

- function theme_aggregator_block_item ($item, $feed=0)
- function theme_aggregator_summary_item ($item)
- function theme_aggregator_page_item ($item)
- function theme_filter_tips ($tips, $long=false, $extra= ”)
- function theme_forum_display ($forums, $topics, $parents, $tid, $sortby, $forum_per_page)
- function theme_forum_list ($forums, $parents, $tid)
- function theme_forum_topic_list ($tid, $topics, $sortby, $forum_per_page)
- function theme_search_item ($item, $type)

3.8.1 Detailed Description

Functions that display HTML, and which can be customized by themes.

All functions that produce HTML for display should be themeable. This means that they should be named with the theme_ prefix, and invoked using theme() rather than being called directly. This allows themes to override the display of any Drupal object.

The theme system is described and defined in theme.inc.

3.8.2 Function Documentation

3.8.2.1 function theme_aggregator_block_item ($ *item*, $ *feed* = 0)

Format an individual feed item for display in the block.

Definition at line 1060 of file aggregator.module.

References check_plain(), l(), module_exist(), and theme().

```
1060                                                    {
1061   global $user;
1062
1063   if ($user->uid && module_exist('blog') && user_access('edit own blog')) {
1064     if ($image = theme('image', 'misc/blog.png', t('blog it'), t('blog it'))) {
1065       $output .= '<div class="icon">'. l($image, 'node/add/blog', array('title' => t('Comment on this
1066     }
1067   }
1068
1069   // Display the external link to the item.
1070   $output .= '<a href="'. check_url($item->link) .'">'. check_plain($item->title) ."</a>\n";
1071
1072   return $output;
1073 }
```

3.8.2.2 function theme_aggregator_feed ($ *feed*)

Format a news feed.

Definition at line 1031 of file aggregator.module.

References check_plain(), format_interval(), l(), and theme().

```
1031                                        {
1032   $output  = '';
1033
1034   if ($feed->image) {
1035     $output .= $feed->image;
1036   }
1037
1038   $output .= $feed->description;
1039   $output .= '<h3>'. t('URL') ."</h3>\n";
1040   $output .= theme('xml_icon', $feed->url);
1041   $output .= '<a href="'. check_url($feed->link) .'">'. check_plain($feed->link) ."</a>\n";
1042   $output .= '<h3>'. t('Last update') ."</h3>\n";
1043   $updated = t('%time ago', array('%time' => format_interval(time() - $feed->checked)));
1044
1045   if (user_access('administer news feeds')) {
1046     $output .= l($updated, 'admin/aggregator');
1047   }
1048   else {
1049     $output .= $updated;
1050   }
1051
1052   return $output;
1053 }
```

3.8.2.3 function theme_aggregator_page_item ($ *item*)

Format an individual feed item for display on the aggregator page.

Definition at line 1097 of file aggregator.module.

References check_plain(), db_query(), format_date(), and l().

```
1097                                            {
1098   static $last;
1099
1100   $date = format_date($item->timestamp, 'custom', 'Ymd');
1101   if ($date != $last) {
1102     $last = $date;
1103     $output .= '<h3>'. format_date($item->timestamp, 'custom', 'F j, Y') ."</h3>\n";
1104   }
1105
1106   $output .= "<div class=\"news-item\">\n";
1107   $output .= ' <div class="date">'. format_date($item->timestamp, 'custom', 'H:i') ."</div>\n";
1108   $output .= " <div class=\"body\">\n";
1109   $output .= '  <div class="title"><a href="'. check_url($item->link) .'">'. check_plain($item->title
1110   if ($item->description) {
1111     $output .= '  <div class="description">'. $item->description ."</div>\n";
1112   }
1113   if ($item->ftitle && $item->fid) {
1114     $output .= '  <div class="source">'. t('Source') .': '. l($item->ftitle, "aggregator/sources/$ite
1115   }
1116
1117   $result = db_query('SELECT c.title, c.cid FROM {aggregator_category_item} ci LEFT JOIN {aggregator_
1118   $categories = array();
1119   while ($category = db_fetch_object($result)) {
1120     $categories[] = l($category->title, 'aggregator/categories/'. $category->cid);
1121   }
1122   if ($categories) {
1123     $output .= '  <div class="categories">'. t('Categories') .': '. implode(', ', $categories) ."</di
1124   }
1125
1126   $output .= " </div>\n";
1127   $output .= "</div>\n";
```

```
1128
1129   return $output;
1130 }
```

3.8.2.4 function theme_aggregator_summary_item ($ *item*)

Return a themed item heading for summary pages located at "aggregator/sources" and "aggregator/categories".

Parameters:

$item The item object from the aggregator module.

Returns:

A string containing the output.

Definition at line 1084 of file aggregator.module.

References check_plain(), and format_interval().

```
1084                                                {
1085   $output = '<a href="'. check_url($item->link) .'">'. check_plain($item->title) .'</a> <span class="
1086   if ($item->feed_link) {
1087     $output .= ', <span class="source"><a href="'. $item->feed_link .'">'. $item->feed_title .'</a></
1088   }
1089   return $output ."\n";
1090 }
```

3.8.2.5 theme_block ($ *block*)

Return a themed block.

You can style your blocks by defining .block (all blocks), .block-*module* (all blocks of module *module*), and #block-*module*-*delta* (specific block of module *module* with delta *delta*) in your theme's CSS.

Parameters:

$block An object populated with fields from the "blocks" database table ($block->module, $block->delta, $block->region, ...) and fields returned by *module*_block('view') ($block->subject, $block->content, ...).

Returns:

A string containing the block output.

Definition at line 722 of file theme.inc.

```
722                               {
723   $output  = "<div class=\"block block-$block->module\" id=\"block-$block->module-$block->delta\">\n";
724   $output .= " <h2 class=\"title\">$block->subject</h2>\n";
725   $output .= " <div class=\"content\">$block->content</div>\n";
726   $output .= "</div>\n";
727   return $output;
728 }
```

3.8.2.6 theme_blocks ($ *region*)

Return a set of blocks available for the current user.

Parameters:

$region Which set of blocks to retrieve.

Returns:

A string containing the themed blocks for this region.

Definition at line 861 of file theme.inc.

References module_invoke(), and theme().

```
861                                 {
862   $output = '';
863
864   if ($list = module_invoke('block', 'list', $region)) {
865     foreach ($list as $key => $block) {
866       // $key == <i>module</i>_<i>delta</i>
867       $output .= theme('block', $block);
868     }
869   }
870   return $output;
871 }
```

3.8.2.7 theme_box ($ *title*, $ *content*, $ *region* = 'main')

Return a themed box.

Parameters:

$title The subject of the box.

$content The content of the box.

$region The region in which the box is displayed.

Returns:

A string containing the box output.

Definition at line 702 of file theme.inc.

```
702                                                     {
703   $output = '<h2 class="title">'. $title .'</h2><div>'. $content .'</div>';
704   return $output;
705 }
```

3.8.2.8 theme_breadcrumb ($ *breadcrumb*)

Return a themed breadcrumb trail.

Parameters:

$breadcrumb An array containing the breadcrumb links.

Returns:
a string containing the breadcrumb output.

Definition at line 487 of file theme.inc.

```
487                                       {
488   return '<div class="breadcrumb">'. implode($breadcrumb, ' &raquo; ') .'</div>';
489 }
```

3.8.2.9 theme_closure ($ *main* = 0)

Execute hook_footer() which is run at the end of the page right before the close of the body tag.

Parameters:
$main (optional)

Returns:
A string containing the results of the hook_footer() calls.

Definition at line 826 of file theme.inc.

References module_invoke_all().

```
826                                     {
827   $footer = module_invoke_all('footer', $main);
828   return implode($footer, "\n");
829 }
```

3.8.2.10 theme_confirm ($ *question*, $ *path*, $ *description* = NULL, $ *yes* = NULL, $ *no* = NULL, $ *extra* = NULL, $ *name* = 'confirm')

Output a confirmation form

This function outputs a complete form for confirming an action. A link is offered to go back to the item that is being changed in case the user changes his/her mind.

You should use $_POST['edit'][$name] (where $name is usually 'confirm') to check if the confirmation was successful.

Parameters:
$question The question to ask the user (e.g. "Are you sure you want to delete the block *foo*?").

$path The page to go to if the user denies the action.

$description Additional text to display (defaults to "This action cannot be undone.").

$yes A caption for the button which confirms the action (e.g. "Delete", "Replace", ...).

$no A caption for the link which denies the action (e.g. "Cancel").

$extra Additional HTML to inject into the form, for example form_hidden()s.

$name The internal name used to refer to the confirmation item.

Returns:
A themed HTML string representing the form.

Definition at line 902 of file theme.inc.

References drupal_set_title(), form(), form_hidden(), form_submit(), and l().

```
902
903   drupal_set_title($question);
904
905   if (is_null($description)) {
906     $description = t('This action cannot be undone.');
907   }
908
909   $output .= '<p>'. $description ."</p>\n";
910   if (!is_null($extra)) {
911     $output .= $extra;
912   }
913   $output .= '<div class="container-inline">';
914   $output .= form_submit($yes ? $yes : t('Confirm'));
915   $output .= l($no ? $no : t('Cancel'), $path);
916   $output .= "</div>\n";
917
918   $output .= form_hidden($name, 1);
919   return form($output, 'post', NULL, array('class' => 'confirmation'));
920 }
```

3.8.2.11 theme_error ($ *message*)

Return a themed error message. REMOVE: this function is deprecated an no longer used in core.

Parameters:
$message The error message to be themed.

Returns:
A string containing the error output.

Definition at line 800 of file theme.inc.

```
800                                 {
801   return '<div class="error">'. $message .'</div>';
802 }
```

3.8.2.12 function theme_filter_tips ($ *tips*, $ *long* = false, $ *extra* = '')

Format a set of filter tips.

Definition at line 824 of file filter.module.

```
824                                                                 {
825   $output = '';
826
827   $multiple = count($tips) > 1;
```

```
828   if ($multiple) {
829     $output = t('Input formats') .':';
830   }
831
832   if (count($tips)) {
833     if ($multiple) {
834       $output .= '<ul>';
835     }
836     foreach ($tips as $name => $tiplist) {
837       if ($multiple) {
838         $output .= '<li>';
839         $output .= '<strong>'. $name .'</strong>:<br />';
840       }
841
842       $tips = '';
843       foreach ($tiplist as $tip) {
844         $tips .= '<li'. ($long ? ' id="filter-'. str_replace("/", "-", $tip['id']) .'">' : '>') . $tip
845       }
846
847       if ($tips) {
848         $output .= "<ul class=\"tips\">$tips</ul>";
849       }
850
851       if ($multiple) {
852         $output .= '</li>';
853       }
854     }
855     if ($multiple) {
856       $output .= '</ul>';
857     }
858   }
859
860   return $output;
861 }
```

3.8.2.13 theme_form_element ($ *title*, $ *value*, $ *description* = NULL, $ *id* = NULL, $ *required* = FALSE, $ *error* = FALSE)

Return a themed form element.

Parameters:

$title the form element's title

$value the form element's data

$description the form element's description or explanation

$id the form element's ID used by the <label> tag

$required a boolean to indicate whether this is a required field or not

$error a string with an error message filed against this form element

Returns:

a string representing the form element

Definition at line 555 of file theme.inc.

```
555
556
557   $output  = "<div class=\"form-item\">\n";
```

```
558   $required = $required ? '<span class="form-required">*</span>' : '';
559
560   if ($title) {
561     if ($id) {
562       $output .= " <label for=\"$id\">$title:</label>$required<br />\n";
563     }
564     else {
565       $output .= " <label>$title:</label>$required<br />\n";
566     }
567   }
568
569   $output .= " $value\n";
570
571   if ($description) {
572     $output .= " <div class=\"description\">$description</div>\n";
573   }
574
575   $output .= "</div>\n";
576
577   return $output;
578 }
```

3.8.2.14 function theme_forum_display ($ *forums*, $ *topics*, $ *parents*, $ *tid*, $ *sortby*, $ *forum_per_page*)

Format the forum body.

Definition at line 752 of file forum.module.

References drupal_set_title(), l(), menu_set_location(), module_exist(), theme(), url(), and variable_get().

```
752                                                                                                                {
753   global $user;
754   // forum list, topics list, topic browser and 'add new topic' link
755
756   $vocabulary = taxonomy_get_vocabulary(variable_get('forum_nav_vocabulary', ''));
757   $title = $vocabulary->name;
758
759   // Breadcrumb navigation:
760   $breadcrumb = array();
761   if ($tid) {
762     $breadcrumb[] = array('path' => 'forum', 'title' => $title);
763   }
764
765   if ($parents) {
766     $parents = array_reverse($parents);
767     foreach ($parents as $p) {
768       if ($p->tid == $tid) {
769         $title = $p->name;
770       }
771       else {
772         $breadcrumb[] = array('path' => 'forum/'. $p->tid, 'title' => $p->name);
773       }
774     }
775   }
776
777   drupal_set_title($title);
778
779   $breadcrumb[] = array('path' => $_GET['q']);
780   menu_set_location($breadcrumb);
781
782   if (count($forums) || count($parents)) {
```

```
783     $output  = '<div id="forum">';
784     $output .= '<ul>';
785
786     if (module_exist('tracker')) {
787       if ($user->uid) {
788         $output .= ' <li>'. l(t('My forum discussions.'), "tracker/$user->uid") .'</li>';
789       }
790
791       $output .= ' <li>'. l(t('Active forum discussions.'), 'tracker') .'</li>';
792     }
793
794     if (user_access('create forum topics')) {
795       $output .= '<li>'. l(t('Post new forum topic.'), "node/add/forum/$tid") .'</li>';
796     }
797     else if ($user->uid) {
798       $output .= '<li>'. t('You are not allowed to post a new forum topic.') .'</li>';
799     }
800     else {
801       $output .= '<li>'. t('<a href="%login">Login</a> to post a new forum topic.', array('%login' =>
802     }
803     $output .= '</ul>';
804
805     $output .= theme('forum_list', $forums, $parents, $tid);
806
807     if ($tid && !in_array($tid, variable_get('forum_containers', array()))) {
808       drupal_add_link(array('rel' => 'alternate',
809                             'type' => 'application/rss+xml',
810                             'title' => 'RSS - '. $title,
811                             'href' => url('taxonomy/term/'. $tid .'/0/feed')));
812
813       $output .= theme('forum_topic_list', $tid, $topics, $sortby, $forum_per_page);
814       $output .= theme('xml_icon', url("taxonomy/term/$tid/0/feed"));
815     }
816     $output .= '</div>';
817   }
818   else {
819     drupal_set_title(t('No forums defined'));
820     $output = '';
821   }
822
823   return $output;
824 }
```

3.8.2.15 function theme_forum_list ($ *forums*, $ *parents*, $ *tid*)

Format the forum listing.

Definition at line 831 of file forum.module.

References check_plain(), l(), and theme().

```
831                                                 {
832   global $user;
833
834   if ($forums) {
835
836     $header = array(t('Forum'), t('Topics'), t('Posts'), t('Last post'));
837
838     foreach ($forums as $forum) {
839       if ($forum->container) {
840         $description  = '<div style="margin-left: '. ($forum->depth * 30) ."px;\">\n";
841         $description .= ' <div class="name">'. l($forum->name, "forum/$forum->tid") ."</div>\n";
842
```

```
843         if ($forum->description) {
844           $description .= ' <div class="description">'. check_plain($forum->description) ."</div>\n";
845         }
846         $description .= "</div>\n";
847 
848         $rows[] = array(array('data' => $description, 'class' => 'container', 'colspan' => '4'));
849       }
850       else {
851         $forum->old_topics = _forum_topics_read($forum->tid, $user->uid);
852         if ($user->uid) {
853           $new_topics = $forum->num_topics - $forum->old_topics;
854         }
855         else {
856           $new_topics = 0;
857         }
858 
859         $description  = '<div style="margin-left: '. ($forum->depth * 30) ."px;\">\n";
860         $description .= ' <div class="name">'. l($forum->name, "forum/$forum->tid") ."</div>\n";
861 
862         if ($forum->description) {
863           $description .= ' <div class="description">'. check_plain($forum->description) ."</div>\n";
864         }
865         $description .= "</div>\n";
866 
867         $rows[] = array(
868           array('data' => $description, 'class' => 'forum'),
869           array('data' => $forum->num_topics . ($new_topics ? '<br />'. l(t('%a new', array('%a' => $n
870           array('data' => $forum->num_posts, 'class' => 'posts'),
871           array('data' => _forum_format($forum->last_post), 'class' => 'last-reply'));
872       }
873     }
874 
875     return theme('table', $header, $rows);
876 
877   }
878 
879 }
```

3.8.2.16 function theme_forum_topic_list ($ *tid*, $ *topics*, $ *sortby*, $ *forum_per_page*)

Format the topic listing.

Definition at line 886 of file forum.module.

References check_plain(), l(), tablesort_pager(), and theme().

```
886                                                                                 {
887   global $forum_topic_list_header;
888 
889   if ($topics) {
890 
891     foreach ($topics as $topic) {
892       // folder is new if topic is new or there are new comments since last visit
893       if ($topic->tid != $tid) {
894         $rows[] = array(
895           array('data' => _forum_icon($topic->new, $topic->num_comments, $topic->comment_mode, $topic-
896           array('data' => check_plain($topic->title), 'class' => 'title'),
897           array('data' => l(t('This topic has been moved'), "forum/$topic->tid"), 'colspan' => '3')
898         );
899       }
900       else {
901         $rows[] = array(
902           array('data' => _forum_icon($topic->new, $topic->num_comments, $topic->comment_mode, $topic-
```

```
903           array('data' => l($topic->title, "node/$topic->nid"), 'class' => 'topic'),
904           array('data' => $topic->num_comments . ($topic->new_replies ? '<br />'. l(t('%a new', array(
905           array('data' => _forum_format($topic), 'class' => 'created'),
906           array('data' => _forum_format($topic->last_reply), 'class' => 'last-reply')
907         );
908       }
909     }
910
911     if ($pager = theme('pager', NULL, $forum_per_page, 0, tablesort_pager())) {
912       $rows[] = array(array('data' => $pager, 'colspan' => '5', 'class' => 'pager'));
913     }
914   }
915
916   $output .= theme('table', $forum_topic_list_header, $rows);
917
918   return $output;
919 }
```

3.8.2.17 theme_image ($ *path*, $ *alt* = '', $ *title* = '', $ *attr* = '', $ *getsize* = true)

Return a themed image.

Parameters:

$path The path of the image file.

$alt The alternative text for text-based browsers.

$title The title text is displayed when the image is hovered in some popular browsers.

$attr Attributes placed in the img tag.

$getsize If set to true, the image's dimension are fetched and added as width/height attributes.

Returns:

A string containing the image tag.

Definition at line 474 of file theme.inc.

```
474                                                                                 {
475   if (!$getsize || (is_file($path) && (list($width, $height, $type, $image_attributes) = @getimagesize
476     return '<img src="'. check_url($path) .'" '. $attr .' alt="'. check_plain($alt) .'" title="'. chec
477   }
478 }
```

3.8.2.18 theme_item_list ($ *items* = array(), $ *title* = NULL)

Return a themed list of items.

Parameters:

$items An array of items to be displayed in the list.

$title The title of the list.

Returns:

A string containing the list output.

Definition at line 773 of file theme.inc.

```
773                                                                 {
774   $output = '<div class="item-list">';
775   if (isset($title)) {
776     $output .= '<h3>'. $title .'</h3>';
777   }
778
779   if (isset($items)) {
780     $output .= '<ul>';
781     foreach ($items as $item) {
782       $output .= '<li>'. $item .'</li>';
783     }
784     $output .= '</ul>';
785   }
786   $output .= '</div>';
787   return $output;
788 }
```

3.8.2.19 theme_links ($ *links*, $ *delimiter* = ' | ')

Return a themed set of links.

Parameters:

$links An array of links to be themed.

$delimiter A string used to separate the links.

Returns:

A string containing the themed links.

Definition at line 454 of file theme.inc.

```
454                                                 {
455   return implode($delimiter, $links);
456 }
```

3.8.2.20 theme_mark ($ *type* = MARK_NEW)

Return a themed marker, useful for marking new or updated content.

Parameters:

$type Number representing the marker type to display

See also:

MARK_NEW, MARK_UPDATED, MARK_READ

Returns:

A string containing the marker.

Definition at line 740 of file theme.inc.

```
740                                     {
741   global $user;
742   if ($user->uid && $type != MARK_READ) {
743     return '<span class="marker">*</span>';
744   }
745 }
```

3.8.2.21 theme_menu_item ($ *mid*, $ *children* = '', $ *leaf* = TRUE)

Generate the HTML output for a single menu item.

Parameters:

$mid The menu id of the item.

$children A string containing any rendered child items of this menu.

$leaf A boolean indicating whether this menu item is a leaf.

Definition at line 593 of file menu.inc.

References menu_item_link().

```
593                                                         {
594   return '<li class="'. ($leaf ? 'leaf' : ($children ? 'expanded' : 'collapsed')) .'">'. menu_item_lin
595 }
```

3.8.2.22 theme_menu_item_link ($ *item*, $ *link_item*)

Generate the HTML representing a given menu item ID.

Parameters:

$item The menu item to render.

$link_mid The menu item which should be used to find the correct path.

Definition at line 607 of file menu.inc.

References l().

```
607                                                 {
608   return l($item['title'], $link_item['path'], array_key_exists('description', $item) ? array('title'
609 }
```

3.8.2.23 theme_menu_local_task ($ *mid*, $ *active*, $ *primary*)

Generate the HTML representing a given menu item ID as a tab.

Parameters:

$mid The menu ID to render.

$active Whether this tab or a subtab is the active menu item.

$primary Whether this tab is a primary tab or a subtab.

Definition at line 697 of file menu.inc.

References menu_item_link().

```
697                                                          {
698   if ($active) {
699     return '<li class="active">'. menu_item_link($mid) ."</li>\n";
700   }
701   else {
702     return '<li>'. menu_item_link($mid) ."</li>\n";
703   }
704 }
```

3.8.2.24 theme_menu_local_tasks ()

Returns the rendered local tasks. The default implementation renders them as tabs.

Definition at line 634 of file menu.inc.

References menu_primary_local_tasks(), and menu_secondary_local_tasks().

```
634                                    {
635   $output = '';
636
637   if ($primary = menu_primary_local_tasks()) {
638     $output .= "<ul class=\"tabs primary\">\n". $primary ."</ul>\n";
639   }
640   if ($secondary = menu_secondary_local_tasks()) {
641     $output .= "<ul class=\"tabs secondary\">\n". $secondary ."</ul>\n";
642   }
643
644   return $output;
645 }
```

3.8.2.25 theme_menu_tree ($ *pid* = 1)

Generate the HTML for a menu tree.

Parameters:

$pid The parent id of the menu.

Definition at line 556 of file menu.inc.

References menu_tree().

```
556                                     {
557   if ($tree = menu_tree($pid)) {
558     return "\n<ul>\n". $tree ."\n</ul>\n";
559   }
560 }
```

3.8.2.26 theme_node ($ *node*, $ *teaser* = FALSE, $ *page* = FALSE)

Return a themed node.

Parameters:

$node An object providing all relevant information for displaying a node:

- $node->nid: The ID of the node.
- $node->type: The content type (story, blog, forum...).
- $node->title: The title of the node.
- $node->created: The creation date, as a UNIX timestamp.
- $node->teaser: A shortened version of the node body.
- $node->body: The entire node contents.
- $node->changed: The last modification date, as a UNIX timestamp.
- $node->uid: The ID of the author.
- $node->username: The username of the author.

$teaser Whether to display the teaser only, as on the main page.

$page Whether to display the node as a standalone page. If TRUE, do not display the title because it will be provided by the menu system.

Returns:

A string containing the node output.

Definition at line 513 of file theme.inc.

References check_plain(), format_name(), module_exist(), and theme().

```
513                                                                 {
514   if (module_exist('taxonomy')) {
515     $terms = taxonomy_link('taxonomy terms', $node);
516   }
517
518   if ($page == 0) {
519     $output = '<h2 class="title">'. check_plain($node->title) .'</h2> by '. format_name($node);
520   }
521   else {
522     $output = 'by '. format_name($node);
523   }
524
525   if (count($terms)) {
526     $output .= ' <small>('. theme('links', $terms) .')</small><br />';
527   }
528
529   if ($teaser && $node->teaser) {
530     $output .= $node->teaser;
531   }
532   else {
533     $output .= $node->body;
534   }
535
536   if ($node->links) {
537     $output .= '<div class="links">'. theme('links', $node->links) .'</div>';
538   }
539
540   return $output;
541 }
```

3.8.2.27 theme_onload_attribute ($ *theme_onloads* = array())

Call hook_onload() in all modules to enable modules to insert JavaScript that will get run once the page has been loaded by the browser.

Parameters:

$theme_onloads Additional onload directives.

Returns:

A string containing the onload attributes.

Definition at line 840 of file theme.inc.

References module_invoke_all().

```
840                                                         {
841   if (!is_array($theme_onloads)) {
842     $theme_onloads = array($theme_onloads);
843   }
844   // Merge theme onloads (javascript rollovers, image preloads, etc.)
845   // with module onloads (htmlarea, etc.)
846   $onloads = array_merge(module_invoke_all('onload'), $theme_onloads);
847   if (count($onloads)) {
848     return ' onload="' . implode('; ', $onloads) . '"';
849   }
850   return '';
851 }
```

3.8.2.28 theme_page ($ *content*)

Return an entire Drupal page displaying the supplied content.

Parameters:

$content A string to display in the main content area of the page.

Returns:

A string containing the entire HTML page.

Definition at line 376 of file theme.inc.

References drupal_get_html_head(), drupal_get_title(), theme(), theme_get_styles(), and variable_get().

```
376                                 {
377   $output = "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\" \"http://www.w3.org/TR/x
378   $output .= '<html xmlns="http://www.w3.org/1999/xhtml">';
379   $output .= '<head>';
380   $output .= ' <title>'. (drupal_get_title() ? strip_tags(drupal_get_title()) : variable_get('site_nam
381   $output .= drupal_get_html_head();
382   $output .= theme_get_styles();
383
384   $output .= ' </head>';
385   $output .= ' <body style="background-color: #fff; color: #000;"'. theme('onload_attribute'). '">';
386   $output .= '<table border="0" cellspacing="4" cellpadding="4"><tr><td style="vertical-align: top; wi
387
388   $output .= theme('blocks', 'all');
389   $output .= '</td><td style="vertical-align: top;">';
```

```
390
391   $output .= theme('breadcrumb', drupal_get_breadcrumb());
392   $output .= '<h1>' . drupal_get_title() . '</h1>';
393
394   if ($tabs = theme('menu_local_tasks')) {
395    $output .= $tabs;
396   }
397
398   if ($help = menu_get_active_help()) {
399     $output .= '<small>'. $help .'</small><hr />';
400   }
401
402   $output .= theme_status_messages();
403
404   $output .= "\n<!-- begin content -->\n";
405   $output .= $content;
406   $output .= "\n<!-- end content -->\n";
407
408   $output .= '</td></tr></table>';
409   $output .= theme_closure();
410   $output .= '</body></html>';
411
412   return $output;
413 }
```

3.8.2.29 theme_pager ($ *tags* = array(), $ *limit* = 10, $ *element* = 0, $ *attributes* = array())

Format a query pager.

Menu callbacks that display paged query results should call theme('pager') to retrieve a pager control so that users can view other results.

Parameters:

$tags An array of labels for the controls in the pager.

$limit The number of query results to display per page.

$element An optional integer to distinguish between multiple pagers on one page.

$attributes An associative array of query string parameters to append to the pager links.

Returns:

An HTML string that generates the query pager.

Definition at line 100 of file pager.inc.

References theme().

```
100                                                                                                                 {
101   global $pager_total;
102   $output = '';
103
104   if ($pager_total[$element] > $limit) {
105     $output .= '<div id="pager" class="container-inline">';
106     $output .= theme('pager_first', ($tags[0] ? $tags[0] : t('first page')), $limit, $element, $attrib
107     $output .= theme('pager_previous', ($tags[1] ? $tags[1] : t('previous page')), $limit, $element, 1
108     $output .= theme('pager_list', $limit, $element, ($tags[2] ? $tags[2] : 9 ), '', $attributes);
109     $output .= theme('pager_next', ($tags[3] ? $tags[3] : t('next page')), $limit, $element, 1, $attri
110     $output .= theme('pager_last', ($tags[4] ? $tags[4] : t('last page')), $limit, $element, $attribut
111     $output .= '</div>';
112
```

```
113     return $output;
114   }
115 }
```

3.8.2.30 theme_pager_detail ($ *limit*, $ *element* = 0, $ *format* = '%d through %d of %d.')

Format a summary of the current pager position, such as "6 through 10 of 52".

Parameters:

$limit The number of query results to display per page.

$element An optional integer to distinguish between multiple pagers on one page.

$format A printf-style format string for customizing the pager text.

Returns:

An HTML string that generates this piece of the query pager.

Definition at line 267 of file pager.inc.

```
267                                                                                                 {
268   global $pager_from_array, $pager_total;
269
270   $output = '<div class="pager-detail">';
271   if ($pager_total[$element] > (int)$pager_from_array[$element] + 1) {
272     $output .= sprintf($format, (int)$pager_from_array[$element] + 1, ((int)$pager_from_array[$element
273   }
274   $output .= '</div>';
275
276   return $output;
277 }
```

3.8.2.31 theme_pager_first ($ *text*, $ *limit*, $ *element* = 0, $ *attributes* = array())

Format a "first page" link.

Parameters:

$text The name (or image) of the link.

$limit The number of query results to display per page.

$element An optional integer to distinguish between multiple pagers on one page.

$attributes An associative array of query string parameters to append to the pager links.

Returns:

An HTML string that generates this piece of the query pager.

Definition at line 140 of file pager.inc.

```
140                                                                                 {
141   global $pager_from_array;
142   $output = '<div class="pager-first">';
```

```
143
144   if ($pager_from_array[$element]) {
145     $output .= '<a href="'. pager_link(pager_load_array(0, $element, $pager_from_array), $element, $at
146   }
147   else {
148     $output .= ' ';
149   }
150   $output .= '</div>';
151   return $output;
152 }
```

3.8.2.32 theme_pager_last ($ *text*, $ *limit*, $ *element* = 0, $ *attributes* = array())

Format a "last page" link.

Parameters:

$text The name (or image) of the link.

$limit The number of query results to display per page.

$element An optional integer to distinguish between multiple pagers on one page.

$attributes An associative array of query string parameters to append to the pager links.

Returns:

An HTML string that generates this piece of the query pager.

Definition at line 234 of file pager.inc.

References theme().

```
234                                                                                 {
235   global $pager_from_array, $pager_total;
236
237   $output = '<div class="pager-last">';
238   $last_num = (($pager_total[$element] % $limit) ? ($pager_total[$element] % $limit) : $limit);
239   $from_new = pager_load_array(($pager_total[$element] - $last_num), $element, $pager_from_array);
240   if ($from_new[$element] < ($pager_from_array[$element] + $limit)) {
241     $output .= theme('pager_next', $text, $limit, $element, 1, $attributes);
242   }
243   else if (($from_new[$element] > $pager_from_array[$element]) && ($from_new[$element] > 0) && ($from_
244     $output .= '<a href="'. pager_link($from_new, $element, $attributes) .'">'. $text .'</a>';
245   }
246   else {
247     $output .= ' ';
248   }
249   $output .= '</div>';
250   return $output;
251 }
```

3.8.2.33 theme_pager_list ($ *limit*, $ *element* = 0, $ *quantity* = 5, $ *text* = '', $ *attributes* = array())

Format a list of nearby pages with additional query results.

Parameters:

$limit The number of query results to display per page.

$element An optional integer to distinguish between multiple pagers on one page.

$quantity The number of pages in the list.

$text A string of text to display before the page list.

$attributes An associative array of query string parameters to append to the pager links.

Returns:

An HTML string that generates this piece of the query pager.

Definition at line 297 of file pager.inc.

References theme().

```
297                                                                                                                 {
298   global $pager_from_array, $pager_total;
299
300   $output = '<div class="pager-list">';
301   // Calculate various markers within this pager piece:
302   // Middle is used to "center" pages around the current page.
303   $pager_middle = ceil((int)$quantity / 2);
304   // offset adds "offset" second page
305   $pager_offset = (int)$pager_from_array[$element] % (int)$limit;
306   // current is the page we are currently paged to
307   if (($pager_current = (ceil(($pager_from_array[$element] + 1) / $limit))) < 1) {
308     $pager_current = 1;
309   }
310   // first is the first page listed by this pager piece (re quantity)
311   $pager_first = (int)$pager_current - (int)$pager_middle + 1;
312   // last is the last page listed by this pager piece (re quantity)
313   $pager_last = (int)$pager_current + (int)$quantity - (int)$pager_middle;
314   // max is the maximum number of pages content can is divided into
315   if (!$pager_max = (ceil($pager_total[$element] / $limit))) {
316     $pager_max = 1;
317   }
318   if ((int)$pager_offset) {
319     // adjust for offset second page
320     $pager_max++;
321     $pager_current++;
322   }
323   // End of marker calculations.
324
325   // Prepare for generation loop.
326   $i = (int)$pager_first;
327   if ($pager_last > $pager_max) {
328     // Adjust "center" if at end of query.
329     $i = $i + (int)($pager_max - $pager_last);
330     $pager_last = $pager_max;
331   }
332   if ($i <= 0) {
333     // Adjust "center" if at start of query.
334     $pager_last = $pager_last + (1 - $i);
335     $i = 1;
336   }
337   // End of generation loop preparation.
338
339   // When there is more than one page, create the pager list.
340   if ($i != $pager_max) {
341     $output .= $text;
342     if ($i > 1) {
343       $output .= '<div class="pager-list-dots-left">... </div>';
344     }
345
346     // Now generate the actual pager piece.
347     for (; $i <= $pager_last && $i <= $pager_max; $i++) {
348       if ($i < $pager_current) {
```

```
349         $output .= theme('pager_previous', $i, $limit, $element, ($pager_current - $i), $attributes) .
350       }
351       if ($i == $pager_current) {
352         $output .= '<strong>'. $i .'</strong> ';
353       }
354       if ($i > $pager_current) {
355         $output .= theme('pager_next', $i, $limit, $element, ($i - $pager_current), $attributes) ." ";
356       }
357     }
358
359     if ($i < $pager_max) {
360       $output .= '<div class="pager-list-dots-right">...</div>';
361     }
362   }
363   $output .= '</div>';
364
365   return $output;
366 }
```

3.8.2.34 theme_pager_next ($ *text*, $ *limit*, $ *element* = 0, $ *interval* = 1, $ *attributes* = array())

Format a "next page" link.

Parameters:

$text The name (or image) of the link.

$limit The number of query results to display per page.

$element An optional integer to distinguish between multiple pagers on one page.

$interval The number of pages to move forward when the link is clicked.

$attributes An associative array of query string parameters to append to the pager links.

Returns:

An HTML string that generates this piece of the query pager.

Definition at line 204 of file pager.inc.

```
204                                                                                                 {
205   global $pager_from_array, $pager_total;
206   $output = '<div class="pager-next">';
207   $from_new = pager_load_array(((int)$pager_from_array[$element] + ((int)$limit * (int)$interval)), $e
208   if ($from_new[$element] < $pager_total[$element]) {
209     $output .= '<a href="'. pager_link($from_new, $element, $attributes) .'">'. $text .'</a>';
210   }
211   else {
212     $output .= ' ';
213   }
214   $output .= '</div>';
215   return $output;
216 }
```

3.8.2.35 theme_pager_previous ($ *text*, $ *limit*, $ *element* = 0, $ *interval* = 1, $ *attributes* = array())

Format a "previous page" link.

Parameters:

$text The name (or image) of the link.

$limit The number of query results to display per page.

$element An optional integer to distinguish between multiple pagers on one page.

$interval The number of pages to move backward when the link is clicked.

$attributes An associative array of query string parameters to append to the pager links.

Returns:

An HTML string that generates this piece of the query pager.

Definition at line 172 of file pager.inc.

References theme().

```
172                                                                                                                 {
173   global $pager_from_array;
174   $output = '<div class="pager-previous">';
175   $from_new = pager_load_array(((int)$pager_from_array[$element] - ((int)$limit * (int)$interval)), $e
176   if ($from_new[$element] < 1) {
177     $output .= theme('pager_first', $text, $limit, $element, $attributes);
178   }
179   else {
180     $output .= '<a href="'. pager_link($from_new, $element, $attributes) .'">'. $text .'</a>';
181   }
182   $output .= '</div>';
183   return $output;
184 }
```

3.8.2.36 theme_placeholder ($ *text*)

Format a dynamic text string for emphasised display in a placeholder.

E.g. t('Added term term', array('%term' => theme('placeholder', $term)))

Parameters:

$text The text to format (plain-text).

Returns:

The formatted text (html).

Definition at line 364 of file theme.inc.

References check_plain().

```
364                                  {
365   return '<em>'. check_plain($text) .'</em>';
366 }
```

3.8.2.37 function theme_search_item ($ *item*, $ *type*)

Format a single result entry of a search query.

Modules may implement hook_search_item() in order to override this default function to display search results.

Parameters:

$item A single search result as returned by hook_search(). The result should be an array with keys "link", "title", "type", "user", "date", and "snippet". Optionally, "extra" can be an array of extra info to show along with the result.

$type The type of item found, such as "user" or "node".

Definition at line 830 of file search.module.

```
830                                         {
831   if (module_hook($type, 'search_item')) {
832     $output = module_invoke($type, 'search_item', $item);
833   }
834   else {
835     $output = ' <dt class="title"><a href="'. check_url($item['link']) .'">'. check_plain($item['title
836     $info = array();
837     if ($item['type']) {
838       $info[] = $item['type'];
839     }
840     if ($item['user']) {
841       $info[] = $item['user'];
842     }
843     if ($item['date']) {
844       $info[] = format_date($item['date'], 'small');
845     }
846     if (is_array($item['extra'])) {
847       $info = array_merge($info, $item['extra']);
848     }
849     $output .= ' <dd>'. ($item['snippet'] ? '<p>'. $item['snippet'] . '</p>' : '') . '<p class="search
850   }
851
852   return $output;
853 }
```

3.8.2.38 theme_status_messages ()

Returns themed set of status and/or error messages. The messages are grouped by type.

Returns:

A string containing the messages.

Definition at line 422 of file theme.inc.

```
422                                     {
423   if ($data = drupal_get_messages()) {
424     $output = '';
425     foreach ($data as $type => $messages) {
426       $output .= "<div class=\"messages $type\">\n";
427       if (count($messages) > 1) {
428         $output .= " <ul>\n";
```

```
429         foreach($messages as $message) {
430           $output .= '  <li>'. $message ."</li>\n";
431         }
432         $output .= " </ul>\n";
433       }
434       else {
435         $output .= $messages[0];
436       }
437       $output .= "</div>\n";
438     }
439
440     return $output;
441   }
442 }
```

3.8.2.39 theme_stylesheet_import ($ *stylesheet*, $ *media* = 'all')

Import a stylesheet using .

Parameters:

$stylesheet The filename to point the link at.

$media The media type to specify for the stylesheet

Returns:

A string containing the HTML for the stylesheet import.

Definition at line 759 of file theme.inc.

```
759                                                         {
760   return '<style type="text/css" media="'. $media .'">@import "'. $stylesheet .'";</style>';
761 }
```

3.8.2.40 theme_submenu ($ *links*)

Return a themed submenu, typically displayed under the tabs.

Parameters:

$links An array of links.

Definition at line 587 of file theme.inc.

```
587                               {
588   return '<div class="submenu">'. implode(' | ', $links) .'</div>';
589 }
```

3.8.2.41 theme_table ($ *header*, $ *rows*, $ *attributes* = NULL)

Return a themed table.

Parameters:

$header An array containing the table headers. Each element of the array can be either a localized string or an associative array with the following keys:

- "data": The localized title of the table column.
- "field": The database field represented in the table column (required if user is to be able to sort on this column).
- "sort": A default sort order for this column ("asc" or "desc").
- Any HTML attributes, such as "colspan", to apply to the column header cell.

$rows An array of table rows. Every row is an array of cells, or an associative array with the following keys:

- "data": an array of cells
- Any HTML attributes, such as "class", to apply to the table row.

Each cell can be either a string or an associative array with the following keys:

- "data": The string to display in the table cell.
- Any HTML attributes, such as "colspan", to apply to the table cell.

Here's an example for $rows:

```
*   $rows = array(
*     // Simple row
*     array(
*       'Cell 1', 'Cell 2', 'Cell 3'
*     ),
*     // Row with attributes on the row and some of its cells.
*     array(
*       'data' => array('Cell 1', array('data' => 'Cell 2', 'colspan' => 2)), 'class' => 'funky'
*     )
*   );
*
```

Parameters:

$attributes An array of HTML attributes to apply to the table tag.

Returns:

An HTML string representing the table.

Definition at line 631 of file theme.inc.

References _theme_table_cell(), drupal_attributes(), tablesort_cell(), tablesort_header(), and tablesort_-init().

```
631                                                             {
632
633   $output = '<table'. drupal_attributes($attributes) .">\n";
634
635   // Format the table header:
636   if (count($header)) {
637     $ts = tablesort_init($header);
638     $output .= ' <tr>';
```

```
639     foreach ($header as $cell) {
640       $cell = tablesort_header($cell, $header, $ts);
641       $output .= _theme_table_cell($cell, 1);
642     }
643     $output .= " </tr>\n";
644   }
645
646   // Format the table rows:
647   if (count($rows)) {
648     foreach ($rows as $number => $row) {
649       $attributes = array();
650
651       // Check if we're dealing with a simple or complex row
652       if (isset($row['data'])) {
653         foreach ($row as $key => $value) {
654           if ($key == 'data') {
655             $cells = $value;
656           }
657           else {
658             $attributes[$key] = $value;
659           }
660         }
661       }
662       else {
663         $cells = $row;
664       }
665
666       // Add light/dark class
667       $class = ($number % 2 == 1) ? 'light': 'dark';
668       if (isset($attributes['class'])) {
669         $attributes['class'] .= ' '. $class;
670       }
671       else {
672         $attributes['class'] = $class;
673       }
674
675       // Build row
676       $output .= ' <tr'. drupal_attributes($attributes) .'>';
677       $i = 0;
678       foreach ($cells as $cell) {
679         $cell = tablesort_cell($cell, $header, $ts, $i++);
680         $output .= _theme_table_cell($cell, 0);
681       }
682       $output .= " </tr>\n";
683     }
684   }
685
686   $output .= "</table>\n";
687   return $output;
688 }
```

3.8.2.42 theme_xml_icon ($ *url*)

Return code that emits an XML icon.

Definition at line 811 of file theme.inc.

References theme().

```
811                               {
812   if ($image = theme('image', 'misc/xml.png', t('XML feed'), t('XML feed'))) {
813     return '<div class="xml-icon"><a href="'. check_url($url) .'">'. $image. '</a></div>';
814   }
```

```
815 }
```

3.9 Node access rights

Functions

- function node_access ($op, $node=NULL, $uid=NULL)
- function _node_access_join_sql ($node_alias= 'n', $node_access_alias= 'na')
- function _node_access_where_sql ($op= 'view', $node_access_alias= 'na', $uid=NULL)
- function node_access_grants ($op, $uid=NULL)
- function node_access_view_all_nodes ()
- function node_db_rewrite_sql ($query, $primary_table, $primary_field)

3.9.1 Detailed Description

The node access system determines who can do what to which nodes.

In determining access rights for a node, node_access() first checks whether the user has the "administer nodes" permission. Such users have unrestricted access to all nodes. Then the node module's hook_access() is called, and a TRUE or FALSE return value will grant or deny access. This allows, for example, the blog module to always grant access to the blog author, and for the book module to always deny editing access to PHP pages.

If node module does not intervene (returns NULL), then the node_access table is used to determine access. All node access modules are queried using hook_node_grants() to assemble a list of "grant IDs" for the user. This list is compared against the table. If any row contains the node ID in question (or 0, which stands for "all nodes"), one of the grant IDs returned, and a value of TRUE for the operation in question, then access is granted. Note that this table is a list of grants; any matching row is sufficient to grant access to the node.

In node listings, the process above is followed except that hook_access() is not called on each node for performance reasons and for proper functioning of the pager system. When adding a node listing to your module, be sure to use db_rewrite_sql() to add the appropriate clauses to your query for access checks.

To see how to write a node access module of your own, see node_access_example.module.

3.9.2 Function Documentation

3.9.2.1 function _node_access_join_sql ($ *node_alias* = 'n', $ *node_access_alias* = 'na')

Generate an SQL join clause for use in fetching a node listing.

Parameters:

$node_alias If the node table has been given an SQL alias other than the default "n", that must be passed here.

$node_access_alias If the node_access table has been given an SQL alias other than the default "na", that must be passed here.

Returns:
An SQL join clause.

Definition at line 1872 of file node.module.

References node_access().

```
1872                                                                                {
1873   if (user_access('administer nodes')) {
1874     return '';
1875   }
1876
1877   return 'INNER JOIN {node_access} '. $node_access_alias .' ON '. $node_access_alias .'.nid = '. $noc
1878 }
```

3.9.2.2 function _node_access_where_sql ($ *op* = 'view', $ *node_access_alias* = 'na', $ *uid* = NULL)

Generate an SQL where clause for use in fetching a node listing.

Parameters:
$op The operation that must be allowed to return a node.

$node_access_alias If the node_access table has been given an SQL alias other than the default "na", that must be passed here.

Returns:
An SQL where clause.

Definition at line 1891 of file node.module.

References node_access_grants().

```
1891                                                                                  {
1892   if (user_access('administer nodes')) {
1893     return;
1894   }
1895
1896   $sql = $node_access_alias .'.grant_'. $op .' = 1 AND CONCAT('. $node_access_alias .'.realm, '. $noc
1897   $grants = array();
1898   foreach (node_access_grants($op, $uid) as $realm => $gids) {
1899     foreach ($gids as $gid) {
1900       $grants[] = "'". $realm . $gid ."'";
1901     }
1902   }
1903   $sql .= implode(',', $grants) .')';
1904   return $sql;
1905 }
```

3.9.2.3 function node_access ($ *op*, $ *node* = NULL, $ *uid* = NULL)

Determine whether the current user may perform the given operation on the specified node.

Parameters:
$op The operation to be performed on the node. Possible values are:

- "view"
- "update"
- "delete"

$node The node object (or node array) on which the operation is to be performed.

$uid The user ID on which the operation is to be performed.

Returns:

TRUE if the operation may be performed.

Definition at line 1819 of file node.module.

References array2object(), db_query(), db_result(), module_invoke(), node_access(), and node_access_grants().

Referenced by _node_access_join_sql(), and node_access().

```
1819                                                    {
1820   // Convert the node to an object if necessary:
1821   $node = array2object($node);
1822
1823   // If the node is in a restricted format, disallow editing.
1824   if ($op == 'update' && !filter_access($node->format)) {
1825     return FALSE;
1826   }
1827
1828   if (user_access('administer nodes')) {
1829     return TRUE;
1830   }
1831
1832   if (!user_access('access content')) {
1833     return FALSE;
1834   }
1835
1836   // Can't use node_invoke(), because the access hook takes the $op parameter
1837   // before the $node parameter.
1838   $access = module_invoke(node_get_module_name($node), 'access', $op, $node);
1839   if (!is_null($access)) {
1840     return $access;
1841   }
1842
1843   // If the module did not override the access rights, use those set in the
1844   // node_access table.
1845   if ($node->nid && $node->status) {
1846     $sql = 'SELECT COUNT(*) FROM {node_access} WHERE (nid = 0 OR nid = %d) AND CONCAT(realm, gid) IN
1847     $grants = array();
1848     foreach (node_access_grants($op, $uid) as $realm => $gids) {
1849       foreach ($gids as $gid) {
1850         $grants[] = "'". $realm . $gid ."'";
1851       }
1852     }
1853     $sql .= implode(',', $grants) .') AND grant_'. $op .' = 1';
1854     $result = db_query($sql, $node->nid);
1855     return (db_result($result));
1856   }
1857   return FALSE;
1858 }
```

3.9.2.4 function node_access_grants ($ *op*, $ *uid* = NULL)

Fetch an array of permission IDs granted to the given user ID.

The implementation here provides only the universal "all" grant. A node access module should implement hook_node_grants() to provide a grant list for the user.

Parameters:

$op The operation that the user is trying to perform.

$uid The user ID performing the operation. If omitted, the current user is used.

Returns:

An associative array in which the keys are realms, and the values are arrays of grants for those realms.

Definition at line 1922 of file node.module.

Referenced by _node_access_where_sql(), and node_access().

```
1922                                                {
1923   global $user;
1924
1925   if (isset($uid)) {
1926     $user_object = user_load(array('uid' => $uid));
1927   }
1928   else {
1929     $user_object = $user;
1930   }
1931
1932   return array_merge(array('all' => array(0)), module_invoke_all('node_grants', $user_object, $op));
1933 }
```

3.9.2.5 function node_access_view_all_nodes ()

Determine whether the user has a global viewing grant for all nodes.

Definition at line 1938 of file node.module.

References db_result().

```
1938                                       {
1939   static $access;
1940
1941   if (!isset($access)) {
1942     $sql = 'SELECT COUNT(*) FROM {node_access} WHERE nid = 0 AND CONCAT(realm, gid) IN (';
1943     $grants = array();
1944     foreach (node_access_grants('view') as $realm => $gids) {
1945       foreach ($gids as $gid) {
1946         $grants[] = "'". $realm . $gid ."'";
1947       }
1948     }
1949     $sql .= implode(',', $grants) .') AND grant_view = 1';
1950     $result = db_query($sql, $node->nid);
1951     $access = db_result($result);
1952   }
1953
1954   return $access;
1955 }
```

3.9.2.6 function node_db_rewrite_sql ($ *query*, $ *primary_table*, $ *primary_field*)

Implementation of hook_db_rewrite_sql

Definition at line 1960 of file node.module.

```
1960                                                                    {
1961   if ($primary_field == 'nid' && !node_access_view_all_nodes()) {
1962     $return['join'] = _node_access_join_sql($primary_table);
1963     $return['where'] = _node_access_where_sql();
1964     $return['distinct'] = 1;
1965     return $return;
1966   }
1967 }
```

3.10 Search interface

Functions

- function search_index ($sid, $type, $text)
- function search_form ($action= ”, $keys= ”, $type=null, $prompt=null)
- function search_data ($keys=NULL, $type= ’node’)
- function search_excerpt ($keys, $text)

3.10.1 Detailed Description

Helper function for grabbing search keys.

Menu callback; presents the search form and/or search results.

The Drupal search interface manages a global search mechanism.

Modules may plug into this system to provide searches of different types of data. Most of the system is handled by search.module, so this must be enabled for all of the search features to work.

There are three ways to interact with the search system:

- Specifically for searching nodes, you can implement nodeapi(’update index’) and nodeapi(’search result’). However, note that the search system already indexes all visible output of a node, i.e. everything displayed normally by hook_view() and hook_nodeapi(’view’). This is usually sufficient. You should only use this mechanism if you want additional, non-visible data to be indexed.
- Implement hook_search(). This will create a search tab for your module on the /search page with a simple keyword search form. You may optionally implement hook_search_item() to customize the display of your results.
- Implement hook_update_index(). This allows your module to use Drupal’s HTML indexing mechanism for searching full text efficiently.

If your module needs to provide a more complicated search form, then you need to implement it yourself without hook_search(). In that case, you should define it as a local task (tab) under the /search page (e.g. /search/mymodule) so that users can easily find it.

3.10.2 Function Documentation

3.10.2.1 function search_data ($ *keys* = NULL, $ *type* = ’node’)

Perform a standard search on the given keys, and return the formatted results.

Definition at line 678 of file search.module.

```
678                                                 {
679   $output = '';
680
```

```
681   if (isset($keys)) {
682     if (module_hook($type, 'search')) {
683       $results = module_invoke($type, 'search', 'search', $keys);
684       if (is_array($results) && count($results)) {
685         $output .= '<dl class="search-results">';
686         foreach ($results as $entry) {
687           $output .= theme('search_item', $entry, $type);
688         }
689         $output .= '</dl>';
690         $output .= theme('pager', NULL, 15, 0);
691       }
692     }
693   }
694
695   return $output;
696 }
```

3.10.2.2 function search_excerpt ($ *keys*, $ *text*)

Returns snippets from a piece of text, with certain keywords highlighted. Used for formatting search results.

Parameters:

$keys A string containing keywords. They are split into words using the same rules as search indexing.

$text The text to extract fragments from.

Returns:

A string containing HTML for the excerpt.

Definition at line 712 of file search.module.

```
712                                     {
713   $keys = search_keywords_split($keys);
714   $text = strip_tags(str_replace(array('<', '>'), array(' <', '> '), $text));
715   array_walk($keys, '_search_excerpt_replace');
716   $workkeys = $keys;
717
718   // Extract a fragment per keyword for at most 4 keywords.
719   // First we collect ranges of text around each keyword, starting/ending
720   // at spaces.
721   // If the sum of all fragments is too short, we look for second occurrences.
722   $ranges = array();
723   $included = array();
724   $length = 0;
725   while ($length < 256 && count($workkeys)) {
726     foreach ($workkeys as $k => $key) {
727       if (strlen($key) == 0) {
728         unset($workkeys[$k]);
729         continue;
730       }
731       if ($length >= 256) {
732         break;
733       }
734       // Remember occurrence of key so we can skip over it if more occurrences
735       // are desired.
736       if (!isset($included[$key])) {
737         $included[$key] = 0;
738       }
739       // Locate a keyword (position $p), then locate a space in front (position
```

```
740       // $q) and behind it (position $s)
741       if (preg_match('/\b'. $key .'\b/iu', $text, $match, PREG_OFFSET_CAPTURE, $included[$key])) {
742         $p = $match[0][1];
743         if (($q = strpos($text, ' ', max(0, $p - 60))) !== false) {
744           $end = substr($text, $p, 80);
745           if (($s = strrpos($end, ' ')) !== false) {
746             $ranges[$q] = $p + $s;
747             $length += $p + $s - $q;
748             $included[$key] = $p + 1;
749           }
750           else {
751             unset($workkeys[$k]);
752           }
753         }
754         else {
755           unset($workkeys[$k]);
756         }
757       }
758       else {
759         unset($workkeys[$k]);
760       }
761     }
762   }
763
764   // If we didn't find anything, return the beginning.
765   if (count($ranges) == 0) {
766     return truncate_utf8($text, 256) . ' ...';
767   }
768
769   // Sort the text ranges by starting position.
770   ksort($ranges);
771
772   // Now we collapse overlapping text ranges into one. The sorting makes it O(n).
773   $newranges = array();
774   foreach ($ranges as $from2 => $to2) {
775     if (!isset($from1)) {
776       $from1 = $from2;
777       $to1 = $to2;
778       continue;
779     }
780     if ($from2 <= $to1) {
781       $to1 = max($to1, $to2);
782     }
783     else {
784       $newranges[$from1] = $to1;
785       $from1 = $from2;
786       $to1 = $to2;
787     }
788   }
789   $newranges[$from1] = $to1;
790
791   // Fetch text
792   $out = array();
793   foreach ($newranges as $from => $to) {
794     $out[] = substr($text, $from, $to - $from);
795   }
796   $text = (isset($newranges[0]) ? '' : '... '). implode(' ... ', $out) .' ...';
797
798   // Highlight keywords. Must be done at once to prevent conflicts ('strong' and '<strong>').
799   $text = preg_replace('/\b('. implode('|', $keys) .')\b/iu', '<strong>\0</strong>', $text);
800   return $text;
801 }
```

3.10.2.3 function search_form ($ *action* = ”, $ *keys* = ”, $ *type* = null, $ *prompt* = null)

Render a search form.

Parameters:

$action Form action. Defaults to "search".

$keys The search string entered by the user, containing keywords for the search.

$type The type of search to render the node for. Must be the name of module which implements hook_search(). Defaults to ’node’.

$prompt A piece of text to put before the form (e.g. "Enter your keywords")

Returns:

An HTML string containing the search form.

Definition at line 651 of file search.module.

References form().

```
651                                                                                {
652   $edit = $_POST[’edit’];
653
654   if (!$action) {
655     $action = url(’search/’. $type);
656   }
657   if (!$type) {
658     $type = ’node’;
659   }
660   if (is_null($prompt)) {
661     $prompt = t(’Enter your keywords’);
662   }
663
664   $output = ’ <div class="search-form">’;
665   $box = ’<div class="container-inline">’;
666   $box .= form_textfield(’’, ’keys’, $keys, $prompt ? 40 : 20, 255);
667   $box .= form_submit(t(’Search’));
668   $box .= ’</div>’;
669   $output .= form_item($prompt, $box);
670   $output .= ’</div>’;
671
672   return form($output, ’post’, $action);
673 }
```

3.10.2.4 function search_index ($ *sid*, $ *type*, $ *text*)

Update the full-text search index for a particular item.

Parameters:

$sid A number identifying this particular item (e.g. node id).

$type A string defining this type of item (e.g. ’node’)

$text The content of this item. Must be a piece of HTML text.

Definition at line 324 of file search.module.

References $base_url, search_index(), and variable_get().

Referenced by search_index().

Chapter 4

Drupal File Documentation

4.1 bootstrap.inc File Reference

Enumerations

- enum **CACHE_PERMANENT**
- enum **CACHE_TEMPORARY**
- enum **WATCHDOG_NOTICE**
- enum **WATCHDOG_WARNING**
- enum **WATCHDOG_ERROR**

Functions

- conf_init ()
- drupal_get_filename ($type, $name, $filename=NULL)
- variable_init ($conf=array())
- variable_get ($name, $default)
- variable_set ($name, $value)
- variable_del ($name)
- cache_get ($key)
- cache_set ($cid, $data, $expire=CACHE_PERMANENT, $headers=NULL)
- cache_clear_all ($cid=NULL, $wildcard=false)
- page_set_cache ()
- page_get_cache ()
- bootstrap_invoke_all ($op)
- drupal_load ($type, $name)
- drupal_get_path_map ($action= ”)
- drupal_get_path_alias ($path)
- drupal_get_title ()
- drupal_set_title ($title=NULL)
- drupal_page_header ()
- bootstrap_hooks ()

- drupal_unpack ($obj, $field= 'data')
- referer_uri ()
- arg ($index)
- check_url ($uri)
- request_uri ()
- timer_start ()
- watchdog ($type, $message, $severity=WATCHDOG_NOTICE, $link=NULL)
- drupal_set_message ($message=NULL, $type= 'status')
- drupal_get_messages ()

Variables

- **$config** = conf_init()
- **$conf** = variable_init(isset($conf) ? $conf : array())

4.1.1 Detailed Description

Functions that need to be loaded on every Drupal request.

Definition in file bootstrap.inc.

4.1.2 Function Documentation

4.1.2.1 arg ($ *index*)

Return a component of the current Drupal path.

When viewing a page at the path "admin/node/configure", for example, arg(0) would return "admin", arg(1) would return "node", and arg(2) would return "configure".

Avoid use of this function where possible, as resulting code is hard to read. Instead, attempt to use named arguments in menu callback functions. See the explanation in menu.inc for how to construct callbacks that take arguments.

Definition at line 528 of file bootstrap.inc.

```
528                     {
529   static $arguments, $q;
530
531   if (empty($arguments) || $q != $_GET['q']) {
532     $arguments = explode('/', $_GET['q']);
533   }
534
535   if (array_key_exists($index, $arguments)) {
536     return $arguments[$index];
537   }
538 }
```

4.1.2.2 bootstrap_hooks ()

Define the critical hooks that force modules to always be loaded.

Definition at line 485 of file bootstrap.inc.

```
485                             {
486   return array('init', 'exit');
487 }
```

4.1.2.3 bootstrap_invoke_all ($ *op*)

Call all init or exit hooks without including all modules.

Parameters:

$op The name of the bootstrap hook we wish to invoke.

Definition at line 312 of file bootstrap.inc.

References drupal_load(), module_invoke(), and module_list().

Referenced by drupal_page_header().

```
312                                     {
313   foreach (module_list(FALSE, TRUE) as $module) {
314     drupal_load('module', $module);
315     module_invoke($module, $op);
316   }
317 }
```

4.1.2.4 cache_clear_all ($ *cid* = NULL, $ *wildcard* = false)

Expire data from the cache.

Parameters:

$cid If set, the cache ID to delete. Otherwise, all cache entries that can expire are deleted.

$wildcard If set to true, the $cid is treated as a substring to match rather than a complete ID.

Definition at line 245 of file bootstrap.inc.

References db_query().

Referenced by menu_rebuild(), variable_del(), and variable_set().

```
245                                                         {
246   if (empty($cid)) {
247     db_query("DELETE FROM {cache} WHERE expire != %d AND expire < %d", CACHE_PERMANENT, time());
248   }
249   else {
250     if ($wildcard) {
251       db_query("DELETE FROM {cache} WHERE cid LIKE '%%%s%%'", $cid);
252     }
```

```
253     else {
254       db_query("DELETE FROM {cache} WHERE cid = '%s'", $cid);
255     }
256   }
257 }
```

4.1.2.5 cache_get ($ *key*)

Return data from the persistent cache.

Parameters:

$key The cache ID of the data to retrieve.

Definition at line 198 of file bootstrap.inc.

References db_decode_blob(), db_fetch_object(), and db_query().

Referenced by menu_get_menu(), page_get_cache(), and variable_init().

```
198                           {
199   $cache = db_fetch_object(db_query("SELECT data, created, headers FROM {cache} WHERE cid = '%s'", $ke
200   if (isset($cache->data)) {
201     $cache->data = db_decode_blob($cache->data);
202     return $cache;
203   }
204   return 0;
205 }
```

4.1.2.6 cache_set ($ *cid*, $ *data*, $ *expire* = CACHE_PERMANENT, $ *headers* = NULL)

Store data in the persistent cache.

Parameters:

$cid The cache ID of the data to store.

$data The data to store in the cache. Complex data types must be serialized first.

$expire One of the following values:

- CACHE_PERMANENT: Indicates that the item should never be removed unless explicitly told to using cache_clear_all() with a cache ID.
- CACHE_TEMPORARY: Indicates that the item should be removed at the next general cache wipe.
- A Unix timestamp: Indicates that the item should be kept at least until the given time, after which it behaves like CACHE_TEMPORARY.

$headers A string containing HTTP header information for cached pages.

Definition at line 225 of file bootstrap.inc.

References cache_set(), db_affected_rows(), db_encode_blob(), and db_query().

Referenced by cache_set(), menu_get_menu(), page_set_cache(), and variable_init().

```
225                                                                                 {
226   $data = db_encode_blob($data);
227
228   db_query("UPDATE {cache} SET data = '%s', created = %d, expire = %d, headers = '%s' WHERE cid = '%s'
229   if (!db_affected_rows()) {
230     @db_query("INSERT INTO {cache} (cid, data, created, expire, headers) VALUES ('%s', '%s', %d, %d, '
231   }
232 }
```

4.1.2.7 check_url ($ *uri*)

Prepare a URL for use in an HTML attribute.

We replace (and) with their url-encoded equivalents to prevent XSS attacks.

Definition at line 545 of file bootstrap.inc.

Referenced by pager_link().

```
545                           {
546   $uri = htmlspecialchars($uri, ENT_QUOTES);
547
548   $uri = strtr($uri, array('(' => '%28', ')' => '%29'));
549
550   return $uri;
551 }
```

4.1.2.8 conf_init ()

Locate the appropriate configuration file.

Try finding a matching configuration directory by stripping the website's hostname from left to right and pathname from right to left. The first configuration file found will be used, the remaining will ignored. If no configuration file is found, return a default value '$confdir/default'.

Example for a fictitious site installed at http://www.drupal.org/mysite/test/ the 'settings.php' is searched in the following directories:

1. $confdir/www.drupal.org.mysite.test 2. $confdir/drupal.org.mysite.test 3. $confdir/org.mysite.test

4. $confdir/www.drupal.org.mysite 5. $confdir/drupal.org.mysite 6. $confdir/org.mysite

7. $confdir/www.drupal.org 8. $confdir/drupal.org 9. $confdir/org

10. $confdir/default

Definition at line 44 of file bootstrap.inc.

Referenced by drupal_get_filename().

```
44                        {
45   static $conf = '';
46
47   if ($conf) {
48     return $conf;
49   }
50
51   $confdir = 'sites';
```

```
52   $uri = explode('/', $_SERVER['PHP_SELF']);
53   $server = explode('.', rtrim($_SERVER['HTTP_HOST'], '.'));
54   for ($i = count($uri) - 1; $i > 0; $i--) {
55     for ($j = count($server); $j > 0; $j--) {
56       $dir = implode('.', array_slice($server, -$j)) . implode('.', array_slice($uri, 0, $i));
57       if (file_exists("$confdir/$dir/settings.php")) {
58         $conf = "$confdir/$dir";
59         return $conf;
60       }
61     }
62   }
63   $conf = "$confdir/default";
64   return $conf;
65 }
```

4.1.2.9 drupal_get_filename ($ *type*, $ *name*, $ *filename* = NULL)

Returns and optionally sets the filename for a system item (module, theme, etc.). The filename, whether provided, cached, or retrieved from the database, is only returned if the file exists.

Parameters:

$type The type of the item (i.e. theme, theme_engine, module).

$name The name of the item for which the filename is requested.

$filename The filename of the item if it is to be set explicitly rather than by consulting the database.

Returns:

The filename of the requested item.

Definition at line 83 of file bootstrap.inc.

References conf_init(), db_query(), and db_result().

Referenced by drupal_load(), and module_list().

```
83                                                                {
84   static $files = array();
85
86   if (!$files[$type]) {
87     $files[$type] = array();
88   }
89
90   if ($filename && file_exists($filename)) {
91     $files[$type][$name] = $filename;
92   }
93   elseif ($files[$type][$name]) {
94     // nothing
95   }
96   elseif (($file = db_result(db_query("SELECT filename FROM {system} WHERE name = '%s' AND type = '%s'"
97     $files[$type][$name] = $file;
98   }
99   else {
100    $config = conf_init();
101    $dir = (($type == 'theme_engine') ? 'themes/engines' : "${type}s");
102    $file = (($type == 'theme_engine') ? "$name.engine" : "$name.$type");
103
104    foreach (array("$config/$dir/$file", "$config/$dir/$name/$file", "$dir/$file", "$dir/$name/$file")
105      if (file_exists($file)) {
106        $files[$type][$name] = $file;
```

```
107         break;
108       }
109     }
110   }
111
112   return $files[$type][$name];
113 }
```

4.1.2.10 drupal_get_messages ()

Return all messages that have been set.

As a side effect, this function clears the message queue.

Definition at line 632 of file bootstrap.inc.

References drupal_set_message().

```
632                                 {
633   $messages = drupal_set_message();
634   $_SESSION['messages'] = array();
635
636   return $messages;
637 }
```

4.1.2.11 drupal_get_path_alias ($ *path*)

Given an internal Drupal path, return the alias set by the administrator.

Definition at line 375 of file bootstrap.inc.

References drupal_get_path_map().

Referenced by url().

```
375                                     {
376   if (($map = drupal_get_path_map()) && ($newpath = array_search($path, $map))) {
377     return $newpath;
378   }
379   elseif (function_exists('conf_url_rewrite')) {
380     return conf_url_rewrite($path, 'outgoing');
381   }
382   else {
383     // No alias found. Return the normal path.
384     return $path;
385   }
386 }
```

4.1.2.12 drupal_get_path_map ($ *action* = '')

Return an array mapping path aliases to their internal Drupal paths.

Definition at line 354 of file bootstrap.inc.

References db_fetch_object(), and db_query().

Referenced by drupal_get_normal_path(), drupal_get_path_alias(), and drupal_rebuild_path_map().

```
354                                             {
355   static $map = NULL;
356
357   if ($action == 'rebuild') {
358     $map = NULL;
359   }
360
361   if (is_null($map)) {
362     $map = array();  // Make $map non-null in case no aliases are defined.
363     $result = db_query('SELECT * FROM {url_alias}');
364     while ($data = db_fetch_object($result)) {
365       $map[$data->dst] = $data->src;
366     }
367   }
368
369   return $map;
370 }
```

4.1.2.13 drupal_get_title ()

Get the title of the current page, for display on the page and in the title bar.

Definition at line 391 of file bootstrap.inc.

References check_plain(), drupal_set_title(), and menu_get_active_title().

Referenced by theme_page().

```
391                               {
392   $title = drupal_set_title();
393
394   if (!isset($title)) {
395     // during a bootstrap, menu.inc is not included and thus we cannot provide a title
396     if (function_exists('menu_get_active_title')) {
397       $title = check_plain(menu_get_active_title());
398     }
399   }
400
401   return $title;
402 }
```

4.1.2.14 drupal_load ($ *type*, $ *name*)

Includes a file with the provided type and name. This prevents including a theme, engine, module, etc., more than once.

Parameters:

$type The type of item to load (i.e. theme, theme_engine, module).

$name The name of the item to load.

Returns:

TRUE if the item is loaded or has already been loaded.

Definition at line 331 of file bootstrap.inc.

References drupal_get_filename().

Referenced by bootstrap_invoke_all(), and module_load_all().

```
331                                     {
332   // print $name. '<br />';
333   static $files = array();
334
335   if ($files[$type][$name]) {
336     return TRUE;
337   }
338
339   $filename = drupal_get_filename($type, $name);
340
341   if ($filename) {
342     include_once($filename);
343     $files[$type][$name] = TRUE;
344
345     return TRUE;
346   }
347
348   return FALSE;
349 }
```

4.1.2.15 drupal_page_header ()

Set HTTP headers in preparation for a page response.

Definition at line 419 of file bootstrap.inc.

References bootstrap_invoke_all(), page_get_cache(), timer_start(), and variable_get().

```
419                                 {
420   if (variable_get('dev_timer', 0)) {
421     timer_start();
422   }
423
424   if (variable_get('cache', 0)) {
425     if ($cache = page_get_cache()) {
426       bootstrap_invoke_all('init');
427       // Set default values:
428       $date = gmdate('D, d M Y H:i:s', $cache->created) .' GMT';
429       $etag = '"'. md5($date) .'"';
430
431       // Check http headers:
432       $modified_since = isset($_SERVER['HTTP_IF_MODIFIED_SINCE']) ? $_SERVER['HTTP_IF_MODIFIED_SINCE']
433       if (!empty($_SERVER['HTTP_IF_MODIFIED_SINCE']) && ($timestamp = strtotime($_SERVER['HTTP_IF_MODI
434         $modified_since = $cache->created <= $timestamp;
435       }
436       else {
437         $modified_since = NULL;
438       }
439       $none_match = !empty($_SERVER['HTTP_IF_NONE_MATCH']) ? $_SERVER['HTTP_IF_NONE_MATCH'] == $etag :
440
441       // The type checking here is very important, be careful when changing entries.
442       if (($modified_since !== NULL || $none_match !== NULL) && $modified_since !== false && $none_mat
443         header('HTTP/1.0 304 Not Modified');
444         exit();
445       }
446
```

```
447       // Send appropriate response:
448       header("Last-Modified: $date");
449       header("ETag: $etag");
450
451       // Determine if the browser accepts gzipped data.
452       if (@strpos($_SERVER['HTTP_ACCEPT_ENCODING'], 'gzip') === false && function_exists('gzencode'))
453         // Strip the gzip header and run uncompress.
454         $cache->data = gzinflate(substr(substr($cache->data, 10), 0, -8));
455       }
456       elseif (function_exists('gzencode')) {
457         header('Content-Encoding: gzip');
458       }
459
460       // Send the original request's headers.  We send them one after
461       // another so PHP's header() function can deal with duplicate
462       // headers.
463       $headers = explode("\n", $cache->headers);
464       foreach ($headers as $header) {
465         header($header);
466       }
467
468       print $cache->data;
469       bootstrap_invoke_all('exit');
470       exit();
471     }
472     else {
473       header("Expires: Sun, 19 Nov 1978 05:00:00 GMT");
474       header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
475       header("Cache-Control: no-store, no-cache, must-revalidate");
476       header("Cache-Control: post-check=0, pre-check=0", false);
477       header("Pragma: no-cache");
478     }
479   }
480 }
```

4.1.2.16 drupal_set_message ($ *message* = NULL, $ *type* = 'status')

Set a message for the user to see.

The message is stored in the session so that it can persist through a redirect.

If called with no arguments, this function returns all set messages without clearing them.

Definition at line 611 of file bootstrap.inc.

Referenced by _locale_add_language(), _locale_import_parse_plural_forms(), _locale_import_read_po(), _locale_string_save(), drupal_get_messages(), file_check_directory(), file_copy(), file_move(), file_save_-data(), file_save_upload(), form_set_error(), and page_get_cache().

```
611                                                                 {
612   if (isset($message)) {
613     if (!isset($_SESSION['messages'])) {
614       $_SESSION['messages'] = array();
615     }
616
617     if (!isset($_SESSION['messages'][$type])) {
618       $_SESSION['messages'][$type] = array();
619     }
620
621     $_SESSION['messages'][$type][] = $message;
622   }
623
```

```
624   return $_SESSION['messages'];
625 }
```

4.1.2.17 drupal_set_title ($ *title* = NULL)

Set the title of the current page, for display on the page and in the title bar.

Definition at line 407 of file bootstrap.inc.

Referenced by drupal_access_denied(), drupal_get_title(), drupal_not_found(), theme_confirm(), and theme_forum_display().

```
407                                         {
408   static $stored_title;
409
410   if (isset($title)) {
411     $stored_title = $title;
412   }
413   return $stored_title;
414 }
```

4.1.2.18 drupal_unpack ($ *obj*, $ *field* = 'data')

Unserializes and appends elements from a serialized string.

Parameters:

$obj The object to which the elements are appended.

$field The attribute of $obj whose value should be unserialized.

Definition at line 497 of file bootstrap.inc.

```
497                                                 {
498   if ($obj->$field && $data = unserialize($obj->$field)) {
499     foreach ($data as $key => $value) {
500       if (!isset($obj->$key)) {
501         $obj->$key = $value;
502       }
503     }
504   }
505   return $obj;
506 }
```

4.1.2.19 page_get_cache ()

Retrieve the current page from the cache.

Note, we do not serve cached pages when status messages are waiting (from a redirected form submission which was completed). Because the output handler is not activated, the resulting page will not get cached either.

Definition at line 290 of file bootstrap.inc.

References $base_url, cache_get(), drupal_set_message(), and request_uri().

Referenced by drupal_page_header().

```
290                           {
291   global $user, $base_url;
292
293   $cache = NULL;
294
295   if (!$user->uid && $_SERVER['REQUEST_METHOD'] == 'GET' && count(drupal_set_message()) == 0) {
296     $cache = cache_get($base_url . request_uri());
297
298     if (empty($cache)) {
299       ob_start();
300     }
301   }
302
303   return $cache;
304 }
```

4.1.2.20 page_set_cache ()

Store the current page in the cache.

Definition at line 262 of file bootstrap.inc.

References $base_url, cache_set(), drupal_get_headers(), and request_uri().

Referenced by drupal_page_footer().

```
262                           {
263   global $user, $base_url;
264
265   if (!$user->uid && $_SERVER['REQUEST_METHOD'] == 'GET') {
266     // This will fail in some cases, see page_get_cache() for the explanation.
267     if ($data = ob_get_contents()) {
268       if (function_exists('gzencode')) {
269         if (version_compare(phpversion(), '4.2', '>=')) {
270           $data = gzencode($data, 9, FORCE_GZIP);
271         }
272         else {
273           $data = gzencode($data, FORCE_GZIP);
274         }
275       }
276       ob_end_flush();
277       cache_set($base_url . request_uri(), $data, CACHE_TEMPORARY, drupal_get_headers());
278     }
279   }
280 }
```

4.1.2.21 referer_uri ()

Return the URI of the referring page.

Definition at line 511 of file bootstrap.inc.

```
511                         {
512   if (isset($_SERVER['HTTP_REFERER'])) {
513     return $_SERVER['HTTP_REFERER'];
514   }
515 }
```

4.1.2.22 request_uri ()

Since request_uri() is only available on Apache, we generate an equivalent using other environment vars.

Definition at line 557 of file bootstrap.inc.

Referenced by form(), page_get_cache(), page_set_cache(), and watchdog().

```
557                         {
558
559   if (isset($_SERVER['REQUEST_URI'])) {
560     $uri = $_SERVER['REQUEST_URI'];
561   }
562   else {
563     if (isset($_SERVER['argv'])) {
564       $uri = $_SERVER['PHP_SELF'] .'?'. $_SERVER['argv'][0];
565     }
566     else {
567       $uri = $_SERVER['PHP_SELF'] .'?'. $_SERVER['QUERY_STRING'];
568     }
569   }
570
571   return $uri;
572 }
```

4.1.2.23 timer_start ()

Begin a global timer, for benchmarking of page execution time.

Definition at line 577 of file bootstrap.inc.

Referenced by drupal_page_header().

```
577                         {
578   global $timer;
579   list($usec, $sec) = explode(' ', microtime());
580   $timer = (float)$usec + (float)$sec;
581 }
```

4.1.2.24 variable_del ($ *name*)

Unset a persistent variable.

Parameters:

$name The name of the variable to undefine.

Definition at line 183 of file bootstrap.inc.

References cache_clear_all(), and db_query().

Referenced by update_104().

```
183                               {
184   global $conf;
185
186   db_query("DELETE FROM {variable} WHERE name = '%s'", $name);
187   cache_clear_all('variables');
188
189   unset($conf[$name]);
190 }
```

4.1.2.25 variable_get ($ *name*, $ *default*)

Return a persistent variable.

Parameters:

$name The name of the variable to return.

$default The default value to use if this variable has never been set.

Returns:

The value of the variable.

Definition at line 152 of file bootstrap.inc.

Referenced by _db_query(), _locale_export_po(), drupal_access_denied(), drupal_not_found(), drupal_page_footer(), drupal_page_header(), error_handler(), file_create_path(), file_create_url(), file_save_data(), file_save_upload(), format_name(), init_theme(), module_list(), search_index(), theme_forum_display(), theme_get_setting(), theme_get_settings(), theme_page(), update_104(), and url().

```
152                                       {
153   global $conf;
154
155   return isset($conf[$name]) ? $conf[$name] : $default;
156 }
```

4.1.2.26 variable_init ($ *conf* = array())

Load the persistent variable table.

The variable table is composed of values that have been saved in the table with variable_set() as well as those explicitly specified in the configuration file.

Definition at line 122 of file bootstrap.inc.

References cache_get(), cache_set(), db_fetch_object(), and db_query().

```
122                                       {
123   // NOTE: caching the variables improves performance with 20% when serving cached pages.
124   if ($cached = cache_get('variables')) {
```

```
125     $variables = unserialize($cached->data);
126   }
127   else {
128     $result = db_query('SELECT * FROM {variable}');
129     while ($variable = db_fetch_object($result)) {
130       $variables[$variable->name] = unserialize($variable->value);
131     }
132     cache_set('variables', serialize($variables));
133   }
134
135   foreach ($conf as $name => $value) {
136     $variables[$name] = $value;
137   }
138
139   return $variables;
140 }
```

4.1.2.27 variable_set ($ *name*, $ *value*)

Set a persistent variable.

Parameters:

$name The name of the variable to set.

$value The value to set. This can be any PHP data type; these functions take care of serialization as necessary.

Definition at line 167 of file bootstrap.inc.

References cache_clear_all(), and db_query().

Referenced by update_104().

```
167                                     {
168   global $conf;
169
170   db_query("DELETE FROM {variable} WHERE name = '%s'", $name);
171   db_query("INSERT INTO {variable} (name, value) VALUES ('%s', '%s')", $name, serialize($value));
172   cache_clear_all('variables');
173
174   $conf[$name] = $value;
175 }
```

4.1.2.28 watchdog ($ *type*, $ *message*, $ *severity* = WATCHDOG_NOTICE, $ *link* = NULL)

Log a system message.

Parameters:

$type The category to which this message belongs.

$message The message to store in the log.

$severity The severity of the message. One of the following values:

- WATCHDOG_NOTICE
- WATCHDOG_WARNING

- WATCHDOG_ERROR

$link A link to associate with the message.

Definition at line 598 of file bootstrap.inc.

References db_query(), and request_uri().

Referenced by _locale_add_language(), _locale_export_po(), drupal_access_denied(), drupal_not_-found(), error_handler(), file_save_data(), file_save_upload(), module_list(), and valid_input_data().

```
598                                                                                  {
599   global $user;
600   db_query("INSERT INTO {watchdog} (uid, type, message, severity, link, location, hostname, timestamp)
601 }
```

4.2 common.inc File Reference

HTTP handling

Functions to properly handle HTTP responses.

- drupal_get_destination ()
- drupal_goto ($path= ”, $query=NULL, $fragment=NULL)
- drupal_not_found ()
- drupal_access_denied ()
- drupal_http_request ($url, $headers=array(), $method= ’GET’, $data=NULL, $retry=3)

Conversion

Converts data structures to different types.

- array2object ($array)
- object2array ($object)

Messages

Frequently used messages.

- message_access ()
- message_na ()

Functions

- drupal_set_breadcrumb ($breadcrumb=NULL)
- drupal_get_breadcrumb ()
- drupal_set_html_head ($data=NULL)
- drupal_get_html_head ()
- drupal_rebuild_path_map ()
- drupal_get_normal_path ($path)
- drupal_set_header ($header=NULL)
- drupal_get_headers ()
- error_handler ($errno, $message, $filename, $line)
- **_fix_gpc_magic** (&$item)
- fix_gpc_magic ()
- locale_initialize ()
- t ($string, $args=0)
- check_plain ($text)
- valid_email_address ($mail)
- valid_url ($url, $absolute=FALSE)
- valid_input_data ($data)

- flood_register_event ($name)
- flood_is_allowed ($name, $threshold)
- **check_file** ($filename)
- format_rss_channel ($title, $link, $description, $items, $language= 'en', $args=array())
- format_rss_item ($title, $link, $description, $args=array())
- format_plural ($count, $singular, $plural)
- format_size ($size)
- format_interval ($timestamp, $granularity=2)
- format_date ($timestamp, $type= 'medium', $format= '', $timezone=NULL)
- format_name ($object)
- form ($form, $method= 'post', $action=NULL, $attributes=NULL)
- form_set_error ($name, $message)
- form_get_errors ()
- _form_get_error ($name)
- **_form_get_class** ($name, $required, $error)
- form_item ($title, $value, $description=NULL, $id=NULL, $required=FALSE, $error=FALSE)
- form_group ($legend, $group, $description=NULL)
- form_radio ($title, $name, $value=1, $checked=FALSE, $description=NULL, $attributes=NULL, $required=FALSE)
- form_radios ($title, $name, $value, $options, $description=NULL, $required=FALSE, $attributes=NULL)
- form_checkbox ($title, $name, $value=1, $checked=FALSE, $description=NULL, $attributes=NULL, $required=FALSE)
- form_checkboxes ($title, $name, $values, $options, $description=NULL, $attributes=NULL, $required=FALSE)
- form_textfield ($title, $name, $value, $size, $maxlength, $description=NULL, $attributes=NULL, $required=FALSE)
- form_password ($title, $name, $value, $size, $maxlength, $description=NULL, $attributes=NULL, $required=FALSE)
- form_textarea ($title, $name, $value, $cols, $rows, $description=NULL, $attributes=NULL, $required=FALSE)
- form_select ($title, $name, $value, $options, $description=NULL, $extra=0, $multiple=FALSE, $required=FALSE)
- form_file ($title, $name, $size, $description=NULL, $required=FALSE)
- form_hidden ($name, $value)
- form_button ($value, $name= 'op', $type= 'submit', $attributes=NULL)
- form_submit ($value, $name= 'op', $attributes=NULL)
- form_weight ($title=NULL, $name= 'weight', $value=0, $delta=10, $description=NULL, $extra=0)
- url ($path=NULL, $query=NULL, $fragment=NULL, $absolute=FALSE)
- drupal_attributes ($attributes=array())
- l ($text, $path, $attributes=array(), $query=NULL, $fragment=NULL, $absolute=FALSE, $html=FALSE)
- drupal_page_footer ()
- drupal_map_assoc ($array, $function=NULL)
- drupal_xml_parser_create (&$data)

4.2.1 Detailed Description

Common functions that many Drupal modules will need to reference.

The functions that are critical and need to be available even when serving a cached page are instead located in bootstrap.inc.

Definition in file common.inc.

4.2.2 Function Documentation

4.2.2.1 array2object ($ *array*)

Convert an associative array to an anonymous object.

Definition at line 424 of file common.inc.

Referenced by node_access().

```
424                                {
425   if (is_array($array)) {
426     $object = new StdClass();
427     foreach ($array as $key => $value) {
428       $object->$key = $value;
429     }
430   }
431   else {
432     $object = $array;
433   }
434
435   return $object;
436 }
```

4.2.2.2 check_plain ($ *text*)

Encode special characters in a plain-text string for display as HTML.

Definition at line 553 of file common.inc.

Referenced by _locale_admin_manage_screen(), _locale_string_seek_form(), drupal_get_title(), form_select(), form_textarea(), l(), theme_aggregator_block_item(), theme_aggregator_feed(), theme_aggregator_page_item(), theme_aggregator_summary_item(), theme_forum_list(), theme_forum_topic_list(), theme_node(), and theme_placeholder().

```
553                               {
554   return htmlspecialchars($text, ENT_QUOTES);
555 }
```

4.2.2.3 drupal_access_denied ()

Generates a 403 error if the request is not allowed.

Definition at line 223 of file common.inc.

References drupal_get_normal_path(), drupal_set_title(), l(), menu_execute_active_handler(), menu_set_active_item(), message_access(), theme(), variable_get(), and watchdog().

Referenced by file_download().

```
223                                  {
224   header('HTTP/1.0 403 Forbidden');
```

```
225   watchdog('access denied', t('%page denied access.', array('%page' => theme('placeholder', $_GET['q']
226
227   $path = drupal_get_normal_path(variable_get('site_403', ''));
228   $status = MENU_NOT_FOUND;
229   if ($path) {
230     menu_set_active_item($path);
231     $status = menu_execute_active_handler();
232   }
233
234   if ($status != MENU_FOUND) {
235     drupal_set_title(t('Access denied'));
236     print theme('page', message_access());
237   }
238 }
```

4.2.2.4 drupal_attributes ($ *attributes* = array())

Format an attribute string to insert in a tag.

Parameters:

$attributes An associative array of HTML attributes.

Returns:

An HTML string ready for insertion in a tag.

Definition at line 1515 of file common.inc.

Referenced by form(), form_button(), form_checkbox(), form_checkboxes(), form_password(), form_radio(), form_radios(), form_textarea(), form_textfield(), l(), and theme_table().

```
1515                                                    {
1516   if ($attributes) {
1517     $t = array();
1518     foreach ($attributes as $key => $value) {
1519       $t[] = $key .'="'. check_plain($value) .'"';
1520     }
1521     return ' '. implode($t, ' ');
1522   }
1523 }
```

4.2.2.5 drupal_get_breadcrumb ()

Get the breadcrumb trail for the current page.

Definition at line 31 of file common.inc.

References drupal_set_breadcrumb(), and menu_get_active_breadcrumb().

```
31                                  {
32   $breadcrumb = drupal_set_breadcrumb();
33
34   if (!isset($breadcrumb)) {
35     $breadcrumb = menu_get_active_breadcrumb();
36   }
37
38   return $breadcrumb;
39 }
```

4.2.2.6 drupal_get_destination ()

Prepare a destination query string for use in combination with drupal_goto(). Used to direct the user back to the referring page after completing a form.

See also:
drupal_goto()

Definition at line 125 of file common.inc.

```
125                                  {
126   $destination[] = $_GET['q'];
127   $params = array('from', 'sort', 'order');
128   foreach ($params as $param) {
129     if (isset($_GET[$param])) {
130       $destination[] = "$param=". $_GET[$param];
131     }
132   }
133   return 'destination='. urlencode(implode('&', $destination));
134 }
```

4.2.2.7 drupal_get_headers ()

Get the HTTP response headers for the current page.

Definition at line 108 of file common.inc.

References drupal_set_header().

Referenced by page_set_cache().

```
108                                {
109   return drupal_set_header();
110 }
```

4.2.2.8 drupal_get_html_head ()

Retrieve output to be displayed in the head tag of the HTML page.

Definition at line 57 of file common.inc.

References $base_url, drupal_set_html_head(), and theme().

Referenced by theme_page().

```
57                                  {
58   global $base_url;
59
60   $output = "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\" />\n";
61   $output .= "<base href=\"$base_url/\" />\n";
62   $output .= theme('stylesheet_import', 'misc/drupal.css');
63
64   return $output . drupal_set_html_head();
65 }
```

4.2.2.9 drupal_get_normal_path ($ *path*)

Given a path alias, return the internal path it represents.

Definition at line 77 of file common.inc.

References drupal_get_path_map().

Referenced by _menu_build(), drupal_access_denied(), drupal_not_found(), and l().

```
77                                      {
78   if (($map = drupal_get_path_map()) && isset($map[$path])) {
79     return $map[$path];
80   }
81   elseif (function_exists('conf_url_rewrite')) {
82     return conf_url_rewrite($path, 'incoming');
83   }
84   else {
85     return $path;
86   }
87 }
```

4.2.2.10 drupal_goto ($ *path* = '', $ *query* = NULL, $ *fragment* = NULL)

Send the user to a different Drupal page.

This issues an on-site HTTP redirect. The function makes sure the redirected URL is formatted correctly.

Usually the redirected URL is constructed from this function's input parameters. However you may override that behavior by setting a *destination* in either the $_REQUEST-array (i.e. by using the query string of an URI) or the $_REQUEST['edit']-array (i.e. by using a hidden form field). This is used to direct the user back to the proper page after completing a form. For example, after editing a post on the 'admin/node'-page or after having logged on using the 'user login'-block in a sidebar. The function drupal_get_destination() can be used to help set the destination URL.

It is advised to use drupal_goto() instead of PHP's header(), because drupal_goto() will append the user's session ID to the URI when PHP is compiled with "–enable-trans-sid".

This function ends the request; use it rather than a print theme('page') statement in your menu callback.

Parameters:

$path A Drupal path.

$query The query string component, if any.

$fragment The destination fragment identifier (named anchor).

See also:

drupal_get_destination()

Definition at line 168 of file common.inc.

References module_invoke_all(), and url().

```
168                                                     {
```

```
169   if ($_REQUEST['destination']) {
170     extract(parse_url($_REQUEST['destination']));
171   }
172   else if ($_REQUEST['edit']['destination']) {
173     extract(parse_url($_REQUEST['edit']['destination']));
174   }
175
176   $url = url($path, $query, $fragment, TRUE);
177
178   if (ini_get('session.use_trans_sid') && session_id() && !strstr($url, session_id())) {
179     $sid = session_name() . '=' . session_id();
180
181     if (strstr($url, '?') && !strstr($url, $sid)) {
182       $url = $url .'&'. $sid;
183     }
184     else {
185       $url = $url .'?'. $sid;
186     }
187   }
188
189   // Before the redirect, allow modules to react to the end of the page request.
190   module_invoke_all('exit', $url);
191
192   header('Location: '. $url);
193
194   // The "Location" header sends a REDIRECT status code to the http
195   // daemon. In some cases this can go wrong, so we make sure none
196   // of the code below the drupal_goto() call gets executed when we redirect.
197   exit();
198 }
```

4.2.2.11 drupal_http_request ($ *url*, $ *headers* = array(), $ *method* = 'GET', $ *data* = NULL, $ *retry* = 3)

Perform an HTTP request.

This is a flexible and powerful HTTP client implementation. Correctly handles GET, POST, PUT or any other HTTP requests. Handles redirects.

Parameters:

$url A string containing a fully qualified URI.

$headers An array containing an HTTP header => value pair.

$method A string defining the HTTP request to use.

$data A string containing data to include in the request.

$retry An integer representing how many times to retry the request in case of a redirect.

Returns:

An object containing the HTTP request headers, response code, headers, data, and redirect status.

Definition at line 261 of file common.inc.

References url().

```
261                                                                                 {
262   $result = new StdClass();
263
264   // Parse the URL, and make sure we can handle the schema.
```

```
265   $uri = parse_url($url);
266   switch ($uri['scheme']) {
267     case 'http':
268       $fp = @fsockopen($uri['host'], ($uri['port'] ? $uri['port'] : 80), $errno, $errstr, 15);
269       break;
270     case 'https':
271       // Note: Only works for PHP 4.3 compiled with OpenSSL.
272       $fp = @fsockopen('ssl://'. $uri['host'], ($uri['port'] ? $uri['port'] : 443), $errno, $errstr, 2
273       break;
274     default:
275       $result->error = 'invalid schema '. $uri['scheme'];
276       return $result;
277   }
278
279   // Make sure the socket opened properly.
280   if (!$fp) {
281     $result->error = trim($errno .' '. $errstr);
282     return $result;
283   }
284
285   // Construct the path to act on.
286   $path = $uri['path'] ? $uri['path'] : '/';
287   if ($uri['query']) {
288     $path .= '?'. $uri['query'];
289   }
290
291   // Create HTTP request.
292   $defaults = array(
293     'Host' => 'Host: '. $uri['host'],
294     'User-Agent' => 'User-Agent: Drupal (+http://www.drupal.org/)',
295     'Content-Length' => 'Content-Length: '. strlen($data)
296   );
297
298   foreach ($headers as $header => $value) {
299     $defaults[$header] = $header .': '. $value;
300   }
301
302   $request = $method .' '. $path ." HTTP/1.0\r\n";
303   $request .= implode("\r\n", $defaults);
304   $request .= "\r\n\r\n";
305   if ($data) {
306     $request .= $data ."\r\n";
307   }
308   $result->request = $request;
309
310   fwrite($fp, $request);
311
312   // Fetch response.
313   $response = '';
314   while (!feof($fp) && $data = fread($fp, 1024)) {
315     $response .= $data;
316   }
317   fclose($fp);
318
319   // Parse response.
320   list($headers, $result->data) = explode("\r\n\r\n", $response, 2);
321   $headers = preg_split("/\r\n|\n|\r/", $headers);
322
323   list($protocol, $code, $text) = explode(' ', trim(array_shift($headers)), 3);
324   $result->headers = array();
325
326   // Parse headers.
327   while ($line = trim(array_shift($headers))) {
328     list($header, $value) = explode(':', $line, 2);
329     $result->headers[$header] = trim($value);
330   }
331
```

```
332   $responses = array(
333     100 => 'Continue', 101 => 'Switching Protocols',
334     200 => 'OK', 201 => 'Created', 202 => 'Accepted', 203 => 'Non-Authoritative Information', 204 => '
335     300 => 'Multiple Choices', 301 => 'Moved Permanently', 302 => 'Found', 303 => 'See Other', 304 =>
336     400 => 'Bad Request', 401 => 'Unauthorized', 402 => 'Payment Required', 403 => 'Forbidden', 404 =>
337     500 => 'Internal Server Error', 501 => 'Not Implemented', 502 => 'Bad Gateway', 503 => 'Service Un
338   );
339   // RFC 2616 states that all unknown HTTP codes must be treated the same as
340   // the base code in their class.
341   if (!isset($responses[$code])) {
342     $code = floor($code / 100) * 100;
343   }
344
345   switch ($code) {
346     case 200: // OK
347     case 304: // Not modified
348       break;
349     case 301: // Moved permanently
350     case 302: // Moved temporarily
351     case 307: // Moved temporarily
352       $location = $result->headers['Location'];
353
354       if ($retry) {
355         $result = drupal_http_request($result->headers['Location'], $headers, $method, $data, --$retry
356         $result->redirect_code = $result->code;
357       }
358       $result->redirect_url = $location;
359
360       break;
361     default:
362       $result->error = $text;
363   }
364
365   $result->code = $code;
366   return $result;
367 }
```

4.2.2.12 drupal_map_assoc ($ *array*, $ *function* = NULL)

Form an associative array from a linear array.

This function walks through the provided array and constructs an associative array out of it. The keys of the resulting array will be the values of the input array. The values will be the same as the keys unless a function is specified, in which case the output of the function is used for the values instead.

Parameters:

$array A linear array.

$function The name of a function to apply to all values before output.

Returns:

An associative array.

Definition at line 1592 of file common.inc.

```
1592                                                {
1593   if (!isset($function)) {
1594     $result = array();
1595     foreach ($array as $value) {
1596       $result[$value] = $value;
```

```
1597     }
1598     return $result;
1599   }
1600   elseif (function_exists($function)) {
1601     $result = array();
1602     foreach($array as $value) {
1603       $result[$value] = $function($value);
1604     }
1605     return $result;
1606   }
1607 }
```

4.2.2.13 drupal_not_found ()

Generates a 404 error if the request can not be handled.

Definition at line 203 of file common.inc.

References drupal_get_normal_path(), drupal_set_title(), menu_execute_active_handler(), menu_set_active_item(), theme(), variable_get(), and watchdog().

Referenced by file_download(), and file_transfer().

```
203                             {
204   header('HTTP/1.0 404 Not Found');
205   watchdog('page not found', t('%page not found.', array('%page' => theme('placeholder', $_GET['q']))))
206
207   $path = drupal_get_normal_path(variable_get('site_404', ''));
208   $status = MENU_NOT_FOUND;
209   if ($path) {
210     menu_set_active_item($path);
211     $status = menu_execute_active_handler();
212   }
213
214   if ($status != MENU_FOUND) {
215     drupal_set_title(t('Page not found'));
216     print theme('page', '');
217   }
218 }
```

4.2.2.14 drupal_page_footer ()

Perform end-of-request tasks.

This function sets the page cache if appropriate, and allows modules to react to the closing of the page by calling hook_exit().

Definition at line 1568 of file common.inc.

References module_invoke_all(), page_set_cache(), and variable_get().

```
1568                               {
1569   if (variable_get('cache', 0)) {
1570     page_set_cache();
1571   }
1572
1573   module_invoke_all('exit');
1574 }
```

4.2.2.15 drupal_rebuild_path_map ()

Regenerate the path map from the information in the database.

Definition at line 70 of file common.inc.

References drupal_get_path_map().

```
70                                      {
71   drupal_get_path_map('rebuild');
72 }
```

4.2.2.16 drupal_set_breadcrumb ($ *breadcrumb* = NULL)

Set the breadcrumb trail for the current page.

Parameters:

$breadcrumb Array of links, starting with "home" and proceeding up to but not including the current page.

Definition at line 19 of file common.inc.

Referenced by drupal_get_breadcrumb().

```
19                                                      {
20   static $stored_breadcrumb;
21
22   if (isset($breadcrumb)) {
23     $stored_breadcrumb = $breadcrumb;
24   }
25   return $stored_breadcrumb;
26 }
```

4.2.2.17 drupal_set_header ($ *header* = NULL)

Set an HTTP response header for the current page.

Definition at line 92 of file common.inc.

Referenced by drupal_get_headers().

```
92                                              {
93   // We use an array to guarantee there are no leading or trailing delimiters.
94   // Otherwise, header('') could get called when serving the page later, which
95   // ends HTTP headers prematurely on some PHP versions.
96   static $stored_headers = array();
97
98   if (strlen($header)) {
99     header($header);
100     $stored_headers[] = $header;
101   }
102   return implode("\n", $stored_headers);
103 }
```

4.2.2.18 drupal_set_html_head ($ *data* = NULL)

Add output to the head tag of the HTML page. This function can be called as long the headers aren't sent.

Definition at line 45 of file common.inc.

Referenced by drupal_get_html_head().

```
45                                              {
46   static $stored_head = '';
47
48   if (!is_null($data)) {
49     $stored_head .= $data ."\n";
50   }
51   return $stored_head;
52 }
```

4.2.2.19 drupal_xml_parser_create (&$ *data*)

Prepare a new XML parser.

This is a wrapper around xml_parser_create() which extracts the encoding from the XML data first and sets the output encoding to UTF-8. This function should be used instead of xml_parser_create(), because PHP's XML parser doesn't check the input encoding itself.

This is also where unsupported encodings will be converted. Callers should take this into account: $data might have been changed after the call.

Parameters:

&$data The XML data which will be parsed later.

Returns:

An XML parser object.

Definition at line 1626 of file common.inc.

4.2.2.20 error_handler ($ *errno*, $ *message*, $ *filename*, $ *line*)

Log errors as defined by administrator Error levels: 1 = Log errors to database. 2 = Log errors to database and to screen.

Definition at line 378 of file common.inc.

References variable_get(), and watchdog().

```
378                                                             {
379   if ($errno & (E_ALL ^ E_NOTICE)) {
380     $types = array(1 => 'error', 2 => 'warning', 4 => 'parse error', 8 => 'notice', 16 => 'core error'
381     $entry = $types[$errno] .': '. $message .' in '. $filename .' on line '. $line .'.';
382
383     if (variable_get('error_level', 1) == 1) {
384       print '<pre>'. $entry .'</pre>';
385     }
```

```
386
387     watchdog('php', t('%message in %file on line %line.', array('%error' => $types[$errno], '%message'
388   }
389 }
```

4.2.2.21 fix_gpc_magic ()

Correct double-escaping problems caused by "magic quotes" in some PHP installations.

Definition at line 404 of file common.inc.

```
404                           {
405   static $fixed = false;
406   if (!$fixed && ini_get('magic_quotes_gpc')) {
407     array_walk($_GET, '_fix_gpc_magic');
408     array_walk($_POST, '_fix_gpc_magic');
409     array_walk($_COOKIE, '_fix_gpc_magic');
410     array_walk($_REQUEST, '_fix_gpc_magic');
411     $fixed = true;
412   }
413 }
```

4.2.2.22 flood_is_allowed ($ *name*, $ *threshold*)

Check if the current visitor (hostname/IP) is allowed to proceed with the specified event. The user is allowed to proceed if he did not trigger the specified event more than $threshold times per hour.

Parameters:

$name The name of the event.

$number The maximum number of the specified event per hour (per visitor).

Returns:

True if the user did not exceed the hourly threshold. False otherwise.

Definition at line 672 of file common.inc.

```
672                                             {
673   $number = db_num_rows(db_query("SELECT event FROM {flood} WHERE event = '%s' AND hostname = '%s' AND
674   return ($number < $threshold ? TRUE : FALSE);
675 }
```

4.2.2.23 flood_register_event ($ *name*)

Register an event for the current visitor (hostname/IP) to the flood control mechanism.

Parameters:

$name The name of the event.

Definition at line 656 of file common.inc.

```
656                                     {
657   db_query("INSERT INTO {flood} (event, hostname, timestamp) VALUES ('%s', '%s', %d)", $name, $_SERVER
658 }
```

4.2.2.24 l ($ *text*, $ *path*, $ *attributes* = array(), $ *query* = NULL, $ *fragment* = NULL, $ *absolute* = FALSE, $ *html* = FALSE)

Format an internal Drupal link.

This function correctly handles aliased paths, and allows themes to highlight links to the current page correctly, so all internal links output by modules should be generated by this function if possible.

Parameters:

$text The text to be enclosed with the anchor tag.

$path The Drupal path being linked to, such as "admin/node".

$attributes An associative array of HTML attributes to apply to the anchor tag.

$query A query string to append to the link.

$fragment A fragment identifier (named anchor) to append to the link.

$absolute Whether to force the output to be an absolute link (beginning with http:). Useful for links that will be displayed outside the site, such as in an RSS feed.

$html Whether the title is HTML, or just plain-text.

Returns:

an HTML string containing a link to the given path.

Definition at line 1550 of file common.inc.

References check_plain(), drupal_attributes(), and drupal_get_normal_path().

Referenced by _locale_admin_manage_screen(), _locale_string_seek(), drupal_access_denied(), format_name(), menu_get_active_breadcrumb(), tablesort_header(), theme_aggregator_block_item(), theme_aggregator_feed(), theme_aggregator_page_item(), theme_confirm(), theme_forum_display(), theme_forum_list(), theme_forum_topic_list(), theme_get_settings(), and theme_menu_item_link().

```
1550
1551   if (drupal_get_normal_path($path) == $_GET['q']) {
1552     if (isset($attributes['class'])) {
1553       $attributes['class'] .= ' active';
1554     }
1555     else {
1556       $attributes['class'] = 'active';
1557     }
1558   }
1559   return '<a href="'. check_url(url($path, $query, $fragment, $absolute)) .'"'. drupal_attributes($at
1560 }
```

4.2.2.25 locale_initialize ()

Initialize the localization system.

Definition at line 488 of file common.inc.

```
488                               {
489   global $user;
490
491   if (function_exists('i18n_get_lang')) {
492     return i18n_get_lang();
493   }
494
495   if (function_exists('locale')) {
496     $languages = locale_supported_languages();
497     $languages = $languages['name'];
498   }
499   else {
500     // Ensure the locale/language is correctly returned, even without locale.module.
501     // Useful for e.g. XML/HTML 'lang' attributes.
502     $languages = array('en' => 'English');
503   }
504   if ($user->uid && $languages[$user->language]) {
505     return $user->language;
506   }
507   else {
508     return key($languages);
509   }
510 }
```

4.2.2.26 message_access ()

Return a string with an "access denied" message.

Always consider whether to use drupal_access_denied() instead to return a proper (and customizable) 403 error.

Definition at line 470 of file common.inc.

Referenced by drupal_access_denied().

```
470                             {
471   return t('You are not authorized to access this page.');
472 }
```

4.2.2.27 message_na ()

Return a string with a "not applicable" message.

Definition at line 477 of file common.inc.

Referenced by _locale_admin_manage_screen().

```
477                         {
478   return t('n/a');
479 }
```

4.2.2.28 object2array ($ *object*)

Convert an object to an associative array.

Definition at line 441 of file common.inc.

```
441                                {
442   if (is_object($object)) {
443     foreach ($object as $key => $value) {
444       $array[$key] = $value;
445     }
446   }
447   else {
448     $array = $object;
449   }
450
451   return $array;
452 }
```

4.2.2.29 t ($ *string*, $ *args* = 0)

Translate strings to the current locale.

When using t(), try to put entire sentences and strings in one t() call. This makes it easier for translators. HTML markup within translation strings is acceptable, if necessary. The suggested syntax for a link embedded within a translation string is:

```
$msg = t('You must log in below or <a href="%url">create a new
          account</a> before viewing the next page.', array('%url'
          => url('user/register')));
```

We suggest the same syntax for links to other sites. This makes it easy to change link URLs if needed (which happens often) without requiring updates to translations.

Parameters:

$string A string containing the English string to translate.

$args An associative array of replacements to make after translation. Incidences of any key in this array are replaced with the corresponding value.

Returns:

The translated string.

Definition at line 536 of file common.inc.

```
536                                {
537   global $locale;
538   if (function_exists('locale') && $locale != 'en') {
539     $string = locale($string);
540   }
541
542   if (!$args) {
543     return $string;
544   }
545   else {
546     return strtr($string, $args);
547   }
548 }
```

4.2.2.30 url ($ *path* = NULL, $ *query* = NULL, $ *fragment* = NULL, $ *absolute* = FALSE)

Generate an internal Drupal URL.

Parameters:

$path The Drupal path being linked to, such as "admin/node".

$query A query string to append to the link.

$fragment A fragment identifier (named anchor) to append to the link.

$absolute Whether to force the output to be an absolute link (beginning with http:). Useful for links that will be displayed outside the site, such as in an RSS feed.

Returns:

an HTML string containing a link to the given path.

When creating links in modules, consider whether l() could be a better alternative than url().

Definition at line 1449 of file common.inc.

References $base_url, drupal_get_path_alias(), and variable_get().

Referenced by _locale_add_language(), _locale_admin_manage_screen(), _locale_string_seek_form(), db_connect(), drupal_goto(), drupal_http_request(), file_create_url(), menu_get_active_help(), pager_-link(), theme_forum_display(), and valid_url().

```
1449                                                                                {
1450   global $base_url;
1451
1452   static $script;
1453
1454   if (empty($script)) {
1455     // On some web servers, such as IIS, we can't omit "index.php".  So, we
1456     // generate "index.php?q=foo" instead of "?q=foo" on anything that is not
1457     // Apache.
1458     $script = (strpos($_SERVER['SERVER_SOFTWARE'], 'Apache') === false) ? 'index.php' : '';
1459   }
1460
1461   $path = drupal_get_path_alias($path);
1462
1463   if (isset($fragment)) {
1464     $fragment = '#'. $fragment;
1465   }
1466
1467   $base = ($absolute ? $base_url .'/' : '');
1468
1469   if (variable_get('clean_url', '0') == '0') {
1470     if (isset($path)) {
1471       if (isset($query)) {
1472         return $base . $script .'?q='. $path .'&'. $query . $fragment;
1473       }
1474       else {
1475         return $base . $script .'?q='. $path . $fragment;
1476       }
1477     }
1478     else {
1479       if (isset($query)) {
1480         return $base . $script .'?'. $query . $fragment;
1481       }
1482       else {
1483         return $base . $fragment;
1484       }
1485     }
1486   }
```

```
1487   else {
1488     if (isset($path)) {
1489       if (isset($query)) {
1490         return $base . $path .'?'. $query . $fragment;
1491       }
1492       else {
1493         return $base . $path . $fragment;
1494       }
1495     }
1496     else {
1497       if (isset($query)) {
1498         return $base . $script .'?'. $query . $fragment;
1499       }
1500       else {
1501         return $base . $fragment;
1502       }
1503     }
1504   }
1505 }
```

4.3 cron.php File Reference

4.3.1 Detailed Description

Handles incoming requests to fire off regularly-scheduled tasks (cron jobs).

Definition in file cron.php.

4.4 database.inc File Reference

Functions

- db_prefix_tables ($sql)
- db_set_active ($name= 'default')
- db_query ($query)
- db_queryd ($query)
- _db_rewrite_sql ($query= '', $primary_table= 'n', $primary_field= 'nid', $args=array())
- db_rewrite_sql ($query, $primary_table= 'n', $primary_field= 'nid', $args=array())

4.4.1 Detailed Description

Wrapper for database interface code.

Definition in file database.inc.

4.5 database.mysql.inc File Reference

Functions

- db_connect ($url)
- _db_query ($query, $debug=0)
- db_fetch_object ($result)
- db_fetch_array ($result)
- db_num_rows ($result)
- db_result ($result, $row=0)
- db_error ()
- db_next_id ($name)
- db_affected_rows ()
- db_query_range ($query)
- db_encode_blob ($data)
- db_decode_blob ($data)
- db_escape_string ($text)

4.5.1 Detailed Description

Database interface code for MySQL database servers.

Definition in file database.mysql.inc.

4.5.2 Function Documentation

4.5.2.1 _db_query ($ *query*, $ *debug* = 0)

Helper function for db_query().

Definition at line 40 of file database.mysql.inc.

References variable_get().

```
40                                          {
41   global $active_db;
42   global $queries;
43
44   if (variable_get('dev_query', 0)) {
45     list($usec, $sec) = explode(' ', microtime());
46     $timer = (float)$usec + (float)$sec;
47   }
48
49   $result = mysql_query($query, $active_db);
50
51   if (variable_get('dev_query', 0)) {
52     list($usec, $sec) = explode(' ', microtime());
53     $stop = (float)$usec + (float)$sec;
54     $diff = $stop - $timer;
```

```
55     $queries[] = array($query, $diff);
56   }
57
58   if ($debug) {
59     print '<p>query: '. $query .'<br />error:'. mysql_error() .'</p>';
60   }
61
62   if (!mysql_errno()) {
63     return $result;
64   }
65   else {
66     trigger_error(mysql_error() ."\nquery: ". htmlspecialchars($query), E_USER_ERROR);
67   }
68 }
```

4.5.2.2 db_affected_rows ()

Determine the number of rows changed by the preceding query.

Definition at line 162 of file database.mysql.inc.

Referenced by cache_set().

```
162                             {
163   return mysql_affected_rows();
164 }
```

4.5.2.3 db_decode_blob ($ *data*)

Returns text from a Binary Large OBject value.

Parameters:

$data Data to decode.

Returns:

Decoded data.

Definition at line 227 of file database.mysql.inc.

```
227                                 {
228   return $data;
229 }
```

4.5.2.4 db_encode_blob ($ *data*)

Returns a properly formatted Binary Large OBject value.

Parameters:

$data Data to encode.

Returns:
Encoded data.

Definition at line 215 of file database.mysql.inc.

```
215                                 {
216   return $data;
217 }
```

4.5.2.5 db_error ()

Determine whether the previous query caused an error.

Definition at line 137 of file database.mysql.inc.

```
137                     {
138   return mysql_errno();
139 }
```

4.5.2.6 db_escape_string ($ *text*)

Prepare user input for use in a database query, preventing SQL injection attacks.

Definition at line 234 of file database.mysql.inc.

Referenced by db_query_range().

```
234                                   {
235   return addslashes($text);
236 }
```

4.5.2.7 db_fetch_array ($ *result*)

Fetch one result row from the previous query as an array.

Parameters:
$result A database query result resource, as returned from db_query().

Returns:
An associative array representing the next row of the result. The keys of this object are the names of the table fields selected by the query, and the values are the field values for this result row.

Definition at line 95 of file database.mysql.inc.

```
95                                  {
96   if ($result) {
97     return mysql_fetch_array($result, MYSQL_ASSOC);
98   }
99 }
```

4.5.2.8 db_fetch_object ($ *result*)

Fetch one result row from the previous query as an object.

Parameters:

$result A database query result resource, as returned from db_query().

Returns:

An object representing the next row of the result. The attributes of this object are the table fields selected by the query.

Definition at line 79 of file database.mysql.inc.

```
79                                      {
80   if ($result) {
81     return mysql_fetch_object($result);
82   }
83 }
```

4.5.2.9 db_next_id ($ *name*)

Return a new unique ID in the given sequence.

For compatibility reasons, Drupal does not use auto-numbered fields in its database tables. Instead, this function is used to return a new unique ID of the type requested. If necessary, a new sequence with the given name will be created.

Definition at line 149 of file database.mysql.inc.

References db_prefix_tables(), db_query(), and db_result().

```
149                             {
150   $name = db_prefix_tables($name);
151   db_query('LOCK TABLES {sequences} WRITE');
152   $id = db_result(db_query("SELECT id FROM {sequences} WHERE name = '%s'", $name)) + 1;
153   db_query("REPLACE INTO {sequences} VALUES ('%s', %d)", $name, $id);
154   db_query('UNLOCK TABLES');
155
156   return $id;
157 }
```

4.5.2.10 db_num_rows ($ *result*)

Determine how many result rows were found by the preceding query.

Parameters:

$result A database query result resource, as returned from db_query().

Returns:

The number of result rows.

Definition at line 109 of file database.mysql.inc.

```
109                                {
110   if ($result) {
111     return mysql_num_rows($result);
112   }
113 }
```

4.5.2.11 db_query_range ($ *query*)

Runs a limited-range query in the active database.

Use this as a substitute for db_query() when a subset of the query is to be returned. User-supplied arguments to the query should be passed in as separate parameters so that they can be properly escaped to avoid SQL injection attacks.

Parameters:

$query A string containing an SQL query.

... A variable number of arguments which are substituted into the query using printf() syntax. Instead of a variable number of query arguments, you may also pass a single array containing the query arguments.

$from The first result row to return.

$count The maximum number of result rows to return.

Returns:

A database query result resource, or FALSE if the query was not executed correctly.

Definition at line 188 of file database.mysql.inc.

References _db_query(), db_escape_string(), and db_prefix_tables().

```
188                                {
189   $args = func_get_args();
190   $count = array_pop($args);
191   $from = array_pop($args);
192
193   $query = db_prefix_tables($query);
194   if (count(func_get_args()) > 3) {
195     // Check for array (alternative syntax).
196     if (is_array($args[1])) {
197       $args = array_merge(array($query), $args[1]);
198     }
199     $args = array_map('db_escape_string', $args);
200     $args[0] = $query;
201     $query = call_user_func_array('sprintf', $args);
202   }
203   $query .= ' LIMIT '. $from .', '. $count;
204   return _db_query($query);
205 }
```

4.5.2.12 db_result ($ *result*, $ *row* = 0)

Return an individual result field from the previous query.

Only use this function if exactly one field is being selected; otherwise, use db_fetch_object() or db_fetch_array().

Parameters:

$result A database query result resource, as returned from db_query().

$row The index of the row whose result is needed.

Returns:

The resulting field.

Definition at line 128 of file database.mysql.inc.

Referenced by node_access_view_all_nodes().

```
128                                     {
129   if ($result && mysql_num_rows($result) > $row) {
130     return mysql_result($result, $row);
131   }
132 }
```

4.6 database.pgsql.inc File Reference

Functions

- _db_query ($query, $debug=0)
- db_fetch_object ($result)
- db_fetch_array ($result)
- db_num_rows ($result)
- db_result ($result, $row=0)
- db_error ()
- db_next_id ($name)
- db_affected_rows ()
- db_query_range ($query)
- db_encode_blob ($data)
- db_decode_blob ($data)
- db_escape_string ($text)

4.6.1 Detailed Description

Database interface code for PostgreSQL database servers.

Definition in file database.pgsql.inc.

4.6.2 Function Documentation

4.6.2.1 _db_query ($ *query*, $ *debug* = 0)

Helper function for db_query().

Definition at line 36 of file database.pgsql.inc.

References variable_get().

Referenced by db_query_range().

```
36                                      {
37   global $active_db, $last_result;
38   global $queries;
39
40   if (variable_get('dev_query', 0)) {
41     list($usec, $sec) = explode(' ', microtime());
42     $timer = (float)$usec + (float)$sec;
43   }
44
45   $last_result = pg_query($active_db, $query);
46
47   if (variable_get('dev_query', 0)) {
48     list($usec, $sec) = explode(' ', microtime());
49     $stop = (float)$usec + (float)$sec;
50     $diff = $stop - $timer;
```

```
51     $queries[] = array($query, $diff);
52   }
53
54   if ($debug) {
55     print '<p>query: '. $query .'<br />error:'. pg_last_error() .'</p>';
56   }
57
58   if ($last_result !== FALSE) {
59     return $last_result;
60   }
61   else {
62     trigger_error(pg_last_error() ."\nquery: ". htmlspecialchars($query), E_USER_ERROR);
63   }
64 }
```

4.6.2.2 db_affected_rows ()

Determine the number of rows changed by the preceding query.

Definition at line 155 of file database.pgsql.inc.

```
155                             {
156   global $last_result;
157   return pg_affected_rows($last_result);
158 }
```

4.6.2.3 db_decode_blob ($ *data*)

Returns text from a Binary Large OBject value.

Parameters:

$data Data to decode.

Returns:

Decoded data.

Definition at line 221 of file database.pgsql.inc.

Referenced by cache_get().

```
221                               {
222   return stripcslashes($data);
223 }
```

4.6.2.4 db_encode_blob ($ *data*)

Returns a properly formatted Binary Large OBject value.

Parameters:

$data Data to encode.

Returns:
Encoded data.

Definition at line 209 of file database.pgsql.inc.

Referenced by cache_set().

```
209                                {
210   return addcslashes($data, "\0..\37\\");
211 }
```

4.6.2.5 db_error ()

Determine whether the previous query caused an error.

Definition at line 135 of file database.pgsql.inc.

```
135                     {
136   return pg_last_error();
137 }
```

4.6.2.6 db_escape_string ($ *text*)

Prepare user input for use in a database query, preventing SQL injection attacks. Note: This function requires PostgreSQL 7.2 or later.

Definition at line 229 of file database.pgsql.inc.

Referenced by _locale_string_seek(), and tablesort_sql().

```
229                                  {
230   return pg_escape_string($text);
231 }
```

4.6.2.7 db_fetch_array ($ *result*)

Fetch one result row from the previous query as an array.

Parameters:
$result A database query result resource, as returned from db_query().

Returns:
An associative array representing the next row of the result. The keys of this object are the names of the table fields selected by the query, and the values are the field values for this result row.

Definition at line 91 of file database.pgsql.inc.

Referenced by update_101().

```
91                                  {
92   if ($result) {
93     return pg_fetch_assoc($result);
94   }
95 }
```

4.6.2.8 db_fetch_object ($ *result*)

Fetch one result row from the previous query as an object.

Parameters:

$result A database query result resource, as returned from db_query().

Returns:

An object representing the next row of the result. The attributes of this object are the table fields selected by the query.

Definition at line 75 of file database.pgsql.inc.

Referenced by _locale_add_language(), _locale_admin_manage_screen(), _locale_export_po(), _locale_string_save(), _locale_string_seek(), _menu_build(), cache_get(), drupal_get_path_map(), list_theme_engines(), list_themes(), module_list(), update_101(), update_97(), update_98(), and variable_init().

```
75                                  {
76   if ($result) {
77     return pg_fetch_object($result);
78   }
79 }
```

4.6.2.9 db_next_id ($ *name*)

Return a new unique ID in the given sequence.

For compatibility reasons, Drupal does not use auto-numbered fields in its database tables. Instead, this function is used to return a new unique ID of the type requested. If necessary, a new sequence with the given name will be created.

Definition at line 147 of file database.pgsql.inc.

References db_prefix_tables(), db_query(), and db_result().

Referenced by menu_rebuild().

```
147                             {
148   $id = db_result(db_query("SELECT nextval('%s_seq')", db_prefix_tables($name)));
149   return $id;
150 }
```

4.6.2.10 db_num_rows ($ *result*)

Determine how many result rows were found by the preceding query.

Parameters:

$result A database query result resource, as returned from db_query().

Returns:

The number of result rows.

Definition at line 105 of file database.pgsql.inc.

Referenced by update_101().

```
105                                {
106   if ($result) {
107     return pg_num_rows($result);
108   }
109 }
```

4.6.2.11 db_query_range ($ *query*)

Runs a limited-range query in the active database.

Use this as a substitute for db_query() when a subset of the query is to be returned. User-supplied arguments to the query should be passed in as separate parameters so that they can be properly escaped to avoid SQL injection attacks.

Parameters:

$query A string containing an SQL query.

... A variable number of arguments which are substituted into the query using printf() syntax. Instead of a variable number of query arguments, you may also pass a single array containing the query arguments.

$from The first result row to return.

$count The maximum number of result rows to return.

Returns:

A database query result resource, or FALSE if the query was not executed correctly.

Definition at line 182 of file database.pgsql.inc.

References _db_query(), db_escape_string(), and db_prefix_tables().

Referenced by pager_query().

```
182                                  {
183   $args = func_get_args();
184   $count = array_pop($args);
185   $from = array_pop($args);
186
187   $query = db_prefix_tables($query);
188   if (count(func_get_args()) > 3) {
189     // Check for array (alternative syntax).
```

```
190     if (is_array($args[1])) {
191       $args = array_merge(array($query), $args[1]);
192     }
193     $args = array_map('db_escape_string', $args);
194     $args[0] = $query;
195     $query = call_user_func_array('sprintf', $args);
196   }
197   $query .= ' LIMIT '. $count .' OFFSET '. $from;
198   return _db_query($query);
199 }
```

4.6.2.12 db_result ($ *result*, $ *row* = 0)

Return an individual result field from the previous query.

Only use this function if exactly one field is being selected; otherwise, use db_fetch_object() or db_fetch_array().

Parameters:

$result A database query result resource, as returned from db_query().

$row The index of the row whose result is needed.

Returns:

The resulting field.

Definition at line 124 of file database.pgsql.inc.

Referenced by db_next_id(), drupal_get_filename(), node_access(), pager_query(), and update_101().

```
124                                     {
125   if ($result && pg_num_rows($result) > $row) {
126     $res = pg_fetch_row($result, $row);
127 
128     return $res[0];
129   }
130 }
```

4.7 file.inc File Reference

Enumerations

- enum **IS_WINDOWS**
- enum **FILE_DOWNLOADS_PUBLIC**
- enum **FILE_DOWNLOADS_PRIVATE**
- enum **FILE_CREATE_DIRECTORY**
- enum **FILE_MODIFY_PERMISSIONS**
- enum **FILE_DIRECTORY_TEMP**
- enum **FILE_EXISTS_RENAME**
- enum **FILE_EXISTS_REPLACE**
- enum **FILE_EXISTS_ERROR**

Functions

- file_create_url ($path)
- file_create_path ($dest=0)
- file_check_directory (&$directory, $mode=0, $form_item=NULL)
- file_check_path (&$path)
- file_check_upload ($source)
- file_check_location ($source, $directory=0)
- file_copy (&$source, $dest=0, $replace=FILE_EXISTS_RENAME)
- file_move (&$source, $dest=0, $replace=FILE_EXISTS_RENAME)
- **file_create_filename** ($basename, $directory)
- **file_delete** ($path)
- file_save_upload ($source, $dest=0, $replace=FILE_EXISTS_RENAME)
- file_save_data ($data, $dest, $replace=FILE_EXISTS_RENAME)
- file_transfer ($source, $headers)
- file_download ()
- file_scan_directory ($dir, $mask, $nomask=array('.', '..', 'CVS'), $callback=0, $recurse=TRUE, $key= 'filename', $min_depth=0, $depth=0)

4.7.1 Detailed Description

API for handling file uploads and server file management.

Definition in file file.inc.

4.8 index.php File Reference

Variables

- **$status** = menu_execute_active_handler()
- **break**
- case **MENU_ACCESS_DENIED**

4.8.1 Detailed Description

The PHP page that serves all page requests on a Drupal installation.

The routines here dispatch control to the appropriate handler, which then prints the appropriate page.

Definition in file index.php.

4.9 locale.inc File Reference

Functions

- _locale_add_language ($code, $name, $onlylanguage=TRUE)
- _locale_admin_manage_screen ()
- _locale_admin_manage_add_screen ()
- _locale_admin_import_screen ()
- _locale_import_po ($file, $lang, $mode)
- _locale_import_read_po ($file)
- _locale_import_parse_header ($header)
- _locale_import_parse_plural_forms ($pluralforms, $filename)
- _locale_import_parse_arithmetic ($string)
- _locale_import_tokenize_formula ($formula)
- _locale_import_append_plural ($entry, $key)
- _locale_import_shorten_comments ($comment)
- _locale_import_parse_quoted ($string)
- _locale_admin_export_screen ()
- _locale_export_po ($language)
- _locale_export_print ($str)
- _locale_export_wrap ($str, $len)
- _locale_export_remove_plural ($entry)
- **_locale_string_delete** ($lid)
- _locale_string_save ($lid)
- _locale_string_edit ($lid)
- _locale_string_language_list ($translation)
- _locale_string_seek_query ()
- _locale_string_seek ()
- _locale_string_seek_form ()
- _locale_prepare_iso_list ()
- _locale_get_iso639_list ()

4.9.1 Detailed Description

Admin-related functions for locale.module.

Definition in file locale.inc.

4.9.2 Function Documentation

4.9.2.1 _locale_add_language ($ *code*, $ *name*, $ *onlylanguage* = TRUE)

Helper function to add a language

Definition at line 15 of file locale.inc.

References db_fetch_object(), db_query(), drupal_set_message(), theme(), url(), and watchdog().

```
15                                                                  {
16   db_query("INSERT INTO {locales_meta} (locale, name) VALUES ('%s','%s')", $code, $name);
17   $result = db_query("SELECT lid FROM {locales_source}");
18   while ($string = db_fetch_object($result)) {
19     db_query("INSERT INTO {locales_target} (lid, locale) VALUES (%d,'%s')", $string->lid, $code);
20   }
21
22   // If only the language was added, and not a PO file import triggered
23   // the language addition, we need to inform the user on how to start
24   // a translation
25   if ($onlylanguage) {
26     $message = t('%locale language added. You can now import a translation. See the <a href="%locale-he
27   }
28   else {
29     $message = t('%locale language added.', array('%locale' => theme('placeholder', t($name))));
30   }
31
32   drupal_set_message($message);
33   watchdog('locale', t('%language language (%locale) added.', array('%language' => theme('placeholder',
34 }
```

4.9.2.2 _locale_admin_export_screen ()

User interface for the translation export screen

Definition at line 695 of file locale.inc.

References form(), form_select(), and form_submit().

```
695                                       {
696   $languages = locale_supported_languages(FALSE, TRUE);
697   $languages = array_map("t", $languages['name']);
698   unset($languages['en']);
699   $output = '';
700
701   // Offer language specific export if any language is set up
702   if (count($languages)) {
703     $output .= '<h2>'. t('Export translation') .'</h2>';
704     $form = form_select(t('Language name'), 'langcode', '', $languages, t('Select the language you wou
705     $form .= form_submit(t('Export'));
706     $output .= form($form);
707   }
708
709   // Complete template export of the strings
710   $output .= '<h2>'. t('Export template') .'</h2>';
711   $form = t('<p>Generate a gettext Portable Object Template (.pot) file with all the interface strings
712   $form .= form_submit(t('Export'));
713   $output .= form($form);
714
715   return $output;
716 }
```

4.9.2.3 _locale_admin_import_screen ()

User interface for the translation import screen

Definition at line 91 of file locale.inc.

References form(), form_radios(), and form_select().

```
91                                      {
92   $languages = locale_supported_languages(FALSE, TRUE);
93   $languages = array_map("t", $languages['name']);
94   unset($languages['en']);
95
96   if (!count($languages)) {
97     $languages = _locale_prepare_iso_list();
98   }
99   else {
100    $languages = array(
101      t('Already added languages') => $languages,
102      t('Languages not yet added') => _locale_prepare_iso_list()
103    );
104  }
105
106  $form = form_file(t('Language file'), 'file', 50, t('A gettext Portable Object (.po) file.'));
107  $form .= form_select(t('Import into'), 'langcode', '', $languages, t('Choose the language you want t
108  $form .= form_radios(t('Mode'), 'mode', 'overwrite', array('overwrite' => t('Strings in the uploaded
109  $form .= form_submit(t('Import'));
110  $output = form($form, 'POST', url('admin/locale/language/import'), array('enctype' => 'multipart/for
111  return $output;
112 }
```

4.9.2.4 _locale_admin_manage_add_screen ()

User interface for the language addition screen

Definition at line 68 of file locale.inc.

References _locale_prepare_iso_list(), form(), form_select(), form_submit(), and form_textfield().

```
68                                          {
69
70   $isocodes = _locale_prepare_iso_list();
71
72   $output = '<h2>'. t('From language list') .'</h2>';
73   $form = form_select(t('Language name'), 'langcode', key($isocodes), $isocodes, t('Select your languag
74   $form .= form_submit(t('Add language'));
75   $output .= form($form);
76
77   $edit = &$_POST['edit'];
78   $output .= '<h2>'. t('Custom language') .'</h2>';
79   $form = form_textfield(t('Language code'), 'langcode', $edit['langcode'], 70, 12, t("Commonly this is
80   $form .= form_textfield(t('Language name in English'), 'langname', $edit['langname'], 70, 64, t('Name
81   $form .= form_submit(t('Add language'));
82   $output .= form($form);
83
84   return $output;
85 }
```

4.9.2.5 _locale_admin_manage_screen ()

User interface for the language management screen

Definition at line 39 of file locale.inc.

References check_plain(), db_fetch_object(), db_query(), form(), form_checkbox(), form_radio(), form_-submit(), form_textfield(), l(), message_na(), theme(), and url().

```
39                                      {
40   $edit = &$_POST['edit'];
41   $languages = locale_supported_languages(TRUE, TRUE);
42
43   $header = array(array('data' => t('Code')), array('data' => t('English name')), array('data' => t('En
44
45   foreach ($languages['name'] as $key => $lang) {
46
47     $status = db_fetch_object(db_query("SELECT isdefault, enabled FROM {locales_meta} WHERE locale = '%
48
49     if ($key == 'en') {
50       $rows[] = array('en', check_plain($lang), form_checkbox('', 'enabled][en', 1, $status->enabled),
51     }
52     else {
53       $original = db_fetch_object(db_query("SELECT COUNT(*) AS strings FROM {locales_source}"));
54       $translation = db_fetch_object(db_query("SELECT COUNT(*) AS translation FROM {locales_target} WHE
55
56       $ratio = ($original->strings > 0 && $translation->translation > 0) ? round(($translation->transla
57
58       $rows[] = array(check_plain($key), ($key != 'en' ? form_textfield('', 'name]['. $key, $lang, 15,
59     }
60   }
61
62   return form(theme('table', $header, $rows) . form_submit(t('Save configuration')), 'POST', url('admin
63 }
```

4.9.2.6 _locale_export_po ($ *language*)

Exports a Portable Object (Template) file for a language

Parameters:

$language Selects a language to generate the output for

Definition at line 723 of file locale.inc.

References _locale_export_print(), _locale_export_remove_plural(), db_fetch_object(), db_query(), theme(), variable_get(), and watchdog().

```
723                                      {
724   global $user;
725
726   // Get language specific strings, or all strings
727   if ($language) {
728     $meta = db_fetch_object(db_query("SELECT * FROM {locales_meta} WHERE locale = '%s'", $language));
729     $result = db_query("SELECT s.lid, s.source, s.location, t.translation, t.plid, t.plural FROM {loca
730   }
731   else {
732     $result = db_query("SELECT s.lid, s.source, s.location, t.plid, t.plural FROM {locales_source} s I
733   }
734
```

```
735   // Build array out of the database results
736   $parent = array();
737   while ($child = db_fetch_object($result)) {
738     if ($child->source != '') {
739       $parent[$child->lid]['comment'] = $child->location;
740       $parent[$child->lid]['msgid'] = $child->source;
741       if ($child->plid) {
742         $parent[$child->lid][$child->plid]['plural'] = $child->lid;
743         $parent[$child->lid][$child->plid]['translation'] = $child->translation;
744         $parent[$child->lid][$child->plid]['msgid'] = $child->source;
745       }
746       else {
747         $parent[$child->lid]['translation'] = $child->translation;
748       }
749     }
750   }
751
752   // Generating Portable Object file for a language
753   if ($language) {
754     $filename = $language .'.po';
755     $header .= "# $meta->name translation of ". variable_get('site_name', 'Drupal') ."\n";
756     $header .= '# Copyright (c) '. date('Y') .' '. $user->name .' <'. $user->mail .">\n";
757     $header .= "#\n";
758     $header .= "msgid \"\"\n";
759     $header .= "msgstr \"\"\n";
760     $header .= "\"Project-Id-Version: PROJECT VERSION\\n\"\n";
761     $header .= "\"POT-Creation-Date: ". date("Y-m-d H:iO") ."\\n\"\n";
762     $header .= "\"PO-Revision-Date: ". date("Y-m-d H:iO") ."\\n\"\n";
763     $header .= "\"Last-Translator: ". $user->name .' <'. $user->mail .">\\n\"\n";
764     $header .= "\"Language-Team: ". $meta->name .' <'. $user->mail .">\\n\"\n";
765     $header .= "\"MIME-Version: 1.0\\n\"\n";
766     $header .= "\"Content-Type: text/plain; charset=utf-8\\n\"\n";
767     $header .= "\"Content-Transfer-Encoding: 8bit\\n\"\n";
768     if ($meta->formula && $meta->plurals) {
769       $header .= "\"Plural-Forms: nplurals=". $meta->plurals ."; plural=". strtr($meta->formula, '$',
770     }
771     $header .= "\n";
772     watchdog('locale', t('Exported %locale translation file: %filename.', array('%locale' => theme('pl
773   }
774
775   // Generating Portable Object Template
776   else {
777     $filename = variable_get('site_name', 'drupal') .'.pot';
778     $header .= "# LANGUAGE translation of PROJECT\n";
779     $header .= "# Copyright (c) YEAR NAME <EMAIL@ADDRESS>\n";
780     $header .= "#\n";
781     $header .= "msgid \"\"\n";
782     $header .= "msgstr \"\"\n";
783     $header .= "\"Project-Id-Version: PROJECT VERSION\\n\"\n";
784     $header .= "\"POT-Creation-Date: ". date("Y-m-d H:iO") ."\\n\"\n";
785     $header .= "\"PO-Revision-Date: YYYY-mm-DD HH:MM+ZZZZ\\n\"\n";
786     $header .= "\"Last-Translator: NAME <EMAIL@ADDRESS>\\n\"\n";
787     $header .= "\"Language-Team: LANGUAGE <EMAIL@ADDRESS>\\n\"\n";
788     $header .= "\"MIME-Version: 1.0\\n\"\n";
789     $header .= "\"Content-Type: text/plain; charset=utf-8\\n\"\n";
790     $header .= "\"Content-Transfer-Encoding: 8bit\\n\"\n";
791     $header .= "\"Plural-Forms: nplurals=INTEGER; plural=EXPRESSION;\\n\"\n";
792     $header .= "\n";
793     watchdog('locale', t('Exported translation file: %filename.', array('%filename' => theme('placehol
794   }
795
796   // Start download process
797   header("Content-Disposition: attachment; filename=$filename");
798   header("Content-Type: text/plain; charset=utf-8");
799
800   print $header;
801
```

```
802   foreach ($parent as $lid => $message) {
803     if (!isset($done[$lid])) {
804       if ($message['comment']) {
805         print '#: '. $message['comment'] ."\n";
806       }
807       print 'msgid '. _locale_export_print($message['msgid']);
808       if (isset($message[1]['plural'])) {
809         print 'msgid_plural '. _locale_export_print($message[1]['msgid']);
810         if ($language) {
811           for ($i = 0; $i < $meta->plurals; $i++) {
812             print 'msgstr['. $i .'] '. _locale_export_print(_locale_export_remove_plural($message[${i}
813             $done[$message[${i}]['plural']] = 1;
814           }
815         }
816         else {
817           print 'msgstr[0] ""'. "\n";
818           print 'msgstr[1] ""'. "\n";
819           $done[$message[0]['plural']] = 1;
820           $done[$message[1]['plural']] = 1;
821         }
822       }
823       else {
824         if ($language) {
825           print 'msgstr '. _locale_export_print($message['translation']);
826         }
827         else {
828           print 'msgstr ""'. "\n";
829         }
830       }
831       print "\n";
832     }
833   }
834   die();
835 }
```

4.9.2.7 _locale_export_print ($ *str*)

Print out a string on multiple lines

Definition at line 840 of file locale.inc.

References _locale_export_wrap().

Referenced by _locale_export_po().

```
840                                     {
841   $stri = addcslashes($str, "\0..\37\\\"");
842   $parts = array();
843
844   // Cut text into several lines
845   while ($stri != "") {
846     $i = strpos($stri, "\\n");
847     if ($i === FALSE) {
848       $curstr = $stri;
849       $stri = "";
850     }
851     else {
852       $curstr = substr($stri, 0, $i + 2);
853       $stri = substr($stri, $i + 2);
854     }
855     $curparts = explode("\n", _locale_export_wrap($curstr, 70));
856     $parts = array_merge($parts, $curparts);
857   }
```

```
858
859   if (count($parts) > 1) {
860     return "\"\"\n\"". implode("\"\n\"", $parts) ."\"\n";
861   }
862   else {
863     return "\"$parts[0]\"\n";
864   }
865 }
```

4.9.2.8 _locale_export_remove_plural ($ *entry*)

Removes plural index information from a string

Definition at line 900 of file locale.inc.

Referenced by _locale_export_po().

```
900                                                 {
901   return preg_replace('/(%count)\[[0-9]\]/', '\\1', $entry);
902 }
```

4.9.2.9 _locale_export_wrap ($ *str*, $ *len*)

Custom word wrapping for Portable Object (Template) files.

Author:
Jacobo Tarrio

Definition at line 872 of file locale.inc.

Referenced by _locale_export_print().

```
872                                           {
873   $words = split(" ", $str);
874   $ret = array();
875
876   $cur = "";
877   $nstr = 1;
878   while (count($words)) {
879     $word = array_shift($words);
880     if ($nstr) {
881       $cur = $word;
882       $nstr = 0;
883     }
884     elseif (strlen("$cur $word") > $len) {
885       $ret[] = $cur . " ";
886       $cur = $word;
887     }
888     else {
889       $cur = "$cur $word";
890     }
891   }
892   $ret[] = $cur;
893
894   return implode("\n", $ret);
895 }
```

4.9.2.10 _locale_get_iso639_list ()

Some of the common languages with their English and native names

Based on ISO 639 and http://people.w3.org/rishida/names/languages.html

Definition at line 1129 of file locale.inc.

Referenced by _locale_prepare_iso_list(), and update_101().

```
1129                                    {
1130   return array(
1131     "aa" => array("Afar"),
1132     "ab" => array("Abkhazian", "ÐřÒğÑĄÑČÐř ÐśÑŃÐůÑĹÓŹÐř"),
1133     "ae" => array("Avestan"),
1134     "af" => array("Afrikaans"),
1135     "ak" => array("Akan"),
1136     "am" => array("Amharic", "áŁăáĹŻáĹ■áŁŻ"),
1137     "ar" => array("Arabic", "ØğÙĎØźØśØĺÙŁØl’"),
1138     "as" => array("Assamese"),
1139     "av" => array("Avar"),
1140     "ay" => array("Aymara"),
1141     "az" => array("Azerbaijani", "azÉŹrbaycan"),
1142     "ba" => array("Bashkir"),
1143     "be" => array("Belarusian", "ÐŚÐţÐżÐřÑĂÑČÑĄÐžÐřÑŔ"),
1144     "bg" => array("Bulgarian", "ÐŚÑŁÐżÐşÐřÑĂÑĄÐžÐÿ"),
1145     "bh" => array("Bihari"),
1146     "bi" => array("Bislama"),
1147     "bm" => array("Bambara", "Bamanankan"),
1148     "bn" => array("Bengali"),
1149     "bo" => array("Tibetan"),
1150     "br" => array("Breton"),
1151     "bs" => array("Bosnian", "Bosanski"),
1152     "ca" => array("Catalan", "CatalÃă"),
1153     "ce" => array("Chechen"),
1154     "ch" => array("Chamorro"),
1155     "co" => array("Corsican"),
1156     "cr" => array("Cree"),
1157     "cs" => array("Czech", "ÄŇeÅątina"),
1158     "cu" => array("Old Slavonic"),
1159     "cv" => array("Welsh", "Cymraeg"),
1160     "cy" => array("Welch"),
1161     "da" => array("Danish"),
1162     "de" => array("German", "Deutsch"),
1163     "dv" => array("Maldivian"),
1164     "dz" => array("Bhutani"),
1165     "ee" => array("Ewe", "ÆŘĚŃÉŻ"),
1166     "el" => array("Greek", "ÎŢÎżÎźÎůÎ¡ÎźÎžÎň"),
1167     "en" => array("English"),
1168     "eo" => array("Esperanto"),
1169     "es" => array("Spanish", "EspaÃśol"),
1170     "et" => array("Estonian", "Eesti"),
1171     "eu" => array("Basque", "Euskera"),
1172     "fa" => array("Persian", "ÙĄØğØśØşÛŇ"),
1173     "ff" => array("Fulah", "Fulfulde"),
1174     "fi" => array("Finnish", "Suomi"),
1175     "fj" => array("Fiji"),
1176     "fo" => array("Faeroese"),
1177     "fr" => array("French", "FranÃğais"),
1178     "fy" => array("Frisian", "Frysk"),
1179     "ga" => array("Irish", "Gaeilge"),
1180     "gd" => array("Scots Gaelic"),
1181     "gl" => array("Galician", "Galego"),
1182     "gn" => array("Guarani"),
1183     "gu" => array("Gujarati"),
1184     "gv" => array("Manx"),
1185     "ha" => array("Hausa"),
```

```
1186     "he" => array("Hebrew", "ŒćŒŚŒĺŒŹŒł"),
1187     "hi" => array("Hindi", "àd’źàd’£àd’ĺàě■àd’ęàěĂ"),
1188     "ho" => array("Hiri Motu"),
1189     "hr" => array("Croatian", "Hrvatski"),
1190     "hu" => array("Hungarian", "Magyar"),
1191     "hy" => array("Armenian", "ÕĂÕąÕţÕěÖĂÕěÕű"),
1192     "hz" => array("Herero"),
1193     "ia" => array("Interlingua"),
1194     "id" => array("Indonesian", "Bahasa Indonesia"),
1195     "ie" => array("Interlingue"),
1196     "ig" => array("Igbo"),
1197     "ik" => array("Inupiak"),
1198     "is" => array("Icelandic", "Ã■slenska"),
1199     "it" => array("Italian", "Italiano"),
1200     "iu" => array("Inuktitut"),
1201     "ja" => array("Japanese", "æŮěæIJňèłđ"),
1202     "jv" => array("Javanese"),
1203     "ka" => array("Georgian"),
1204     "kg" => array("Kongo"),
1205     "ki" => array("Kikuyu"),
1206     "kj" => array("Kwanyama"),
1207     "kk" => array("Kazakh", "ÒŽĐřĐůĐřÒŻ"),
1208     "kl" => array("Greenlandic"),
1209     "km" => array("Cambodian"),
1210     "kn" => array("Kannada", "àšŢàšĺąş■àšĺàšą"),
1211     "ko" => array("Korean", "íŢIJêţ■ìŰt’"),
1212     "kr" => array("Kanuri"),
1213     "ks" => array("Kashmiri"),
1214     "ku" => array("Kurdish", "KurdÃő"),
1215     "kv" => array("Komi"),
1216     "kw" => array("Cornish"),
1217     "ky" => array("Kirghiz", "ĐŽÑŃÑĂĐşÑŃĐů"),
1218     "la" => array("Latin", "Latina"),
1219     "lb" => array("Luxembourgish"),
1220     "lg" => array("Luganda"),
1221     "ln" => array("Lingala"),
1222     "lo" => array("Laothian"),
1223     "lt" => array("Lithuanian", "LietuviÅąkai"),
1224     "lv" => array("Latvian", "LatvieÅąu"),
1225     "mg" => array("Malagasy"),
1226     "mh" => array("Marshallese"),
1227     "mi" => array("Maori"),
1228     "mk" => array("Macedonian", "ĐIJĐřĐžĐţĐt’Đ¿Đ¡ÑĄĐžĐÿ"),
1229     "ml" => array("Malayalam", "àt’őàt’šàt’ŕàt’¿àt’şàt’Ć"),
1230     "mn" => array("Mongolian"),
1231     "mo" => array("Moldavian"),
1232     "mr" => array("Marathi"),
1233     "ms" => array("Malay", "Bahasa Melayu"),
1234     "mt" => array("Maltese", "Malti"),
1235     "my" => array("Burmese"),
1236     "na" => array("Nauru"),
1237     "nd" => array("North Ndebele"),
1238     "ne" => array("Nepali"),
1239     "ng" => array("Ndonga"),
1240     "nl" => array("Dutch", "Nederlands"),
1241     "no" => array("Norwegian", "Norsk"),
1242     "nr" => array("South Ndebele"),
1243     "nv" => array("Navajo"),
1244     "ny" => array("Chichewa"),
1245     "oc" => array("Occitan"),
1246     "om" => array("Oromo"),
1247     "or" => array("Oriya"),
1248     "os" => array("Ossetian"),
1249     "pa" => array("Punjabi"),
1250     "pi" => array("Pali"),
1251     "pl" => array("Polish", "Polski"),
1252     "ps" => array("Pashto", "Ù¿ÚŽØłÙĹ"),
```

```
1253     "pt" => array("Portuguese", "PortuguÃłs"),
1254     "qu" => array("Quechua"),
1255     "rm" => array("Rhaeto-Romance"),
1256     "rn" => array("Kirundi"),
1257     "ro" => array("Romanian", "RomÃćnÄČ"),
1258     "ru" => array("Russian", "ĐăÑČÑąÑąĐžĐÿĐź"),
1259     "rw" => array("Kinyarwanda"),
1260     "sa" => array("Sanskrit"),
1261     "sc" => array("Sardinian"),
1262     "sd" => array("Sindhi"),
1263     "se" => array("Northern Sami"),
1264     "sg" => array("Sango"),
1265     "sh" => array("Serbo-Croatian"),
1266     "si" => array("Singhalese"),
1267     "sk" => array("Slovak", "SlovenÄ■ina"),
1268     "sl" => array("Slovenian", "SlovenÅąÄ■ina"),
1269     "sm" => array("Samoan"),
1270     "sn" => array("Shona"),
1271     "so" => array("Somali"),
1272     "sq" => array("Albanian", "Shqip"),
1273     "sr" => array("Serbian", "ĐąÑĂĐ£ÑąĐžĐÿ"),
1274     "ss" => array("Siswati"),
1275     "st" => array("Sesotho"),
1276     "su" => array("Sudanese"),
1277     "sv" => array("Swedish", "Svenska"),
1278     "sw" => array("Swahili", "Kiswahili"),
1279     "ta" => array("Tamil", "àõd’àõőàõ£àõt’àŕ■"),
1280     "te" => array("Telugu", "àřd’àśĘàřšàśĄàřŮàśĄ"),
1281     "tg" => array("Tajik"),
1282     "th" => array("Thai", "àÿăàÿšàÿl’àÿšàźĎàÿŮàÿć"),
1283     "ti" => array("Tigrinya"),
1284     "tk" => array("Turkmen"),
1285     "tl" => array("Tagalog"),
1286     "tn" => array("Setswana"),
1287     "to" => array("Tonga"),
1288     "tr" => array("Turkish", "TÃijrkÃğe"),
1289     "ts" => array("Tsonga"),
1290     "tt" => array("Tatar", "TatarÃğa"),
1291     "tw" => array("Twi"),
1292     "ty" => array("Tahitian"),
1293     "ug" => array("Uighur"),
1294     "uk" => array("Ukrainian", "ĐčĐžÑĂĐřÑŮĐ¡ÑąÑŃĐžĐř"),
1295     "ur" => array("Urdu", "ØğØśØŕÙĹ"),
1296     "uz" => array("Uzbek", "o’zbek"),
1297     "ve" => array("Venda"),
1298     "vi" => array("Vietnamese", "Tiáž£ng ViážĞt"),
1299     "vo" => array("VolapÃijk"),
1300     "wo" => array("Wolof"),
1301     "xh" => array("Xhosa", "isiXhosa"),
1302     "yi" => array("Yiddish"),
1303     "yo" => array("Yoruba", "YorÃźbÃą"),
1304     "za" => array("Zhuang"),
1305     "zh-hans" => array("Chinese, Simplified", "ÇőĂä¡Şäÿ■ŒŰĞ"),
1306     "zh-hant" => array("Chinese, Traditional", "ÇźĄéńŤäÿ■ŒŰĞ"),
1307     "zu" => array("Zulu", "isiZulu"),
1308   );
1309 }
```

4.9.2.11 _locale_import_append_plural ($ *entry*, $ *key*)

Modify a string to contain proper count indices

This is a callback function used via array_map()

Parameters:

$entry An array element

$key Index of the array element

Definition at line 644 of file locale.inc.

```
644                                                 {
645   // No modifications for 0, 1
646   if ($key == 0 || $key == 1) {
647     return $entry;
648   }
649
650   // First remove any possibly false indices, then add new ones
651   $entry = preg_replace('/(%count)\[[0-9]\]/', '\\1', $entry);
652   return preg_replace('/(%count)/', "\\1[$key]", $entry);
653 }
```

4.9.2.12 _locale_import_parse_arithmetic ($ *string*)

Parses and sanitizes an arithmetic formula into a PHP expression

While parsing, we ensure, that the operators have the right precedence and associativity.

Parameters:

$string A string containing the arithmetic formula

Returns:

The PHP version of the formula

Author:

Jacobo Tarrio

Definition at line 484 of file locale.inc.

References _locale_import_tokenize_formula().

Referenced by _locale_import_parse_plural_forms().

```
484                                                 {
485   // Operator precedence table
486   $prec = array("(" => -1, ")" => -1, "?" => 1, ":" => 1, "||" => 3, "&&" => 4, "==" => 5, "!=" => 5,
487   // Right associativity
488   $rasc = array("?" => 1, ":" => 1);
489
490   $tokens = _locale_import_tokenize_formula($string);
491
492   // Parse by converting into infix notation then back into postfix
493   $opstk = array();
494   $elstk = array();
495
496   foreach ($tokens as $token) {
497     $ctok = $token;
498
499     // Numbers and the $n variable are simply pushed into $elarr
500     if (is_numeric($token)) {
501       $elstk[] = $ctok;
502     }
```

```
503     elseif ($ctok == "n") {
504       $elstk[] = ’$n’;
505     }
506     elseif ($ctok == "(") {
507       $opstk[] = $ctok;
508     }
509     elseif ($ctok == ")") {
510       $topop = array_pop($opstk);
511       while (($topop != NULL) && ($topop != "(")) {
512         $elstk[] = $topop;
513         $topop = array_pop($opstk);
514       }
515     }
516     elseif ($prec[$ctok]) {
517       // If it’s an operator, then pop from $oparr into $elarr until the
518       // precedence in $oparr is less than current, then push into $oparr
519       $topop = array_pop($opstk);
520       while (($topop != NULL) && ($prec[$topop] >= $prec[$ctok]) && !(($prec[$topop] == $prec[$ctok])
521         $elstk[] = $topop;
522         $topop = array_pop($opstk);
523       }
524       if ($topop) {
525         $opstk[] = $topop;   // Return element to top
526       }
527       $opstk[] = $ctok;      // Parentheses are not needed
528     }
529     else {
530       return false;
531     }
532   }
533
534   // Flush operator stack
535   $topop = array_pop($opstk);
536   while ($topop != NULL) {
537     $elstk[] = $topop;
538     $topop = array_pop($opstk);
539   }
540
541   // Now extract formula from stack
542   $prevsize = count($elstk) + 1;
543   while (count($elstk) < $prevsize) {
544     $prevsize = count($elstk);
545     for ($i = 2; $i < count($elstk); $i++) {
546       $op = $elstk[$i];
547       if ($prec[$op]) {
548         $f = "";
549         if ($op == ":") {
550           $f = $elstk[$i - 2] .”):”. $elstk[$i - 1] .”)”;
551         }
552         elseif ($op == "?") {
553           $f = "(". $elstk[$i - 2] ."?(". $elstk[$i - 1];
554         }
555         else {
556           $f = "(". $elstk[$i - 2] . $op . $elstk[$i - 1] .")";
557         }
558         array_splice($elstk, $i - 2, 3, $f);
559         break;
560       }
561     }
562   }
563
564   // If only one element is left, the number of operators is appropriate
565   if (count($elstk) == 1) {
566     return $elstk[0];
567   }
568   else {
569     return FALSE;
```

```
570   }
571 }
```

4.9.2.13 _locale_import_parse_header ($ *header*)

Parses a Gettext Portable Object file header

Parameters:

$header A string containing the complete header

Returns:

An associative array of key-value pairs

Author:

Jacobo Tarrio

Definition at line 418 of file locale.inc.

```
418                                                 {
419   $hdr = array();
420
421   $lines = explode("\n", $header);
422   foreach ($lines as $line) {
423     $line = trim($line);
424     if ($line) {
425       list($tag, $contents) = explode(":", $line, 2);
426       $hdr[trim($tag)] = trim($contents);
427     }
428   }
429
430   return $hdr;
431 }
```

4.9.2.14 _locale_import_parse_plural_forms ($ *pluralforms*, $ *filename*)

Parses a Plural-Forms entry from a Gettext Portable Object file header

Parameters:

$pluralforms A string containing the Plural-Forms entry

$filename A string containing the filename

Returns:

An array containing the number of plurals and a formula in PHP for computing the plural form

Author:

Jacobo Tarrio

Definition at line 442 of file locale.inc.

References _locale_import_parse_arithmetic(), drupal_set_message(), and theme().

```
442                                                                    {
443   // First, delete all whitespace
444   $pluralforms = strtr($pluralforms, array(" " => "", "\t" => ""));
445
446   // Select the parts that define nplurals and plural
447   $nplurals = strstr($pluralforms, "nplurals=");
448   if (strpos($nplurals, ";")) {
449     $nplurals = substr($nplurals, 9, strpos($nplurals, ";") - 9);
450   }
451   else {
452     return FALSE;
453   }
454   $plural = strstr($pluralforms, "plural=");
455   if (strpos($plural, ";")) {
456     $plural = substr($plural, 7, strpos($plural, ";") - 7);
457   }
458   else {
459     return FALSE;
460   }
461
462   // Get PHP version of the plural formula
463   $plural = _locale_import_parse_arithmetic($plural);
464
465   if ($plural) {
466     return array($nplurals, $plural);
467   }
468   else {
469     drupal_set_message(t("Translation file %filename broken: plural formula couldn't get parsed.", arr
470     return FALSE;
471   }
472 }
```

4.9.2.15 _locale_import_parse_quoted ($ *string*)

Parses a string in quotes

Parameters:

$string A string specified with enclosing quotes

Returns:

The string parsed from inside the quotes

Definition at line 675 of file locale.inc.

Referenced by _locale_import_read_po().

```
675                                                {
676   if (substr($string, 0, 1) != substr($string, -1, 1)) {
677     return FALSE;   // Start and end quotes must be the same
678   }
679   $quote = substr($string, 0, 1);
680   $string = substr($string, 1, -1);
681   if ($quote == '"') {        // Double quotes: strip slashes
682     return stripcslashes($string);
683   }
684   elseif ($quote == "'") {  // Simple quote: return as-is
685     return $string;
686   }
687   else {
688     return FALSE;             // Unrecognized quote
```

```
689   }
690 }
```

4.9.2.16 _locale_import_po ($ *file*, $ *lang*, $ *mode*)

Parses Gettext Portable Object file information and inserts into database

Parameters:

$file Object contains properties of local file to be imported

$edit Language code

$mode should existing translations be replaced?

Definition at line 121 of file locale.inc.

```
121                                                 {
122   // If not in 'safe mode', increase the maximum execution time:
123   if (!ini_get('safe_mode')) {
124     set_time_limit(240);
125   }
126
127   // Check if we have the language already in the database
128   if (!db_fetch_object(db_query("SELECT locale FROM {locales_meta} WHERE locale = '%s'", $lang))) {
129     drupal_set_message(t('Unsupported language selected for import.'), 'error');
130     return FALSE;
131   }
132
133   // Check if we can get the strings from the file
134   if (!($strings = _locale_import_read_po($file))) {
135     drupal_set_message(t('Translation file %filename broken: Could not be read.', array('%filename' =>
136     return FALSE;
137   }
138
139   // Strip out header from the string pairs
140   $header = $strings[""]["msgstr"];
141   unset($strings[""]);
142
143   // Get information from the header into the database
144   if ($header) {
145     $hdr = _locale_import_parse_header($header);
146
147     // Get the plural formula
148     if ($hdr["Plural-Forms"] && $p = _locale_import_parse_plural_forms($hdr["Plural-Forms"], $file->fi
149       list($nplurals, $plural) = $p;
150       db_query("UPDATE {locales_meta} SET plurals = '%d', formula = '%s' WHERE locale = '%s'", $nplura
151     }
152     else {
153       db_query("UPDATE {locales_meta} SET plurals = '%d', formula = '%s' WHERE locale = '%s'", 0, '',
154     }
155   }
156   else {
157     drupal_set_message(t('Translation file %filename broken: No header.', array('%filename' => theme('
158     return FALSE;
159   }
160
161   $additions = 0;
162   $updates = 0;
163   foreach ($strings as $value) {
164     $comments = _locale_import_shorten_comments($value['#']);
165
```

```
166     // Handle a translation for some plural string
167     if (strpos($value['msgid'], "\0")) {
168       $english = explode("\0", $value['msgid'], 2);
169       $entries = array_keys($value['msgstr']);
170       for ($i = 3; $i <= count($entries); $i++) {
171         $english[] = $english[1];
172       }
173       $translation = array_map('_locale_import_append_plural', $value['msgstr'], $entries);
174       $english = array_map('_locale_import_append_plural', $english, $entries);
175       foreach ($translation as $key => $trans) {
176         if ($key == 0) {
177           $plid = 0;
178         }
179         $loc = db_fetch_object(db_query("SELECT lid FROM {locales_source} WHERE source = '%s'", $engli
180         if ($loc->lid) { // a string exists
181           $lid = $loc->lid;
182           // update location field
183           db_query("UPDATE {locales_source} SET location = '%s' WHERE lid = %d", $comments, $lid);
184           $trans2 = db_fetch_object(db_query("SELECT lid, translation, plid, plural FROM {locales_targ
185           if (!$trans2->lid) { // no translation in current language
186             db_query("INSERT INTO {locales_target} (lid, locale, translation, plid, plural) VALUES (%d
187             $additions++;
188           } // translation exists
189           else if ($mode == 'overwrite' || $trans2->translation == '') {
190             db_query("UPDATE {locales_target} SET translation = '%s', plid = %d, plural = %d WHERE loc
191             if ($trans2->translation == '') {
192               $additions++;
193             }
194             else {
195               $updates++;
196             }
197           }
198         }
199         else { // no string
200           db_query("INSERT INTO {locales_source} (location, source) VALUES ('%s', '%s')", $comments, $
201           $loc = db_fetch_object(db_query("SELECT lid FROM {locales_source} WHERE source = '%s'", $eng
202           $lid = $loc->lid;
203           db_query("INSERT INTO {locales_target} (lid, locale, translation, plid, plural) VALUES (%d,
204           if ($trans != '') {
205             $additions++;
206           }
207         }
208         $plid = $lid;
209       }
210     }
211
212     // A simple translation
213     else {
214       $english = $value['msgid'];
215       $translation = $value['msgstr'];
216       $loc = db_fetch_object(db_query("SELECT lid FROM {locales_source} WHERE source = '%s'", $english
217       if ($loc->lid) { // a string exists
218         $lid = $loc->lid;
219         // update location field
220         db_query("UPDATE {locales_source} SET location = '%s' WHERE source = '%s'", $comments, $englis
221         $trans = db_fetch_object(db_query("SELECT lid, translation FROM {locales_target} WHERE lid = %
222         if (!$trans->lid) { // no translation in current language
223           db_query("INSERT INTO {locales_target} (lid, locale, translation) VALUES (%d, '%s', '%s')",
224           $additions++;
225         } // translation exists
226         else if ($mode == 'overwrite') { //overwrite in any case
227           db_query("UPDATE {locales_target} SET translation = '%s' WHERE locale = '%s' AND lid = %d",
228           if ($trans->translation == '') {
229             $additions++;
230           }
231           else {
232             $updates++;
```

```
233           }
234         } // overwrite if empty string
235         else if ($trans->translation == '') {
236           db_query("UPDATE {locales_target} SET translation = '%s' WHERE locale = '%s' AND lid = %d",
237           $additions++;
238         }
239       }
240       else { // no string
241         db_query("INSERT INTO {locales_source} (location, source) VALUES ('%s', '%s')", $comments, $en
242         $loc = db_fetch_object(db_query("SELECT lid FROM {locales_source} WHERE source = '%s'", $engli
243         $lid = $loc->lid;
244         db_query("INSERT INTO {locales_target} (lid, locale, translation) VALUES (%d, '%s', '%s')", $l
245         if ($translation != '') {
246           $additions++;
247         }
248       }
249     }
250   }
251
252   // Successful import
253   // rebuild locale cache
254   cache_clear_all("locale:$lang");
255
256   // rebuild the menu, strings may have changed
257   menu_rebuild();
258
259   drupal_set_message(t('Translation successfully imported. %number translated strings added to languag
260   watchdog('locale', t('Imported %file into %locale: %number new strings added and %update updated.',
261   return TRUE;
262 }
```

4.9.2.17 _locale_import_read_po ($ *file*)

Parses Gettext Portable Object file into an array

Parameters:

$file Object with properties of local file to parse

Author:

Jacobo Tarrio

Definition at line 270 of file locale.inc.

References _locale_import_parse_quoted(), drupal_set_message(), and theme().

```
270                                       {
271
272   $message = theme('placeholder', $file->filename);
273   $fd = fopen($file->filepath, "rb");
274   if (!$fd) {
275     drupal_set_message(t('Translation import failed: file %filename cannot be read.', array('%filename
276     return FALSE;
277   }
278   $info = fstat($fd);
279   $len = $info["size"];
280   $po = fread($fd, $len);
281   fclose($fd);
282
283   $context = "COMMENT"; // Parser context: COMMENT, MSGID, MSGID_PLURAL, MSGSTR and MSGSTR_ARR
284   $current = array();   // Current entry being read
```

```
285   $strings = array();   // List of entries read
286   $plural = 0;          // Current plural form
287
288   $po = strtr($po, array("\\\n" => ""));
289   $lines = split("\n", $po);
290   $lineno = 0;
291
292   foreach ($lines as $line) {
293     $lineno++;
294     $line = trim($line);
295
296     if (!strncmp("#", $line, 1)) { // A comment
297       if ($context == "COMMENT") { // Already in comment context: add
298         $current["#"][] = substr($line, 1);
299       }
300       elseif (($context == "MSGSTR") || ($context == "MSGSTR_ARR")) { // End current entry, start a ne
301         $strings[$current["msgid"]] = $current;
302         $current = array();
303         $current["#"][] = substr($line, 1);
304         $context = "COMMENT";
305       }
306       else { // Parse error
307         drupal_set_message(t("Translation file %filename broken: expected 'msgstr' in line %line.", ar
308         return FALSE;
309       }
310     }
311     elseif (!strncmp("msgid_plural", $line, 12)) {
312       if ($context != "MSGID") { // Must be plural form for current entry
313         drupal_set_message(t("Translation file %filename broken: unexpected 'msgid_plural' in line %li
314         return FALSE;
315       }
316       $line = trim(substr($line, 12));
317       $quoted = _locale_import_parse_quoted($line);
318       if ($quoted === false) {
319         drupal_set_message(t('Translation file %filename broken: syntax error in line %line.', array('
320         return FALSE;
321       }
322       $current["msgid"] = $current["msgid"] ."\0". $quoted;
323       $context = "MSGID_PLURAL";
324     }
325     elseif (!strncmp("msgid", $line, 5)) {
326       if ($context == "MSGSTR") {   // End current entry, start a new one
327         $strings[$current["msgid"]] = $current;
328         $current = array();
329       }
330       elseif ($context == "MSGID") { // Already in this context? Parse error
331         drupal_set_message(t("Translation file %filename broken: unexpected 'msgid' in line %line.", a
332         return FALSE;
333       }
334       $line = trim(substr($line, 5));
335       $quoted = _locale_import_parse_quoted($line);
336       if ($quoted === false) {
337         drupal_set_message(t('Translation file %filename broken: syntax error in line %line.', array('
338         return FALSE;
339       }
340       $current["msgid"] = $quoted;
341       $context = "MSGID";
342     }
343     elseif (!strncmp("msgstr[", $line, 7)) {
344       if (($context != "MSGID") && ($context != "MSGID_PLURAL") && ($context != "MSGSTR_ARR")) { // Mu
345         drupal_set_message(t("Translation file %filename broken: unexpected 'msgstr[]' in line %line."
346         return FALSE;
347       }
348       if (strpos($line, "]") === false) {
349         drupal_set_message(t('Translation file %filename broken: syntax error in line %line.', array('
350         return FALSE;
351       }
```

```
352       $frombracket = strstr($line, "[");
353       $plural = substr($frombracket, 1, strpos($frombracket, "]") - 1);
354       $line = trim(strstr($line, " "));
355       $quoted = _locale_import_parse_quoted($line);
356       if ($quoted === false) {
357         drupal_set_message(t('Translation file %filename broken: syntax error in line %line.', array('
358         return FALSE;
359       }
360       $current["msgstr"][$plural] = $quoted;
361       $context = "MSGSTR_ARR";
362     }
363     elseif (!strncmp("msgstr", $line, 6)) {
364       if ($context != "MSGID") {   // Should come just after a msgid block
365         drupal_set_message(t("Translation file %filename broken: unexpected 'msgstr' in line %line.",
366         return FALSE;
367       }
368       $line = trim(substr($line, 6));
369       $quoted = _locale_import_parse_quoted($line);
370       if ($quoted === false) {
371         drupal_set_message(t('Translation file %filename broken: syntax error in line %line.', array('
372         return FALSE;
373       }
374       $current["msgstr"] = $quoted;
375       $context = "MSGSTR";
376     }
377     elseif ($line != "") {
378       $quoted = _locale_import_parse_quoted($line);
379       if ($quoted === false) {
380         drupal_set_message(t('Translation file %filename broken: syntax error in line %line.', array('
381         return FALSE;
382       }
383       if (($context == "MSGID") || ($context == "MSGID_PLURAL")) {
384         $current["msgid"] .= $quoted;
385       }
386       elseif ($context == "MSGSTR") {
387         $current["msgstr"] .= $quoted;
388       }
389       elseif ($context == "MSGSTR_ARR") {
390         $current["msgstr"][$plural] .= $quoted;
391       }
392       else {
393         drupal_set_message(t('Translation file %filename broken: unexpected string in line %line.', ar
394         return FALSE;
395       }
396     }
397   }
398
399   // End of PO file, flush last entry
400   if (($context == "MSGSTR") || ($context == "MSGSTR_ARR")) {
401     $strings[$current["msgid"]] = $current;
402   }
403   elseif ($context != "COMMENT") {
404     drupal_set_message(t('Translation file %filename broken: unexpected end of file at line %line.', a
405     return FALSE;
406   }
407
408   return $strings;
409 }
```

4.9.2.18 _locale_import_shorten_comments ($ *comment*)

Generate a short, one string version of the passed comment array

Parameters:
$comment An array of strings containing a comment

Returns:
Short one string version of the comment

Definition at line 661 of file locale.inc.

```
661                                                 {
662   $comm = '';
663   while(strlen($comm) < 128 && count($comment)) {
664     $comm .= substr(array_shift($comment), 1) .', ';
665   }
666   return substr($comm, 0, -2);
667 }
```

4.9.2.19 _locale_import_tokenize_formula ($ *formula*)

Backward compatible implementation of token_get_all() for formula parsing

Parameters:
$string A string containing the arithmetic formula

Returns:
The PHP version of the formula

Author:
Gerhard Killesreiter

Definition at line 580 of file locale.inc.

Referenced by _locale_import_parse_arithmetic().

```
580                                                 {
581   $formula = str_replace(" ", "", $formula);
582   $tokens = array();
583   for ($i = 0; $i < strlen($formula); $i++) {
584     if (is_numeric($formula{$i})) {
585       $num = $formula{$i};
586       $j = $i + 1;
587       while($j < strlen($formula) && is_numeric($formula{$j})) {
588         $num .= $formula{$j};
589         $j++;
590       }
591       $i = $j - 1;
592       $tokens[] = $num;
593     }
594     elseif ($pos = strpos(" =<>!&|", $formula{$i})) { // We won't have a space
595       $next = $formula{($i+1)};
596       switch ($pos) {
597         case 1:
598         case 2:
599         case 3:
600         case 4:
601           if ($next == '=') {
602             $tokens[] = $formula{$i} .'=';
603             $i++;
```

```
604           }
605           else {
606             $tokens[] = $formula{$i};
607           }
608           break;
609         case 5:
610           if ($next == '&') {
611             $tokens[] = '&&';
612             $i++;
613           }
614           else {
615             $tokens[] = $formula{$i};
616           }
617           break;
618         case 6:
619           if ($next == '|') {
620             $tokens[] = '||';
621             $i++;
622           }
623           else {
624             $tokens[] = $formula{$i};
625           }
626           break;
627       }
628     }
629     else {
630       $tokens[] = $formula{$i};
631     }
632   }
633   return $tokens;
634 }
```

4.9.2.20 _locale_prepare_iso_list ()

Prepares the language code list for a select form item with only the unsupported ones

Definition at line 1104 of file locale.inc.

References _locale_get_iso639_list().

Referenced by _locale_admin_manage_add_screen().

```
1104                                    {
1105   $languages = locale_supported_languages(FALSE, TRUE);
1106   $isocodes = _locale_get_iso639_list();
1107   foreach ($isocodes as $key => $value) {
1108     if (isset($languages['name'][$key])) {
1109       unset($isocodes[$key]);
1110       continue;
1111     }
1112     if (count($value) == 2) {
1113       $tname = t($value[0]);
1114       $isocodes[$key] = ($tname == $value[1]) ? $tname : "$tname ($value[1])";
1115     }
1116     else {
1117       $isocodes[$key] = t($value[0]);
1118     }
1119   }
1120   asort($isocodes);
1121   return $isocodes;
1122 }
```

4.9.2.21 _locale_string_edit ($ *lid*)

User interface for string editing

Definition at line 941 of file locale.inc.

References db_query().

```
941                                   {
942   $languages = locale_supported_languages(FALSE, TRUE);
943   unset($languages['name']['en']);
944
945   $result = db_query('SELECT DISTINCT s.source, t.translation, t.locale FROM {locales_source} s INNER
946   $form = '';
947   while ($translation = db_fetch_object($result)) {
948     $orig = $translation->source;
949     $form .= (strlen($orig) > 40) ? form_textarea($languages['name'][$translation->locale], $translati
950     unset($languages['name'][$translation->locale]);
951   }
952   foreach ($languages['name'] as $key => $lang) {
953     $form .= (strlen($orig) > 40) ? form_textarea($lang, $key, '', 70, 15) : form_textfield($lang, $ke
954   }
955   $form = form_item(t('Original text'), wordwrap(check_plain($orig, 0))) . $form;
956
957   $form .= form_submit(t('Save translations'));
958
959   return form($form);
960 }
```

4.9.2.22 _locale_string_language_list ($ *translation*)

List languages in search result table

Definition at line 965 of file locale.inc.

Referenced by _locale_string_seek().

```
965                                                   {
966   $languages = locale_supported_languages(FALSE, TRUE);
967   unset($languages['name']['en']);
968   $output = '';
969   foreach ($languages['name'] as $key => $value) {
970     if (isset($translation[$key])) {
971       $output .= ($translation[$key] != '') ? $key .' ' : "<strike>$key</strike> ";
972     }
973   }
974
975   return $output;
976 }
```

4.9.2.23 _locale_string_save ($ *lid*)

Action handler for string editing

Saves all translations of one string submitted from a form

Definition at line 916 of file locale.inc.

References db_fetch_object(), db_query(), drupal_set_message(), and menu_rebuild().

```
916                                 {
917
918   $edit =& $_POST['edit'];
919   foreach ($edit as $key => $value) {
920     $trans = db_fetch_object(db_query("SELECT translation FROM {locales_target} WHERE lid = %d AND loc
921     if (isset($trans->translation)) {
922       db_query("UPDATE {locales_target} SET translation = '%s' WHERE lid = %d AND locale = '%s'", $val
923     }
924     else {
925       db_query("INSERT INTO {locales_target}  (lid, translation, locale) VALUES (%d, '%s', '%s')", $li
926     }
927   }
928   // refresh the locale cache
929   locale_refresh_cache();
930   // rebuild the menu, strings may have changed
931   menu_rebuild();
932   // delete form data so it will remember where it came from
933   $edit = '';
934
935   drupal_set_message(t('Saved string'));
936 }
```

4.9.2.24 _locale_string_seek ()

Perform a string search and display results in a table

Definition at line 1008 of file locale.inc.

References _locale_string_language_list(), _locale_string_seek_query(), db_escape_string(), db_fetch_-object(), l(), pager_query(), and theme().

```
1008                                {
1009   // We have at least one criterion to match
1010   if ($query = _locale_string_seek_query()) {
1011     $join = "SELECT s.source, s.location, s.lid, t.translation, t.locale FROM {locales_source} s INNE
1012
1013     // Compute LIKE section
1014     switch ($query->searchin) {
1015       case 'translated':
1016         $where = "WHERE (t.translation LIKE '%". db_escape_string($query->string) ."%' AND t.translat
1017         $orderby = "ORDER BY t.translation";
1018         break;
1019       case 'untranslated':
1020         $where = "WHERE (s.source LIKE '%". db_escape_string($query->string) ."%' AND t.translation =
1021         $orderby = "ORDER BY s.source";
1022         break;
1023       case 'all' :
1024       default:
1025         $where = "WHERE (s.source LIKE '%". db_escape_string($query->string) ."%' OR t.translation LI
1026         $orderby = '';
1027         break;
1028     }
1029
1030     switch ($query->language) {
1031       // Force search in source strings
1032       case "en":
1033         $sql = $join ." WHERE s.source LIKE '%". db_escape_string($query->string) ."%' ORDER BY s.sou
1034         break;
1035       // Search in all languages
1036       case "all":
1037         $sql = "$join $where $orderby";
1038         break;
```

```
1039       // Some different language
1040       default:
1041         $sql = "$join $where AND t.locale = '". db_escape_string($query->language) ."' $orderby";
1042     }
1043
1044     $result = pager_query($sql, 50);
1045
1046     $header = array(t('String'), t('Locales'), array('data' => t('Operations'), 'colspan' => '2'));
1047     $arr = array();
1048     while ($locale = db_fetch_object($result)) {
1049       $arr[$locale->lid]['locales'][$locale->locale] = $locale->translation;
1050       $arr[$locale->lid]['location'] = $locale->location;
1051       $arr[$locale->lid]['source'] = $locale->source;
1052     }
1053     foreach ($arr as $lid => $value) {
1054       $source = htmlspecialchars($value['source']);
1055       $rows[] = array(array('data' => (strlen($source) > 150 ? substr($source, 0, 150) .'...' : $sour
1056     }
1057
1058     $request = array();
1059     if (count($query)) {
1060       foreach ($query as $key => $value) {
1061         $request[$key] = (is_array($value)) ? implode(',', $value) : $value;
1062       }
1063     }
1064
1065     if ($pager = theme('pager', NULL, 50, 0, $request)) {
1066       $rows[] = array(array('data' => "$pager", 'colspan' => '5'));
1067     }
1068
1069     $output .= theme('table', $header, $rows);
1070
1071   }
1072
1073   return $output;
1074 }
```

4.9.2.25 _locale_string_seek_form ()

User interface for the string search screen

Definition at line 1079 of file locale.inc.

References _locale_string_seek_query(), check_plain(), form(), form_group(), form_radios(), form_submit(), form_textfield(), and url().

```
1079                                    {
1080
1081   // Get *all* languages set up
1082   $languages = locale_supported_languages(FALSE, TRUE);
1083   asort($languages['name']); unset($languages['name']['en']);
1084   $languages['name'] = array_map('check_plain', $languages['name']);
1085
1086   // Present edit form preserving previous user settings
1087   $query = _locale_string_seek_query();
1088   $form .= form_textfield(t('Strings to search for'), 'string', $query->string, 30, 30, t('Leave blan
1089   $form .= form_radios(t('Language'), 'language', ($query->language ? $query->language : 'all'), arra
1090   $form .= form_radios(t('Search in'), 'searchin', ($query->searchin ? $query->searchin : 'all'), arr
1091
1092   $form .= form_submit(t('Search'));
1093   $output = form(form_group(t('Search strings'), $form), 'POST', url('admin/locale/string/search'));
1094
```

```
1095   return $output;
1096 }
```

4.9.2.26 _locale_string_seek_query ()

Build object out of search criteria specified in request variables

Definition at line 981 of file locale.inc.

Referenced by _locale_string_seek(), and _locale_string_seek_form().

```
981                                     {
982   static $query = NULL;
983
984   if (is_null($query)) {
985     $fields = array('string', 'language', 'searchin');
986     $query = new StdClass;
987     if (is_array($_REQUEST['edit'])) {
988       foreach ($_REQUEST['edit'] as $key => $value) {
989         if (!empty($value) && in_array($key, $fields)) {
990           $query->$key = $value;
991         }
992       }
993     }
994     else {
995       foreach ($_REQUEST as $key => $value) {
996         if (!empty($value) && in_array($key, $fields)) {
997           $query->$key = strpos(',', $value) ? explode(',', $value) : $value;
998         }
999       }
1000     }
1001   }
1002   return $query;
1003 }
```

4.10 menu.inc File Reference

Menu flags

Flags for use in the "type" attribute of menu items.

- enum **MENU_IS_ROOT**
- enum **MENU_VISIBLE_IN_TREE**
- enum **MENU_VISIBLE_IN_BREADCRUMB**
- enum **MENU_VISIBLE_IF_HAS_CHILDREN**
- enum **MENU_MODIFIABLE_BY_ADMIN**
- enum **MENU_MODIFIED_BY_ADMIN**
- enum **MENU_CREATED_BY_ADMIN**
- enum **MENU_IS_LOCAL_TASK**
- enum **MENU_EXPANDED**
- enum **MENU_LINKS_TO_PARENT**

Menu item types

Menu item definitions provide one of these constants, which are shortcuts for combinations of the above flags.

- enum MENU_NORMAL_ITEM
- enum MENU_ITEM_GROUPING
- enum MENU_CALLBACK
- enum MENU_DYNAMIC_ITEM
- enum MENU_SUGGESTED_ITEM
- enum MENU_LOCAL_TASK
- enum MENU_DEFAULT_LOCAL_TASK
- enum MENU_CUSTOM_ITEM
- enum MENU_CUSTOM_MENU

Menu status codes

Status codes for menu callbacks.

- enum **MENU_FOUND**
- enum **MENU_NOT_FOUND**
- enum **MENU_ACCESS_DENIED**

Functions

- menu_get_menu ()
- menu_get_local_tasks ()
- menu_set_location ($location)

- menu_execute_active_handler ()
- menu_get_active_item ()
- menu_set_active_item ($path=NULL)
- menu_get_active_nontask_item ()
- menu_get_active_title ()
- menu_get_active_help ()
- menu_get_active_breadcrumb ()
- menu_in_active_trail ($mid)
- menu_rebuild ()
- theme_menu_tree ($pid=1)
- menu_tree ($pid=1)
- theme_menu_item ($mid, $children= ”, $leaf=TRUE)
- theme_menu_item_link ($item, $link_item)
- menu_item_link ($mid)
- theme_menu_local_tasks ()
- menu_primary_local_tasks ()
- menu_secondary_local_tasks ()
- theme_menu_local_task ($mid, $active, $primary)
- _menu_get_active_trail ()
- _menu_sort ($a, $b)
- _menu_build ()
- _menu_item_is_accessible ($mid)
- _menu_build_visible_tree ($pid=0)
- _menu_append_contextual_items ()
- _menu_find_parents (&$items)
- _menu_build_local_tasks ($pid)

4.10.1 Detailed Description

API for the Drupal menu system.

Definition in file menu.inc.

4.10.2 Function Documentation

4.10.2.1 _menu_append_contextual_items ()

Account for menu items that are only defined at certain paths, so will not be cached.

We don’t support the full range of menu item options for these menu items. We don’t support MENU_-VISIBLE_IF_HAS_CHILDREN, and we require parent items to be declared before their children.

Definition at line 909 of file menu.inc.

References _menu_find_parents(), _menu_item_is_accessible(), _menu_sort(), MENU_NORMAL_-ITEM, and module_invoke_all().

Referenced by menu_get_menu().

```
909                                     {
910   global $_menu;
911
912   // Build a sequential list of all menu items.
913   $menu_item_list = module_invoke_all('menu', FALSE);
914
915   // Menu items not in the DB get temporary negative IDs.
916   $temp_mid = min(array_keys($_menu['items'])) - 1;
917   $new_items = array();
918
919   foreach ($menu_item_list as $item) {
920     if (array_key_exists($item['path'], $_menu['path index'])) {
921       // The menu item already exists, so just add appropriate callback information.
922       $mid = $_menu['path index'][$item['path']];
923
924       $_menu['items'][$mid]['access'] = $item['access'];
925       $_menu['items'][$mid]['callback'] = $item['callback'];
926       $_menu['items'][$mid]['callback arguments'] = $item['callback arguments'];
927     }
928     else {
929       if (!array_key_exists('path', $item)) {
930         $item['path'] = '';
931       }
932       if (!array_key_exists('type', $item)) {
933         $item['type'] = MENU_NORMAL_ITEM;
934       }
935       if (!array_key_exists('weight', $item)) {
936         $item['weight'] = 0;
937       }
938       $_menu['items'][$temp_mid] = $item;
939       $_menu['path index'][$item['path']] = $temp_mid;
940       $new_items[$temp_mid] = $item;
941       $temp_mid--;
942     }
943   }
944
945   // Establish parent-child relationships.
946   _menu_find_parents($new_items);
947
948   // Add new items to the visible tree if necessary.
949   foreach ($new_items as $mid => $item) {
950     $item = $_menu['items'][$mid];
951     if (($item['type'] & MENU_VISIBLE_IN_TREE) && _menu_item_is_accessible($mid)) {
952       $pid = $item['pid'];
953       while ($pid && !array_key_exists($pid, $_menu['visible'])) {
954         $pid = $_menu['items'][$pid]['pid'];
955       }
956       $_menu['visible'][$mid] = array('title' => $item['title'], 'path' => $item['path'], 'pid' => $pi
957       $_menu['visible'][$pid]['children'][] = $mid;
958       usort($_menu['visible'][$pid]['children'], '_menu_sort');
959     }
960   }
961 }
```

4.10.2.2 _menu_build ()

Build the menu by querying both modules and the database.

Definition at line 747 of file menu.inc.

References _menu_build_visible_tree(), _menu_find_parents(), db_fetch_object(), db_query(), drupal_-get_normal_path(), MENU_NORMAL_ITEM, module_exist(), and module_invoke_all().

Referenced by menu_get_menu(), and menu_rebuild().

```
747                       {
748   global $_menu;
749   global $user;
750
751   // Start from a clean slate.
752   $_menu = array();
753
754   $_menu['path index'] = array();
755   // Set up items array, including default "Navigation" menu.
756   $_menu['items'] = array(
757     0 => array('path' => '', 'title' => '', 'type' => MENU_IS_ROOT),
758     1 => array('pid' => 0, 'path' => '', 'title' => t('Navigation'), 'weight' => -50, 'access' => TRUE
759     );
760
761   // Build a sequential list of all menu items.
762   $menu_item_list = module_invoke_all('menu', TRUE);
763
764   // Menu items not in the DB get temporary negative IDs.
765   $temp_mid = -1;
766
767   foreach ($menu_item_list as $item) {
768     if (!array_key_exists('path', $item)) {
769       $item['path'] = '';
770     }
771     if (!array_key_exists('type', $item)) {
772       $item['type'] = MENU_NORMAL_ITEM;
773     }
774     if (!array_key_exists('weight', $item)) {
775       $item['weight'] = 0;
776     }
777     $mid = $temp_mid;
778     if (array_key_exists($item['path'], $_menu['path index'])) {
779       // Newer menu items overwrite older ones.
780       unset($_menu['items'][$_menu['path index'][$item['path']]]);
781     }
782     $_menu['items'][$mid] = $item;
783     $_menu['path index'][$item['path']] = $mid;
784
785     $temp_mid--;
786   }
787
788   // Now fetch items from the DB, reassigning menu IDs as needed.
789   if (module_exist('menu')) {
790     $result = db_query('SELECT * FROM {menu} ORDER BY mid ASC');
791     while ($item = db_fetch_object($result)) {
792       // Handle URL aliases if entered in menu administration.
793       $item->path = drupal_get_normal_path($item->path);
794       if (array_key_exists($item->path, $_menu['path index'])) {
795         // The path is already declared.
796         $old_mid = $_menu['path index'][$item->path];
797         if ($old_mid < 0) {
798           // It had a temporary ID, so use a permanent one.
799           $_menu['items'][$item->mid] = $_menu['items'][$old_mid];
800           unset($_menu['items'][$old_mid]);
801           $_menu['path index'][$item->path] = $item->mid;
802         }
803         else {
804           // It has a permanent ID. Only replace with non-custom menu items.
805           if ($item->type & MENU_CREATED_BY_ADMIN) {
806             $_menu['items'][$item->mid] = array('path' => $item->path, 'access' => TRUE, 'callback' =>
807           }
808           else {
809             // Leave the old item around as a shortcut to this one.
810             $_menu['items'][$item->mid] = $_menu['items'][$old_mid];
811             $_menu['path index'][$item->path] = $item->mid;
```

```
812           }
813         }
814       }
815       else {
816         // The path was not declared, so this is a custom item or an orphaned one.
817         if ($item->type & MENU_CREATED_BY_ADMIN) {
818           $_menu['items'][$item->mid] = array('path' => $item->path, 'access' => TRUE, 'callback' => '
819           if (!empty($item->path)) {
820             $_menu['path index'][$item->path] = $item->mid;
821           }
822         }
823       }
824
825       // If the administrator has changed the item, reflect the change.
826       if ($item->type & MENU_MODIFIED_BY_ADMIN) {
827         $_menu['items'][$item->mid]['title'] = $item->title;
828         $_menu['items'][$item->mid]['description'] = $item->description;
829         $_menu['items'][$item->mid]['pid'] = $item->pid;
830         $_menu['items'][$item->mid]['weight'] = $item->weight;
831         $_menu['items'][$item->mid]['type'] = $item->type;
832       }
833     }
834   }
835
836   // Associate parent and child menu items.
837   _menu_find_parents($_menu['items']);
838
839   // Prepare to display trees to the user as required.
840   _menu_build_visible_tree();
841 }
```

4.10.2.3 _menu_build_local_tasks ($ *pid*)

Find all the items in the current local task tree.

Since this is only for display, we only need title, path, and children for each item.

At the close of this function, $_menu['local tasks'] is populated with the menu items in the local task tree.

Returns:

TRUE if the local task tree is forked. It does not need to be displayed otherwise.

Definition at line 1012 of file menu.inc.

References _menu_item_is_accessible(), and _menu_sort().

Referenced by menu_get_local_tasks().

```
1012                                        {
1013   global $_menu;
1014
1015   $forked = FALSE;
1016
1017   if (isset($_menu['items'][$pid])) {
1018     $parent = $_menu['items'][$pid];
1019
1020     $children = array();
1021     if (array_key_exists('children', $parent)) {
1022       foreach ($parent['children'] as $mid) {
1023         if (($_menu['items'][$mid]['type'] & MENU_IS_LOCAL_TASK) && _menu_item_is_accessible($mid)) {
1024           $children[] = $mid;
```

```
1025           // Beware short-circuiting || operator!
1026           $forked = _menu_build_local_tasks($mid) || $forked;
1027         }
1028       }
1029     }
1030     usort($children, '_menu_sort');
1031     $forked = $forked || count($children) > 1;
1032
1033     $_menu['local tasks'][$pid] = array('title' => $parent['title'], 'path' => $parent['path'], 'chil
1034     foreach ($children as $mid) {
1035       $_menu['local tasks'][$mid]['pid'] = $pid;
1036     }
1037   }
1038
1039   return $forked;
1040 }
```

4.10.2.4 _menu_build_visible_tree ($ *pid* = 0)

Find all visible items in the menu tree, for ease in displaying to user.

Since this is only for display, we only need title, path, and children for each item.

Definition at line 869 of file menu.inc.

References _menu_item_is_accessible(), and _menu_sort().

Referenced by _menu_build().

```
869                                             {
870   global $_menu;
871
872   if (isset($_menu['items'][$pid])) {
873     $parent = $_menu['items'][$pid];
874
875     $children = array();
876     if (array_key_exists('children', $parent)) {
877       usort($parent['children'], '_menu_sort');
878       foreach ($parent['children'] as $mid) {
879         $children = array_merge($children, _menu_build_visible_tree($mid));
880       }
881     }
882     $visible = ($parent['type'] & MENU_VISIBLE_IN_TREE) ||
883       ($parent['type'] & MENU_VISIBLE_IF_HAS_CHILDREN && count($children) > 0);
884     $allowed = _menu_item_is_accessible($pid);
885
886     if (($parent['type'] & MENU_IS_ROOT) || ($visible && $allowed)) {
887       $_menu['visible'][$pid] = array('title' => $parent['title'], 'path' => $parent['path'], 'childre
888       foreach ($children as $mid) {
889         $_menu['visible'][$mid]['pid'] = $pid;
890       }
891       return array($pid);
892     }
893     else {
894       return $children;
895     }
896   }
897
898   return array();
899 }
```

4.10.2.5 _menu_find_parents (&$ *items*)

Establish parent-child relationships.

Definition at line 966 of file menu.inc.

Referenced by _menu_append_contextual_items(), and _menu_build().

```
966                                     {
967   global $_menu;
968
969   foreach ($items as $mid => $item) {
970     if (!isset($item['pid'])) {
971       // Parent's location has not been customized, so figure it out using the path.
972       $parent = $item['path'];
973       do {
974         $parent = substr($parent, 0, strrpos($parent, '/'));
975       }
976       while ($parent && !array_key_exists($parent, $_menu['path index']));
977
978       $pid = $parent ? $_menu['path index'][$parent] : 1;
979       $_menu['items'][$mid]['pid'] = $pid;
980     }
981     else {
982       $pid = $item['pid'];
983     }
984
985     // Don't make root a child of itself.
986     if ($mid) {
987       if (isset ($_menu['items'][$pid])) {
988         $_menu['items'][$pid]['children'][] = $mid;
989       }
990       else {
991         // If parent is missing, it is a menu item that used to be defined
992         // but is no longer. Default to a root-level "Navigation" menu item.
993         $_menu['items'][1]['children'][] = $mid;
994       }
995     }
996   }
997 }
```

4.10.2.6 _menu_get_active_trail ()

Returns an array with the menu items that lead to the current menu item.

Definition at line 713 of file menu.inc.

References menu_get_active_item(), and menu_get_menu().

Referenced by menu_get_active_breadcrumb(), and menu_in_active_trail().

```
713                                   {
714   static $trail;
715
716   if (!isset($trail)) {
717     $menu = menu_get_menu();
718
719     $trail = array();
720
721     $mid = menu_get_active_item();
722
723     // Follow the parents up the chain to get the trail.
```

```
724     while ($mid && $menu['items'][$mid]) {
725       array_unshift($trail, $mid);
726       $mid = $menu['items'][$mid]['pid'];
727     }
728   }
729   return $trail;
730 }
```

4.10.2.7 _menu_item_is_accessible ($ *mid*)

Determine whether the given menu item is accessible to the current user.

Use this instead of just checking the "access" property of a menu item to properly handle items with fall-through semantics.

Definition at line 849 of file menu.inc.

References menu_get_menu().

Referenced by _menu_append_contextual_items(), _menu_build_local_tasks(), _menu_build_visible_-tree(), menu_execute_active_handler(), and menu_get_active_help().

```
849                                       {
850   $menu = menu_get_menu();
851
852   // Follow the path up to find the first "access" attribute.
853   $path = $menu['items'][$mid]['path'];
854   while ($path && (!array_key_exists($path, $menu['path index']) || !array_key_exists('access', $menu[
855     $path = substr($path, 0, strrpos($path, '/'));
856   }
857   if (empty($path)) {
858     return FALSE;
859   }
860   return $menu['items'][$menu['path index'][$path]]['access'];
861 }
```

4.10.2.8 _menu_sort ($ *a*, $ *b*)

Comparator routine for use in sorting menu items.

Definition at line 735 of file menu.inc.

References menu_get_menu().

Referenced by _menu_append_contextual_items(), _menu_build_local_tasks(), and _menu_build_-visible_tree().

```
735                               {
736   $menu = menu_get_menu();
737
738   $a = &$menu['items'][$a];
739   $b = &$menu['items'][$b];
740
741   return $a['weight'] < $b['weight'] ? -1 : ($a['weight'] > $b['weight'] ? 1 : ($a['title'] < $b['titl
742 }
```

4.11 module.inc File Reference

Functions

- module_init ()
- module_iterate ($function, $argument= ”)
- module_list ($refresh=FALSE, $bootstrap=TRUE)
- module_load_all ()
- module_exist ($module)
- module_hook ($module, $hook)
- module_implements ($hook)
- module_invoke ()
- module_invoke_all ()

4.11.1 Detailed Description

API for loading and interacting with Drupal modules.

Definition in file module.inc.

4.11.2 Function Documentation

4.11.2.1 module_exist ($ *module*)

Determine whether a given module exists.

Parameters:

$module The name of the module (without the .module extension).

Returns:

TRUE if the module is both installed and enabled.

Definition at line 99 of file module.inc.

References module_list().

Referenced by _menu_build(), menu_rebuild(), theme_aggregator_block_item(), theme_forum_display(), theme_get_settings(), and theme_node().

```
99                                {
100   $list = module_list();
101   return array_key_exists($module, $list);
102 }
```

4.11.2.2 module_init ()

Initialize all modules.

Definition at line 13 of file module.inc.

References module_invoke_all(), and module_load_all().

```
13                        {
14   module_load_all();
15   module_invoke_all('init');
16 }
```

4.11.2.3 module_iterate ($ *function*, $ *argument* = ”)

Call a function repeatedly with each module in turn as an argument.

Definition at line 21 of file module.inc.

References module_list().

```
21                                                    {
22   foreach (module_list() as $name) {
23     $function($name, $argument);
24   }
25 }
```

4.11.2.4 module_list ($ *refresh* = FALSE, $ *bootstrap* = TRUE)

Collect a list of all loaded modules. During the bootstrap, return only vital modules. See bootstrap.inc

Parameters:

$refresh Whether to force the module list to be regenerated (such as after the administrator has changed the system settings).

$bootstrap Whether to return the reduced set of modules loaded in "bootstrap mode" for cached pages. See bootstrap.inc.

Returns:

An associative array whose keys and values are the names of all loaded modules.

Definition at line 41 of file module.inc.

References db_fetch_object(), db_query(), drupal_get_filename(), variable_get(), and watchdog().

Referenced by bootstrap_invoke_all(), file_download(), form_textarea(), menu_get_active_help(), module_exist(), module_implements(), module_iterate(), and module_load_all().

```
41                                                        {
42   static $list;
43
44   if ($refresh) {
45     $list = array();
```

```
46   }
47
48   if (!$list) {
49     $list = array('filter' => 'filter', 'system' => 'system', 'user' => 'user', 'watchdog' => 'watchdog
50     if ($bootstrap) {
51       $result = db_query("SELECT name, filename, throttle, bootstrap FROM {system} WHERE type = 'module
52     }
53     else {
54       $result = db_query("SELECT name, filename, throttle, bootstrap FROM {system} WHERE type = 'module
55     }
56     while ($module = db_fetch_object($result)) {
57       if (file_exists($module->filename)) {
58         // Determine the current throttle status and see if the module should be
59         // loaded based on server load. We have to directly access the throttle
60         // variables, since throttle.module may not be loaded yet.
61         $throttle = ($module->throttle && variable_get('throttle_level', 0) > 0);
62         if (!$throttle) {
63           drupal_get_filename('module', $module->name, $module->filename);
64           $list[$module->name] = $module->name;
65         }
66       }
67     }
68     asort($list);
69   }
70   return $list;
71 }
```

4.11.2.5 module_load_all ()

Load all the modules that have been enabled in the system table.

Returns:

TRUE if all modules were loaded successfully.

Definition at line 79 of file module.inc.

References drupal_load(), and module_list().

Referenced by module_init().

```
79                          {
80   $list = module_list(TRUE, FALSE);
81   $status = TRUE;
82
83   foreach ($list as $module) {
84     $status = (drupal_load('module', $module) && $status);
85   }
86
87   return $status;
88 }
```

4.12 pager.inc File Reference

Pager pieces

Use these pieces to construct your own custom pagers in your theme. Note that you should NOT modify this file to customize your pager.

- theme_pager_first ($text, $limit, $element=0, $attributes=array())
- theme_pager_previous ($text, $limit, $element=0, $interval=1, $attributes=array())
- theme_pager_next ($text, $limit, $element=0, $interval=1, $attributes=array())
- theme_pager_last ($text, $limit, $element=0, $attributes=array())
- theme_pager_detail ($limit, $element=0, $format= '%d through%d of%d.')
- theme_pager_list ($limit, $element=0, $quantity=5, $text= '', $attributes=array())

Functions

- pager_query ($query, $limit=10, $element=0, $count_query=NULL)
- theme_pager ($tags=array(), $limit=10, $element=0, $attributes=array())
- pager_link ($from_new, $element, $attributes=array())
- **pager_load_array** ($value, $element, $old_array)

4.12.1 Detailed Description

Functions to aid in presenting database results as a set of pages.

Definition in file pager.inc.

4.12.2 Function Documentation

4.12.2.1 pager_link ($ *from_new*, $ *element*, $ *attributes* = array())

Format a link to a specific query result page.

Parameters:

$from_new The first result to display on the linked page.

$element An optional integer to distinguish between multiple pagers on one page.

$attributes An associative array of query string parameters to append to the pager link.

Returns:

An HTML string that generates the link.

Definition at line 383 of file pager.inc.

References check_url(), and url().

```
383                                                                 {
384   $q = $_GET['q'];
385   $from = array_key_exists('from', $_GET) ? $_GET['from'] : '';
386
387   foreach ($attributes as $key => $value) {
388     $query[] = $key .'='. $value;
389   }
390
391   $from_new = pager_load_array($from_new[$element], $element, explode(',', $from));
392   if (count($attributes)) {
393     $url = url($q, 'from='. implode($from_new, ',') .'&'. implode('&', $query));
394   }
395   else {
396     $url = url($q, 'from='. implode($from_new, ','));
397   }
398
399   return check_url($url);
400 }
```

4.13 session.inc File Reference

Functions

- **sess_open** ($save_path, $session_name)
- **sess_close** ()
- **sess_read** ($key)
- **sess_write** ($key, $value)
- **sess_destroy** ($key)
- **sess_gc** ($lifetime)

4.13.1 Detailed Description

User session handling functions.

Definition in file session.inc.

4.14 settings.php File Reference

Variables

- $db_url
- $base_url

4.14.1 Detailed Description

Drupal site-specific configuration file.

The configuration file to be loaded is based upon the rules below.

The configuration directory will be discovered by stripping the website's hostname from left to right and pathname from right to left. The first configuration file found will be used and any others will be ignored. If no other configuration file is found then the default configuration file at 'sites/default' will be used.

For example, for a fictitious site installed at `http://www.drupal.org/mysite/test/`, the 'settings.php' is searched in the following directories:

1. sites/www.drupal.org.mysite.test 2. sites/drupal.org.mysite.test 3. sites/org.mysite.test

4. sites/www.drupal.org.mysite 5. sites/drupal.org.mysite 6. sites/org.mysite

7. sites/www.drupal.org 8. sites/drupal.org 9. sites/org

10. sites/default

Definition in file settings.php.

4.14.2 Variable Documentation

4.14.2.1 $base_url

Initial value:

```
 'http:

ini_set(arg_separator.output,     &amp;)
```

Base URL:

The URL of your website's main page. It is not allowed to have a trailing slash; Drupal will add it for you.

Definition at line 90 of file settings.php.

Referenced by drupal_get_html_head(), page_get_cache(), page_set_cache(), search_index(), and url().

4.14.2.2 $db_url

Initial value:

```
 'mysql:
$db_prefix = ''
```

Database settings:

Note that the $db_url variable gets parsed using PHP's built-in URL parser (i.e. using the "parse_url()" function) so make sure not to confuse the parser. If your username, password or database name contain characters used to delineate $db_url parts, you can escape them via URI hex encodings:

: = 3a / = 2f @ = 40 + = 2b (= 28) = 29 ? = 3f = = 3d & = 26

To specify multiple connections to be used in your site (i.e. for complex custom modules) you can also specify an associative array of $db_url variables with the 'default' element used until otherwise requested.

You can optionally set prefixes for some or all database table names by using the $db_prefix setting. If a prefix is specified, the table name will be prepended with its value. Be sure to use valid database characters only, usually alphanumeric and underscore. If no prefixes are desired, leave it as an empty string ''.

To have all database names prefixed, set $db_prefix as a string:

$db_prefix = 'main_';

To provide prefixes for specific tables, set $db_prefix as an array. The array's keys are the table names and the values are the prefixes. The 'default' element holds the prefix for any tables not specified elsewhere in the array. Example:

$db_prefix = array('default' => 'main_', 'users' => 'shared_', 'sessions' => 'shared_', 'role' => 'shared_', 'authmap' => 'shared_', 'sequences' => 'shared_',);

Database URL format: $db_url = 'mysql://username:password/database'; $db_url = 'pgsql://username:password/database';

Definition at line 81 of file settings.php.

Referenced by db_set_active().

4.15 tablesort.inc File Reference

Functions

- tablesort_init ($header)
- tablesort_pager ()
- tablesort_sql ($header, $before= ”)
- tablesort_header ($cell, $header, $ts)
- tablesort_cell ($cell, $header, $ts, $i)
- tablesort_get_querystring ()
- tablesort_get_order ($headers)
- tablesort_get_sort ($headers)

4.15.1 Detailed Description

Functions to aid in the creation of sortable tables.

All tables created with a call to theme('table') have the option of having column headers that the user can click on to sort the table by that column.

Definition in file tablesort.inc.

4.15.2 Function Documentation

4.15.2.1 tablesort_cell ($ *cell*, $ *header*, $ *ts*, $ *i*)

Format a table cell.

Adds a class attribute to all cells in the currently active column.

Parameters:

$cell The cell to format.

$header An array of column headers in the format described in theme_table().

$ts The current table sort context as returned from tablesort_init().

$i The index of the cell's table column.

Returns:

A properly formatted cell, ready for _theme_table_cell().

Definition at line 113 of file tablesort.inc.

Referenced by theme_table().

```
113                                                 {
114   if (isset($header[$i]) && $header[$i]['data'] == $ts['name'] && $header[$i]['field']) {
115     if (is_array($cell)) {
```

```
116       if (isset($cell['class'])) {
117         $cell['class'] .= ' active';
118       }
119       else {
120         $cell['class'] = 'active';
121       }
122     }
123     else {
124       $cell = array('data' => $cell, 'class' => 'active');
125     }
126   }
127   return $cell;
128 }
```

4.15.2.2 tablesort_get_order ($ *headers*)

Determine the current sort criterion.

Parameters:

$headers An array of column headers in the format described in theme_table().

Returns:

An associative array describing the criterion, containing the keys:

- "name": The localized title of the table column.
- "sql": The name of the database field to sort on.

Definition at line 158 of file tablesort.inc.

Referenced by tablesort_init().

```
158                                       {
159   $order = isset($_GET['order']) ? $_GET['order'] : '';
160   foreach ($headers as $header) {
161     if (isset($header['data']) && $order == $header['data']) {
162       return array('name' => $header['data'], 'sql' => $header['field']);
163     }
164
165     if (isset($header['sort']) && ($header['sort'] == 'asc' || $header['sort'] == 'desc')) {
166       $default = array('name' => $header['data'], 'sql' => $header['field']);
167     }
168   }
169
170   if (isset($default)) {
171     return $default;
172   }
173   else {
174     // The first column specified is initial 'order by' field unless otherwise specified
175     if (is_array($headers[0])) {
176       return array('name' => $headers[0]['data'], 'sql' => $headers[0]['field']);
177     }
178     else {
179       return array('name' => $headers[0]);
180     }
181   }
182 }
```

4.15.2.3 tablesort_get_querystring ()

Compose a query string to append to table sorting requests.

Returns:

A query string that consists of all components of the current page request except for those pertaining to table sorting.

Definition at line 137 of file tablesort.inc.

Referenced by tablesort_init().

```
137                                     {
138   $cgi = $_SERVER['REQUEST_METHOD'] == 'GET' ? $_GET : $_POST;
139   $query_string = '';
140   foreach ($cgi as $key => $val) {
141     if ($key != 'order' && $key != 'sort' && $key != 'q') {
142       $query_string .= '&'. $key .'='. $val;
143     }
144   }
145   return $query_string;
146 }
```

4.15.2.4 tablesort_get_sort ($ *headers*)

Determine the current sort direction.

Parameters:

$headers An array of column headers in the format described in theme_table().

Returns:

The current sort direction ("asc" or "desc").

Definition at line 192 of file tablesort.inc.

Referenced by tablesort_init().

```
192                                       {
193   if (isset($_GET['sort'])) {
194     return ($_GET['sort'] == 'desc') ? 'desc' : 'asc';
195   }
196   // User has not specified a sort. Use default if specified; otherwise use "asc".
197   else {
198     foreach ($headers as $header) {
199       if (is_array($header) && array_key_exists('sort', $header)) {
200         return $header['sort'];
201       }
202     }
203   }
204   return 'asc';
205 }
```

4.15.2.5 tablesort_header ($ *cell*, $ *header*, $ *ts*)

Format a column header.

If the cell in question is the column header for the current sort criterion, it gets special formatting. All possible sort criteria become links.

Parameters:

$cell The cell to format.

$header An array of column headers in the format described in theme_table().

$ts The current table sort context as returned from tablesort_init().

Returns:

A properly formatted cell, ready for _theme_table_cell().

Definition at line 75 of file tablesort.inc.

References l(), and theme().

Referenced by theme_table().

```
75                                              {
76   // Special formatting for the currently sorted column header.
77   if (is_array($cell) && isset($cell['field'])) {
78     $title = t('sort by %s', array('%s' => $cell['data']));
79     if ($cell['data'] == $ts['name']) {
80       $ts['sort'] = (($ts['sort'] == 'asc') ? 'desc' : 'asc');
81       $cell['class'] = 'active';
82       $title = ($ts['sort'] == 'asc' ? t('sort ascending') : t('sort descending'));
83       $image = ' '. theme('image', 'misc/arrow-'. $ts['sort'] .'.png', t('sort icon'), $title);
84     }
85     else {
86       // If the user clicks a different header, we want to sort ascending initially.
87       $ts['sort'] = 'asc';
88       $image = '';
89     }
90     $cell['data'] = l($cell['data'] . $image, $_GET['q'], array('title' => $title), 'sort='. $ts['sort'
91
92     unset($cell['field'], $cell['sort']);
93   }
94   return $cell;
95 }
```

4.15.2.6 tablesort_init ($ *header*)

Initialize the table sort context.

Definition at line 15 of file tablesort.inc.

References tablesort_get_order(), tablesort_get_querystring(), and tablesort_get_sort().

Referenced by tablesort_sql(), and theme_table().

```
15                                  {
16   $ts = tablesort_get_order($header);
17   $ts['sort'] = tablesort_get_sort($header);
18   $ts['query_string'] = tablesort_get_querystring();
19   return $ts;
20 }
```

4.15.2.7 tablesort_pager ()

Fetch pager link arguments.

When producing a sortable table that presents paged data, pass the output of this function into theme('pager') to preserve the current sort order.

Definition at line 28 of file tablesort.inc.

Referenced by theme_forum_topic_list().

```
28                              {
29   $cgi = $_SERVER['REQUEST_METHOD'] == 'GET' ? $_GET : $_POST;
30   unset($cgi['q'], $cgi['from']);
31   return $cgi;
32 }
```

4.16 theme.inc File Reference

Content markers

Markers used by theme_mark() and node_mark() to designate content.

See also:
theme_mark(), node_mark()

- enum **MARK_READ**
- enum **MARK_NEW**
- enum **MARK_UPDATED**

Functions

- theme_help ($section)
- init_theme ()
- list_themes ($refresh=FALSE)
- list_theme_engines ($refresh=FALSE)
- theme ()
- path_to_theme ()
- theme_get_settings ($key=NULL)
- theme_get_setting ($setting_name, $refresh=FALSE)
- theme_add_style ($style= ”)
- theme_get_styles ()
- theme_placeholder ($text)
- theme_page ($content)
- theme_status_messages ()
- theme_links ($links, $delimiter= ’| ’)
- theme_image ($path, $alt= ”, $title= ”, $attr= ”, $getsize=true)
- theme_breadcrumb ($breadcrumb)
- theme_node ($node, $teaser=FALSE, $page=FALSE)
- theme_form_element ($title, $value, $description=NULL, $id=NULL, $required=FALSE, $error=FALSE)
- theme_submenu ($links)
- theme_table ($header, $rows, $attributes=NULL)
- theme_box ($title, $content, $region= ’main’)
- theme_block ($block)
- theme_mark ($type=MARK_NEW)
- theme_stylesheet_import ($stylesheet, $media= ’all’)
- theme_item_list ($items=array(), $title=NULL)
- theme_error ($message)
- **theme_more_help_link** ($url)
- theme_xml_icon ($url)
- theme_closure ($main=0)
- theme_onload_attribute ($theme_onloads=array())
- theme_blocks ($region)

- theme_confirm ($question, $path, $description=NULL, $yes=NULL, $no=NULL, $extra=NULL, $name= 'confirm')
- _theme_table_cell ($cell, $header=0)

4.16.1 Detailed Description

The theme system, which controls the output of Drupal.

The theme system allows for nearly all output of the Drupal system to be customized by user themes.

See also:
Theme system
themeable

Definition in file theme.inc.

4.16.2 Function Documentation

4.16.2.1 _theme_table_cell ($ *cell*, $ *header* = 0)

End of "defgroup themeable".

Definition at line 927 of file theme.inc.

Referenced by theme_table().

```
927                                                 {
928   $attributes = '';
929
930   if (is_array($cell)) {
931     $data = $cell['data'];
932     foreach ($cell as $key => $value) {
933       if ($key != 'data')  {
934         $attributes .= " $key=\"$value\"";
935       }
936     }
937   }
938   else {
939     $data = $cell;
940   }
941
942   if ($header) {
943     $output = "<th$attributes>$data</th>";
944   }
945   else {
946     $output = "<td$attributes>$data</td>";
947   }
948
949   return $output;
950 }
```

4.16.2.2 init_theme ()

Initialize the theme system by loading the theme.

Returns:
The name of the currently selected theme.

Definition at line 48 of file theme.inc.

References list_themes(), theme_add_style(), and variable_get().

Referenced by theme().

```
48                      {
49   global $user, $custom_theme, $theme_engine, $theme_key;
50
51   $themes = list_themes();
52
53   // Only select the user selected theme if it is available in the
54   // list of enabled themes.
55   $theme = $user->theme && $themes[$user->theme]->status ? $user->theme : variable_get('theme_default',
56
57   // Allow modules to override the present theme... only select custom theme
58   // if it is available in the list of installed themes.
59   $theme = $custom_theme && $themes[$custom_theme] ? $custom_theme : $theme;
60
61   // Store the identifier for retrieving theme settings with.
62   $theme_key = $theme;
63
64   // If we're using a style, load its appropriate theme,
65   // which is stored in the style's description field.
66   // Also load the stylesheet using drupal_set_html_head().
67   // Otherwise, load the theme.
68   if (strpos($themes[$theme]->filename, '.css')) {
69     // File is a style; loads its CSS.
70     // Set theme to its template/theme
71     theme_add_style($themes[$theme]->filename);
72     $theme = basename(dirname($themes[$theme]->description));
73   }
74   else {
75     // File is a template/theme
76     // Load its CSS, if it exists
77     if (file_exists($stylesheet = dirname($themes[$theme]->filename) .'/style.css')) {
78       theme_add_style($stylesheet);
79     }
80   }
81
82   if (strpos($themes[$theme]->filename, '.theme')) {
83     // file is a theme; include it
84     include_once($themes[$theme]->filename);
85   }
86   elseif (strpos($themes[$theme]->description, '.engine')) {
87     // file is a template; include its engine
88     include_once($themes[$theme]->description);
89     $theme_engine = basename($themes[$theme]->description, '.engine');
90     if (function_exists($theme_engine .'_init')) {
91       call_user_func($theme_engine .'_init', $themes[$theme]);
92     }
93   }
94
95   return $theme;
96 }
```

4.16.2.3 list_theme_engines ($ *refresh* = FALSE)

Provides a list of currently available theme engines

Parameters:

$refresh Whether to reload the list of themes from the database.

Returns:

An array of the currently available theme engines.

Definition at line 134 of file theme.inc.

References db_fetch_object(), and db_query().

```
134                                                {
135   static $list;
136
137   if ($refresh) {
138     unset($list);
139   }
140
141   if (!$list) {
142     $list = array();
143     $result = db_query("SELECT * FROM {system} WHERE type = 'theme_engine' AND status = '1' ORDER BY r
144     while ($engine = db_fetch_object($result)) {
145       if (file_exists($engine->filename)) {
146         $list[$engine->name] = $engine;
147       }
148     }
149   }
150
151   return $list;
152 }
```

4.16.2.4 list_themes ($ *refresh* = FALSE)

Provides a list of currently available themes.

Parameters:

$refresh Whether to reload the list of themes from the database.

Returns:

An array of the currently available themes.

Definition at line 106 of file theme.inc.

References db_fetch_object(), and db_query().

Referenced by init_theme(), path_to_theme(), and theme_get_setting().

```
106                                          {
107   static $list;
108
109   if ($refresh) {
110     unset($list);
111   }
```

```
112
113   if (!$list) {
114     $list = array();
115     $result = db_query("SELECT * FROM {system} WHERE type = 'theme' ORDER BY name");
116     while ($theme = db_fetch_object($result)) {
117       if (file_exists($theme->filename)) {
118         $list[$theme->name] = $theme;
119       }
120     }
121   }
122
123   return $list;
124 }
```

4.16.2.5 path_to_theme ()

Return the path to the currently selected theme.

Definition at line 201 of file theme.inc.

References list_themes().

```
201                          {
202   global $theme;
203
204   $themes = list_themes();
205
206   return dirname($themes[$theme]->filename);
207 }
```

4.16.2.6 theme ()

Generate the themed representation of a Drupal object.

All requests for themed functions must go through this function. It examines the request and routes it to the appropriate theme function. If the current theme does not implement the requested function, then the current theme engine is checked. If neither the engine nor theme implement the requested function, then the base theme function is called.

For example, to retrieve the HTML that is output by theme_page($output), a module should call theme('page', $output).

Parameters:

$function The name of the theme function to call.

... Additional arguments to pass along to the theme function.

Returns:

An HTML string that generates the themed output.

Definition at line 173 of file theme.inc.

References init_theme().

Referenced by _locale_add_language(), _locale_admin_manage_screen(), _locale_export_po(), _locale_import_parse_plural_forms(), _locale_import_read_po(), _locale_string_seek(), drupal_access_denied(), drupal_get_html_head(), drupal_not_found(), file_check_directory(), form_checkbox(), form_checkboxes(), form_file(), form_item(), form_password(), form_radio(), form_radios(), form_select(), form_textarea(), form_textfield(), menu_get_active_help(), menu_item_link(), menu_primary_local_tasks(), menu_secondary_local_tasks(), menu_tree(), tablesort_header(), theme_aggregator_block_item(), theme_aggregator_feed(), theme_blocks(), theme_forum_display(), theme_forum_list(), theme_forum_topic_list(), theme_get_styles(), theme_node(), theme_page(), theme_pager(), theme_pager_last(), theme_pager_list(), theme_pager_previous(), theme_xml_icon(), and valid_input_data().

```
173                 {
174   global $theme, $theme_engine;
175
176   if (!$theme) {
177     // Initialize the enabled theme.
178     $theme = init_theme();
179   }
180
181   $args = func_get_args();
182   $function = array_shift($args);
183
184   if (($theme != '') && function_exists($theme .'_'. $function)) {
185     // call theme function
186     return call_user_func_array($theme .'_'. $function, $args);
187   }
188   elseif (($theme != '') && isset($theme_engine) && function_exists($theme_engine .'_'. $function)) {
189     // call engine function
190     return call_user_func_array($theme_engine .'_'. $function, $args);
191   }
192   elseif (function_exists('theme_'. $function)){
193     // call Drupal function
194     return call_user_func_array('theme_'. $function, $args);
195   }
196 }
```

4.16.2.7 theme_add_style ($ *style* = ")

Add a theme stylesheet to be included later. This is handled separately from drupal_set_html_head() to enforce the correct CSS cascading order.

Definition at line 322 of file theme.inc.

Referenced by init_theme(), and theme_get_styles().

```
322                                     {
323   static $styles = array();
324   if ($style) {
325     $styles[] = $style;
326   }
327   return $styles;
328 }
```

4.16.2.8 theme_get_setting ($ *setting_name*, $ *refresh* = FALSE)

Retrieve a setting for the current theme. This function is designed for use from within themes & engines to determine theme settings made in the admin interface.

Caches values for speed (use $refresh = TRUE to refresh cache)

Parameters:

$setting_name The name of the setting to be retrieved.

$refresh Whether to reload the cache of settings.

Returns:

The value of the requested setting, NULL if the setting does not exist.

Definition at line 279 of file theme.inc.

References list_themes(), theme_get_settings(), and variable_get().

```
279                                                         {
280   global $theme_key;
281   static $settings;
282
283   if (empty($settings) || $refresh) {
284     $settings = theme_get_settings($theme_key);
285
286     $themes = list_themes();
287     $theme_object = $themes[$theme_key];
288
289     if ($settings['mission'] == '') {
290       $settings['mission'] = variable_get('site_mission', '');
291     }
292
293     if (!$settings['toggle_mission']) {
294       $settings['mission'] = '';
295     }
296
297     if ($settings['toggle_logo']) {
298       if ($settings['default_logo']) {
299         $settings['logo'] = dirname($theme_object->filename) .'/logo.png';
300       }
301       elseif ($settings['logo_path']) {
302         $settings['logo'] = $settings['logo_path'];
303       }
304     }
305
306     if (!$settings['toggle_primary_links']) {
307       $settings['primary_links'] = '';
308     }
309
310     if (!$settings['toggle_secondary_links']) {
311       $settings['secondary_links'] = '';
312     }
313   }
314
315   return isset($settings[$setting_name]) ? $settings[$setting_name] : NULL;
316 }
```

4.16.2.9 theme_get_settings ($ *key* = NULL)

Retrieve an associative array containing the settings for a theme.

The final settings are arrived at by merging the default settings, the site-wide settings, and the settings defined for the specific theme. If no $key was specified, only the site-wide theme defaults are retrieved.

The default values for each of settings are also defined in this function. To add new settings, add their default values here, and then add form elements to system_theme_settings() in system.module.

Parameters:

$key The template/style value for a given theme.

Returns:

An associative array containing theme settings.

Definition at line 226 of file theme.inc.

References l(), module_exist(), and variable_get().

Referenced by theme_get_setting().

```
226                                          {
227   $defaults = array(
228     'primary_links'                =>  l(t('edit primary links'), 'admin/themes/settings'),
229     'secondary_links'              =>  l(t('edit secondary links'), 'admin/themes/settings'),
230     'mission'                      =>  '',
231     'default_logo'                 =>  1,
232     'logo_path'                    =>  '',
233     'toggle_logo'                  =>  1,
234     'toggle_name'                  =>  1,
235     'toggle_search'                =>  1,
236     'toggle_slogan'                =>  0,
237     'toggle_mission'               =>  1,
238     'toggle_primary_links'         =>  1,
239     'toggle_secondary_links'       =>  1,
240     'toggle_node_user_picture'     =>  0,
241     'toggle_comment_user_picture'  =>  0,
242   );
243
244   if (module_exist('node')) {
245     foreach (node_list() as $type) {
246       $defaults['toggle_node_info_' . $type] = 1;
247     }
248   }
249   $settings = array_merge($defaults, variable_get('theme_settings', array()));
250
251   if ($key) {
252     $settings = array_merge($settings, variable_get(str_replace('/', '_', 'theme_'. $key .'_settings')
253   }
254
255   // Only offer search box if search.module is enabled.
256   if (!module_exist('search')) {
257     $settings['toggle_search'] = 0;
258   }
259
260   return $settings;
261 }
```

4.16.2.10 theme_get_styles ()

Return the HTML for a theme's stylesheets.

Definition at line 333 of file theme.inc.

References theme(), and theme_add_style().

Referenced by theme_page().

```
333                                {
334   $output = '';
```

```
335   foreach (theme_add_style() as $style) {
336     $output .= theme('stylesheet_import', $style);
337   }
338   return $output;
339 }
```

4.16.2.11 theme_help ($ *section*)

Hook Help - returns theme specific help and information.

Parameters:

section defines the *section* of the help to be returned.

Returns:

a string containing the help output.

Definition at line 35 of file theme.inc.

```
35                                {
36   switch ($section) {
37     case 'admin/themes#description':
38       return t('The base theme');
39   }
40 }
```

4.17 update.php File Reference

Functions

- **update_data** ($start)
- **update_page_header** ($title)
- **update_page_footer** ()
- **update_page** ()
- **update_info** ()

Variables

- **$access_check** = TRUE

4.17.1 Detailed Description

Administrative page for handling updates from one Drupal version to another.

Point your browser to "http://www.site.com/update.php" and follow the instructions.

If you are not logged in as administrator, you will need to modify the access check statement below. Change the TRUE into a FALSE to disable the access check. After finishing the upgrade, be sure to open this file and change the FALSE back into a TRUE!

Definition in file update.php.

4.18 updates.inc File Reference

Functions

- **update_32** ()
- **update_33** ()
- **update_34** ()
- **update_35** ()
- **update_36** ()
- **update_37** ()
- **update_38** ()
- **update_39** ()
- **update_40** ()
- **update_41** ()
- **update_42** ()
- **update_43** ()
- **update_44** ()
- **update_45** ()
- **update_46** ()
- **update_47** ()
- **update_48** ()
- **update_49** ()
- **update_50** ()
- **update_51** ()
- **update_52** ()
- **update_53** ()
- **update_54** ()
- **update_55** ()
- **update_56** ()
- **update_57** ()
- **update_58** ()
- **update_59** ()
- **_update_thread_structure** ($comments, $pid, $depth, $structure)
- **update_60** ()
- **update_61** ()
- **update_62** ()
- **_update_next_thread** ($structure, $parent)
- **update_63** ()
- **update_64** ()
- **update_65** ()
- **update_66** ()
- **update_67** ()
- **update_68** ()
- **update_69** ()
- **update_70** ()
- **update_71** ()
- **update_72** ()

- **update_73** ()
- **update_74** ()
- **update_75** ()
- **update_76** ()
- **update_77** ()
- **update_78** ()
- **update_79** ()
- **update_80** ()
- **update_81** ()
- **update_82** ()
- **update_83** ()
- **update_84** ()
- **update_85** ()
- **update_86** ()
- **update_87** ()
- **update_88** ()
- **update_89** ()
- **update_90** ()
- **update_91** ()
- **update_92** ()
- **update_93** ()
- update_94 ()
- **update_95** ()
- **update_96** ()
- update_97 ()
- update_98 ()
- **update_99** ()
- **update_100** ()
- update_101 ()
- update_102 ()
- **update_103** ()
- update_104 ()
- **update_105** ()
- **update_106** ()
- **update_107** ()
- **update_108** ()
- **update_109** ()
- **update_110** ()
- **update_111** ()
- **update_112** ()
- **update_113** ()
- **update_114** ()
- **update_115** ()
- **update_116** ()
- **update_117** ()
- **update_118** ()
- **update_119** ()
- **update_120** ()
- **update_121** ()
- **update_122** ()

- **update_123** ()
- **update_124** ()
- **update_125** ()
- **update_126** ()
- **update_127** ()
- **update_128** ()
- update_129 ()
- **update_130** ()
- **update_131** ()
- **update_sql** ($sql)

Variables

- **$sql_updates**

4.18.1 Detailed Description

All incremental database updates performed between Drupal releases.

Definition in file updates.inc.

4.18.2 Function Documentation

4.18.2.1 update_101 ()

Definition at line 1642 of file updates.inc.

References _locale_get_iso639_list(), db_fetch_array(), db_fetch_object(), db_num_rows(), db_query(), and db_result().

```
1642                       {
1646   include_once 'includes/locale.inc';
1647   // get the language columns
1648   $result = db_query('SELECT * FROM {locales} LIMIT 1');
1649   $fields = array();
1650   if (db_num_rows($result)) {
1651     $columns = array_keys(db_fetch_array($result));
1652     foreach ($columns as $field) {
1653       $fields[$field] = 1;
1654     }
1655
1656     // but not the fixed fields
1657     unset($fields['lid'], $fields['location'], $fields['string']);
1658
1659     // insert locales
1660     $list = _locale_get_iso639_list();
1661     foreach ($fields as $key => $value) {
1662       if (db_result(db_query("SELECT COUNT(lid) FROM {locales} WHERE $key != ''"))) {
1663         if (isset($list[$key])) {
1664           $name = $list[$key][0];
1665           if ($key == 'en') {
1666             $key = 'en-local';
1667           }
```

```
1668          db_query("INSERT INTO {locales_meta} (locale, name) VALUES ('%s', '%s')", $key, $name);
1669        }
1670        else {
1671          db_query("INSERT INTO {locales_meta} (locale, name) VALUES ('%s', '%s')", $key, $key);
1672        }
1673      }
1674    }
1675
1676    // get all strings
1677    $result = db_query('SELECT * FROM {locales}');
1678    while ($entry = db_fetch_object($result)) {
1679      // insert string if at least one translation exists
1680      $test = 'return $entry->'. implode(' == "" && $entry->', array_keys($fields)) .' == "";';
1681      if (!eval($test)) {
1682        db_query("INSERT INTO {locales_source} (location, source) VALUES ('%s', '%s')", $entry->locat
1683        $lid = db_fetch_object(db_query("SELECT lid FROM {locales_source} WHERE location = '%s' AND s
1684        foreach ($fields as $key => $value) {
1685          // insert translation if non-empty
1686          if ($key == 'en') {
1687            $keynew = 'en-local';
1688          }
1689          else {
1690            $keynew = $key;
1691          }
1692          db_query("INSERT INTO {locales_target} (lid, translation, locale) VALUES (%d, '%s', '%s')",
1693        }
1694      }
1695    }
1696  }
1697
1698  $ret = array();
1699  $ret[] = update_sql("DROP TABLE {locales}");
1700  return $ret;
1701 }
```

4.18.2.2 update_102 ()

Definition at line 1703 of file updates.inc.

```
1703                        {
1707   return array(update_sql("INSERT INTO {system} (filename, name, type, description, status, throttle,
1708 }
```

4.18.2.3 update_104 ()

Definition at line 1739 of file updates.inc.

References variable_del(), variable_get(), and variable_set().

```
1739                        {
1743   $ret = array();
1744   if (variable_get('theme_default', 'xtemplate') == 'chameleon') {
1745     $ret[] = update_sql("DELETE FROM {system} WHERE name = 'chameleon'");
1746     $ret[] = update_sql("INSERT INTO {system} VALUES ('themes/chameleon/chameleon.theme','chameleon',
1747     $ret[] = update_sql("INSERT INTO {system} VALUES ('themes/chameleon/marvin/style.css','marvin','t
1748     if (variable_get("chameleon_stylesheet", "themes/chameleon/pure/chameleon.css") == "themes/chamel
1749       variable_set('theme_default', 'chameleon/marvin');
```

```
1750     }
1751    else {
1752       variable_set('theme_default', 'chameleon');
1753     }
1754   }
1755   elseif (variable_get('theme_default', 'xtemplate') == 'xtemplate') {
1756     $ret[] = update_sql("DELETE FROM {system} WHERE name = 'xtemplate'");
1757     $ret[] = update_sql("INSERT INTO {system} VALUES ('themes/bluemarine/xtemplate.xtmpl','bluemarine
1758     $ret[] = update_sql("INSERT INTO {system} VALUES ('themes/pushbutton/xtemplate.xtmpl','pushbutton
1759     $ret[] = update_sql("INSERT INTO {system} VALUES ('themes/engines/xtemplate/xtemplate.engine','xt
1760     if (variable_get('xtemplate_template', 'default') == 'pushbutton') {
1761       variable_set('theme_default', 'pushbutton');
1762     }
1763     else {
1764       variable_set('theme_default', 'bluemarine');
1765     }
1766
1767     // Convert old xtemplate settings to new theme system
1768     $settings = array();
1769     $convert = array('xtemplate_primary_links'   => 'primary_links',
1770                      'xtemplate_secondary_links' => 'secondary_links',
1771                      'xtemplate_search_box'      => 'toggle_search',
1772                      'xtemplate_avatar_node'     => 'toggle_node_user_picture',
1773                      'xtemplate_avatar_comment'  => 'toggle_comment_user_picture');
1774     foreach ($convert as $from => $to) {
1775       if (($value = variable_get($from, NULL)) != NULL) {
1776         $settings[$to] = $value;
1777         variable_del($from);
1778       }
1779     }
1780
1781     // Logo requires special treatment. Used to be an HTML tag, now it's a path to an image.
1782     if (($logo = variable_get('xtemplate_logo', NULL)) != NULL) {
1783       $match = array();
1784       // If logo was of the form <img src="..">, convert it.
1785       if (preg_match('@src=(?:["\']?)(.+?)(?:["\']?(?:>| ?/>))@i', $logo, $match)) {
1786         $settings['default_logo'] = 0;
1787         $settings['logo_path'] = $match[1];
1788       }
1789       variable_del('xtemplate_logo');
1790     }
1791
1792     if (count($settings) > 0) {
1793       variable_set('theme_settings', $settings);
1794     }
1795
1796     // These are not part of 'theme_settings'
1797     $convert = array('xtemplate_avatar_default' => 'user_picture_default',
1798                      'xtemplate_mission'        => 'site_mission');
1799     foreach ($convert as $from => $to) {
1800       if (($value = variable_get($from, NULL)) != NULL) {
1801         variable_set($to, $value);
1802         variable_del($from);
1803       }
1804     }
1805   }
1806   return $ret;
1807 }
```

4.18.2.4 update_129 ()

Definition at line 2345 of file updates.inc.

```
2345                      {
2349   $ret = array();
2350
2351   if ($GLOBALS['db_type'] == 'pgsql') {
2352     $ret[] = update_sql('DROP TABLE {search_total}');
2353
2354     $ret[] = update_sql("CREATE TABLE {search_total} (
2355               word varchar(50) NOT NULL default '',
2356               count float default NULL)");
2357     $ret[] = update_sql('CREATE INDEX {search_total}_word_idx ON {search_total}(word)');
2358
2362     include_once('modules/search.module');
2363     search_wipe();
2364   }
2365
2366   return $ret;
2367 }
```

4.18.2.5 update_94 ()

Definition at line 1261 of file updates.inc.

```
1261                      {
1265   $ret = array();
1266   if ($GLOBALS['db_type'] == 'pgsql') {
1267     $ret[] = update_sql('DROP FUNCTION "greatest"(integer, integer)');
1268     $ret[] = update_sql("
1269       CREATE FUNCTION greatest(integer, integer) RETURNS integer AS '
1270         BEGIN
1271           IF $2 IS NULL THEN
1272             RETURN $1;
1273           END IF;
1274           IF $1 > $2 THEN
1275             RETURN $1;
1276           END IF;
1277           RETURN $2;
1278         END;
1279       ' LANGUAGE 'plpgsql';
1280     ");
1281   }
1282   return $ret;
1283 }
```

4.18.2.6 update_97 ()

Definition at line 1343 of file updates.inc.

References db_fetch_object(), and db_query().

```
1343                      {
1347   $convert = array('node/view/' => 'node/', 'book/view/' => 'book/', 'user/view/' => 'user/');
1348   foreach ($convert as $from => $to) {
1349     $result = db_query("SELECT pid, src FROM {url_alias} WHERE src LIKE '%s%%'", $from);
1350     while ($alias = db_fetch_object($result)) {
1351       db_query("UPDATE {url_alias} SET src = '%s' WHERE pid = '%s'", str_replace($from, $to, $alias->
1352     }
1353   }
```

```
1354
1355   return array();
1356 }
```

4.18.2.7 update_98 ()

Definition at line 1358 of file updates.inc.

References db_fetch_object(), and db_query().

```
1358                       {
1362   $result = db_query("SELECT pid, src FROM {url_alias} WHERE src LIKE 'taxonomy/%%'");
1363   while ($alias = db_fetch_object($result)) {
1364     list(, $page, $op, $terms) = explode('/', $alias->src);
1365     if ($page == 'feed' || $page == 'page') {
1366       switch ($op) {
1367         case 'or':
1368           $new = 'taxonomy/term/'. str_replace(',', '+', $terms);
1369           break;
1370         case 'and':
1371           $new = 'taxonomy/term/'. $terms;
1372           break;
1373       }
1374       if ($new) {
1375         if ($page == 'feed') {
1376           $new .= '/0/feed';
1377         }
1378         db_query("UPDATE {url_alias} SET src = '%s' WHERE pid = '%s'", $new, $alias->pid);
1379       }
1380     }
1381   }
1382
1383   return array();
1384 }
```

4.19 xmlrpc.php File Reference

Variables

- **$functions** = module_invoke_all(xmlrpc)
- **$server** = new xmlrpc_server($functions)

4.19.1 Detailed Description

PHP page for handling incoming XML-RPC requests from clients.

Definition in file xmlrpc.php.

Index