Knop Documentation

Release knop9

Knop Project

December 07, 2012

CONTENTS

Contents:

CHAPTER

ONE

KNOP_BASE

class **knop_knoptype**

All Knop custom types should have this type as parent type. This is to be able to identify all registered knop types.

isknoptype()

isknoptype=(isknoptype)

class **knop_base**

Base data type for Knop framework. Contains common member tags. Used as boilerplate when creating the other types. All member tags and instance variables in this type are available in the other knop types as well.

_debug_trace()

_debug_trace=(_debug_trace::array)

debug_trace()

debug_trace=(debug_trace::array)

error_code()

error_code=(error_code)

error_lang()

error_lang=(error_lang)

error_msg(*error_code::integer =?*)

error_msg=(error_msg)

help(*html::boolean =?*, *xhtml::boolean =?*)

instance_unique()

instance_unique=(instance_unique)

instance_varname()

instance_varname=(instance_varname)

tagtime()

tagtime=(tagtime::integer)

tagtime_tagname()

tagtime_tagname=(tagtime_tagname::tag)

varname()

version()

version=(version)

xhtml(*params =?*)

CHAPTER

TWO

KNOP_CACHE

class `knop_cache`

A thread object acting as base to the different Knop cache methods. Methods:

- add Stores a cache for the supplied name Parameters:
  - name type = required, string,
  - content required, any kind of content,
  - expires optional, integer. Defaults to 600 seconds
- get Retrieves a cached content Parameter: name type = required, string
- getall Returns all cached content as a raw map. Useful for debugging
- remove Deletes a cached object Parameter: name type = required, string
- clear Removes all cached content

`active_tick`()

`add`(*name::string*, *content*, *expires::integer =?*)

`caches`()

`caches=(caches)`

`clear`()

`get`(*name::string*)

`getall`()

`purged`()

`purged=(purged)`

`remove`(*name::string*)

`version`()

`version=(version::date)`

`knop_cachedelete`(*-type::string*, *-session::string =?*, *-name::string =?*)

`knop_cachedelete`(*type::string*, *session::string =?*, *name::string =?*)

Deletes the cache for the specified type (and optionally name).

Parameters:

- type (required string) Page variables of the specified type will be deleted from cache.

- session (optional string) The name of an existing session storing the cache to be deleted.
- name (optional string) Extra name parameter used to isolate the cache storage from other sites on the same virtual hosts.

`knop_cachefetch` (*-type::string*, *-session::string =?*, *-name::string =?*, *-maxage::date =?*)

`knop_cachefetch` (*type::string*, *session::string =?*, *name::string =?*, *maxage::date =?*)

Recreates page variables from previously cached instances of the specified type, returns true if successful or false if there was no valid existing cache for the specified type. Caches are stored in a global variable named by host name and document root to isolate the storage of different hosts.

Knop_cachefetch calls the thread object `knop_cache` and can be replaced by direct calls to `knop_cache` if you don't want to get cached objects from a session.

Parameters:

- type (required string) Page variables of the specified type will be stored in cache.
- session (optional string) The name of an existing session to use for cache storage instead of the global storage.
- name (optional string) Extra name parameter to be able to isolate the cache storage from other sites on the same virtual hosts.
- maxage (optional date) Cache data older than the date/time specified in -maxage will not be used.

`knop_cachestore` (*-type::string*, *-expires::integer =?*, *-session::string =?*, *-name::string =?*)

`knop_cachestore` (*type::string*, *expires::integer =?*, *session::string =?*, *name::string =?*)

Stores all instances of page variables of the specified type in a cache object. Caches are stored in a global variable named by host name and document root to isolate the storage of different hosts.

Knop_cachestore calls the thread object `knop_cache` and can be replaced by direct calls to `knop_cache` if you don't want to store the cache in a session.

Parameters:

- type (required string)

  Page variables of the specified type will be stored in cache. Data types can be specified with or without namespace.

- expires (optional integer)

  The number of seconds that the cached data should be valid. Defaults to 600 (10 minutes).

- session (optional string)

  The name of an existing session to use for cache storage instead of the global storage.

- name (optional string)

  Extra name parameter to be able to isolate the cache storage from other sites on the same virtual hosts, or caches for different uses.

CHAPTER

THREE

KNOP_DATABASE

class **`knop_database`**

Custom type to interact with databases. Supports both MySQL and FileMaker datasources Lasso 9 version

`_unknowntag`(...)

Shortcut to field

`acceptDeserializedElement`(*d::serialization_element*)

Called when a knop_database object is retrieved from a session

`action_statement`()

`action_statement=(action_statement::string)`

`addrecord`(*-fields::array*, *-keyvalue::string =?*, *-inlinename::string =?*)

`addrecord`(*fields::array*, *keyvalue::string =?*, *inlinename::string =?*)

Add a new record to the database. A random string keyvalue will be generated unless a -keyvalue is specified.

Parameters:

- fields (required array)

  Lasso-style field values in pair array

- keyvalue (optional)

  If -keyvalue is specified, it must not already exist in the database. Specify -keyvalue = false to prevent generating a keyvalue.

- inlinename (optional)

  Defaults to autocreated inlinename.

`affected_count`()

`affected_count=(affected_count::integer)`

`affectedrecord_keyvalue`()

`affectedrecord_keyvalue=(affectedrecord_keyvalue::string)`

`capturesearchvars`()

capturesearchvars Internal.

`clearlocks`(*-user*)

`clearlocks`(*user*)

Release all record locks for the specified user, suitable to use when showing record list.

Parameters:

- user (required)

  The user to unlock records for.

current_record()

current_record=(current_record::integer)

database()

database=(database::string)

databaserows_map()

databaserows_map=(databaserows_map::map)

datasource_name()

datasource_name=(datasource_name::string)

db_connect()

db_connect=(db_connect::array)

db_registry()

db_registry=(db_registry)

deleterecord(*-keyvalue::string =?*, *-lockvalue::string =?*, *-user =?*)

deleterecord(*keyvalue::string =?*, *lockvalue::string =?*, *user =?*)
Deletes a specific database record.

Parameters:

- keyvalue (optional)

  Keyvalue is ignored if lockvalue is specified

- lockvalue (optional)

  Either keyvalue or lockvalue must be specified

- user (optional)

  If lockvalue is specified, user must be specified as well.

description()

description=(description::string)

error_code()
varname Returns the name of the variable that this type instance is stored in.

error_data()
Returns more info for those errors that provide such.

error_data=(error_data::map)

error_msg(*error_code::integer =?*)

errors_error_data()

errors_error_data=(errors_error_data::map)

field(*fieldname::string*, *recordindex::integer =?*, *index::integer =?*)
A shortcut to return a specific field from a single record result.

field_names (*table::string =?*, *types::boolean =?*)

Returns an array of the field names from the last database query. If no database query has been performed, a "-show" request is performed.

Parameters:

- table (optional)

  Return the field names for the specified table

- types (optional flag)

  If specified, returns a pair array with fieldname and corresponding Lasso data type.

field_names=(field_names::array)

field_names_map()

field_names_map=(field_names_map::map)

found_count()

found_count=(found_count::integer)

get(*index::integer*)

getrecord(*-keyvalue::string =?*, *-keyfield::string =?*, *-inlinename::string =?*, *-lock::boolean =?*, *-user =?*, *-sql::string =?*)

getrecord(*keyvalue::string =?*, *keyfield::string =?*, *inlinename::string =?*, *lock::boolean =?*, *user =?*, *sql::string =?*)

Returns a single specific record from the database, optionally locking the record. If the keyvalue matches multiple records, an error is returned.

Parameters:

- keyvalue (optional)

  Uses a previously set keyvalue if not specified. If no keyvalue is available, an error is returned unless -sql is used.

- keyfield (optional)

  Temporarily override of keyfield specified at oncreate

- inlinename (optional)

  Defaults to autocreated inlinename

- lock (optional flag)

  If flag is specified, a record lock will be set

- user (optional)

  The user who is locking the record (required if using lock)

- sql (optional)

  SQL statement to use instead of keyvalue. Must include the keyfield (and lockfield if locking is used).

host()

host=(host::array)

inlinename()

inlinename=(inlinename::string)

isfilemaker()

isfilemaker=(isfilemaker::boolean)

keyfield()

keyfield=(keyfield::string)

keyvalue()

keyvalue=(keyvalue::string)

lock_expires()

lock_expires=(lock_expires::integer)

lock_seed()

lock_seed=(lock_seed::string)

lockfield()

lockfield=(lockfield::string)

lockvalue()

lockvalue=(lockvalue::string)

lockvalue_encrypted()

lockvalue_encrypted=(lockvalue_encrypted::string)

maxrecords_value()

maxrecords_value=(maxrecords_value)

message()

message=(message::string)

next()

Increments the record pointer, returns true if there are more records to show, false otherwise.

Useful as an alternative to a regular records loop:

```
$database -> select;
while($database -> next);
        $database -> field( \'name\');\'<br>\';
/while;
```

oncreate(*-database::string*, *-table::string*, *-host::array =?*, *-username::string =?*, *-password::string =?*, *-keyfield::string =?*, *-lockfield::string =?*, *-user =?*, *-validate::boolean =?*)

oncreate(*database::string*, *table::string*, *-host::array =?*, *-username::string =?*, *-password::string =?*, *-keyfield::string =?*, *-lockfield::string =?*, *-user =?*, *-validate::boolean =?*)

oncreate(*database::string*, *table::string*, *host::array =?*, *username::string =?*, *password::string =?*, *keyfield::string =?*, *lockfield::string =?*, *user =?*, *validate::boolean =?*)

password()

password=(password::string)

querytime()

querytime=(querytime::integer)

recorddata(*recordindex::integer =?*)

A map containing all fields, only available for single record results.

`recorddata=(recorddata::map)`

`records` (*-inlinename::string =?*)

`records` (*inlinename::string =?*)
Returns all found records as a knop_databaserows object.

`records_array` ()

`records_array=(records_array::staticarray)`

`reset` ()

`resultset_count` (*inlinename::string =?*)

`resultset_count_map` ()

`resultset_count_map=(resultset_count_map::map)`

`saverecord` (*-fields::array, -keyfield::string =?, -keyvalue::string =?, -lockvalue::string =?, -keeplock::boolean =?, -user =?, -inlinename::string =?*)

`saverecord` (*fields::array, keyfield::string =?, keyvalue::string =?, lockvalue::string =?, keeplock::boolean =?, user =?, inlinename::string =?*)
Updates a specific database record.

Parameters:

- fields (required array)

  Lasso-style field values in pair array

- keyfield (optional)

  Keyfield is ignored if lockvalue is specified

- keyvalue (optional)

  Keyvalue is ignored if lockvalue is specified

- lockvalue (optional)

  Either keyvalue or lockvalue must be specified

- keeplock (optional flag)

  Avoid clearing the record lock when saving. Updates the lock timestamp.

- user (optional)

  If lockvalue is specified, user must be specified as well

- inlinename (optional)

  Defaults to autocreated inlinename.

`scrubKeywords` (*input*)

`scrubKeywords` (*input::trait_queriable*)

`searchparams` ()

`searchparams=(searchparams::array)`

`select` (*-search::array =?, -sql::string =?, -keyfield::string =?, -keyvalue::string =?, -inlinename::string =?*)

select (*search::array =?*, *sql::string =?*, *keyfield::string =?*, *keyvalue =?*, *inlinename::string =?*)
perform database query, either Lasso-style pair array or SQL statement. ->recorddata returns a map with all the fields for the first found record. If multiple records are returned, the records can be accessed either through ->inlinename or ->records_array.

Parameters:

- search (optional array)

  Lasso-style search parameters in pair array

- sql (optional string)

  Raw sql query

- keyfield (optional)

  Overrides default keyfield, if any

- keyvalue (optional)
- inlinename (optional)

  Defaults to autocreated inlinename

serializationElements ()
Called when a knop_database object is stored in a session

sethost (*host::array*)
Creates or changes the DB inline host setting.

settable (*table::string*)
Changes the current table for a database object. Useful to be able to create database objects faster by copying an existing object and just change the table name. This is a little bit faster than creating a new instance from scratch, but no table validation is performed. Only do this to add database objects for tables within the same database as the original database object.

shown_count ()

shown_count=(shown_count::integer)

shown_first ()

shown_first=(shown_first::integer)

shown_last ()

shown_last=(shown_last::integer)

size ()

skiprecords_value ()

skiprecords_value=(skiprecords_value::integer)

table ()

table=(table::string)

table_names ()
Returns an array with all table names for the database.

timestampfield ()

timestampfield=(timestampfield::string)

timestampvalue ()

timestampvalue=(timestampvalue::string)

user()

user=(user)

username()

username=(username::string)

version()

version=(version)

class **knop_databaserows**

Custom type to return all record rows from knop_database. Used as output for knop_database->records Lasso 9 version

current_record()

current_record=(current_record::integer)

description()

description=(description::string)

field(*fieldname::string*, *recordindex::integer =?*, *index::integer =?*)

Return an individual field value.

field_names()

field_names=(field_names::array)

field_names_map()

field_names_map=(field_names_map::map)

get(*index::integer*)

next()

Increments the record pointer, returns true if there are more records to show, false otherwise.

onconvert(*recordindex::integer =?*)

Output the current record as a plain array of field values.

oncreate(*records_array::staticarray*, *field_names::array*)

Create a record rows object.

Parameters:

- records_array (array)

  Array of arrays with field values for all fields for each record of all found records

- field_names (array)

  Array with all the field names

records_array()

records_array=(records_array::staticarray)

size()

summary_footer(*fieldname::string*)

Returns true if the specified field name will change in the following record, or if we are at the last record.

summary_header (*fieldname::string*)
Returns true if the specified field name has changed since the previous record, or if we are at the first record.

version ()

version=(version::date)

class **knop_databaserow**
Custom type to return individual record rows from knop_database. Used as output for knop_database->get Lasso 9 version

description ()

description=(description::string)

field (*fieldname::string*, *index::integer =?*)
Return an individual field value.

field_names ()

field_names=(field_names::array)

onconvert ()
Output the record as a plain array of field values.

oncreate (*-record_array::staticarray*, *-field_names::array*)

oncreate (*record_array::staticarray*, *field_names::array*)
Create a record row object.

Parameters:

- record_array (array)

  Array with field values for all fields for the record

- field_names (array)

  Array with all the field names, should be same size as -record_array

record_array ()

record_array=(record_array::staticarray)

version ()

version=(version::date)

CHAPTER

FOUR

KNOP_FORM

class **`knop_form`**

`_unknowntag`(*index::integer =?*)

`actionpath`()

`actionpath=(actionpath)`

`adderror`(*fieldname*)
Adds the name for a field that has validation error, used for custom field validation. calls form -> validate first if needed

`addfield`(*-type*, *-name =?*, *-label =?*, *-value =?*, *-id =?*, *-dbfield =?*, *-hint =?*, *-options =?*, *-multiple =?*, *-linebreak =?*, *-default =?*, *-size::integer =?*, *-maxlength::integer =?*, *-rows::integer =?*, *-cols::integer =?*, *-focus =?*, *-class =?*, *-labelclass =?*, *-disabled =?*, *-raw =?*, *-confirmmessage =?*, *-required =?*, *-validate =?*, *-filter =?*, *-nowarning =?*, *-op::string =?*, *-logical_op::string =?*, *-after =?*)

`addfield`(*type::string*, *name::string =?*, *label =?*, *value =?*, *id =?*, *dbfield =?*, *hint =?*, *options =?*, *default =?*, *size =?*, *maxlength =?*, *rows =?*, *cols =?*, *class =?*, *labelclass =?*, *raw =?*, *confirmmessage =?*, *validate =?*, *filter =?*, *after =?*, *required =?*, *nowarning =?*, *op::string =?*, *logical_op::string =?*, *multiple =?*, *linebreak =?*, *focus =?*, *disabled =?*)
Inserts a form element in the form.

Parameters:

- type (required)

  Supported types are listed in form -> 'validfieldtypes'. Also custom field types addbutton, savebutton or deletebutton are supported (translated to submit buttons with predefined names).

  For the field types html, fieldset and legend use -value to specify the data to display for these fields. A legend field automatically creates a fieldset (closes any previously open fieldsets). Use fieldset with -value = false to close a fieldset without opening a new one.

- name (optional)

  Required for all input types except addbuton, savebutton, deletebutton, fieldset, legend and html

- id (optional)

  id for the html object, will be autogenerated if not specified

- dbfield (optional)

Corresponding database field name (name is used if dbfield is not specified), or null/emtpy string if ignore this field for database

- value (optional)

  Initial value for the field

- hint (optional)

  Optional gray hint text to show in empty text field

- options (optional)

  For select, checkbox and radio, must be array or set. For select, the array can contain - optgroup = label to create an optiongroup.

- multiple (optional flag)

  Used for select

- linebreak (optional flag)

  Put linebreaks between checkbox and radio values

- default (optional)

  Default text to display in a popup menu, will be selected (with empty value) if no current value is set. Is followed by an empty option.

- label (optional)

  Text label for the field

- size (optional)

  Used for text and select

- maxlength (optional)

  Used for text

- rows (optional)

  Used for textarea

- cols (optional)

  Used for textarea

- focus (optional flag)

  The first text field with this parameter specified will get focus when page loads

- class (optional)
- disabled (optional flag)

  The form field will be rendered as disabled

- raw (optional)

  Raw attributes that will be put in the html tag

- confirmmessage (optional)

  Message to show in submit/reset confirm dialog (delete button always shows confirm dialog)

- required (optional flag)

  If specified then the field must not be empty (very basic validation)

- validate (optional)

  Compound expression to validate the field input. The input can be accessed as params inside the expression which should either return true for valid input or false for invalid, or return 0 for valid input or a non-zero error code or error message string for invalid input.

- filter (optional)

  Compound expression to filter the input before it is loaded into the form by ->loadfields. The field value can be accessed as params inside the expression which should return the filtered field value. -filter is applied before validation.

- nowarning (optional flag)

  If specified then changing the field will not trigger an unsaved warning

- after (optional)

  Numeric index or name of field to insert after

afterhandler(*headscript::string =?*, *endscript::string =?*)
Internal member tag. Adds needed javascripts through an atend handler that will be processed when the entire page is done.

Parameters:

- headscript (optional)

  A single script, will be placed before </head> (or at top of page if </head> is missing)

- endscript (optional)

  Multiple scripts (no duplicates), will be placed before </body> (or at end of page if </body> is missing)

backtickthis(*n*)
Internal method used by searchfields to prep db field names

buttontemplate()

buttontemplate=(buttontemplate::string)

class()

class=(class::string)

clearfields()
Empties all form field values

clientparams()

clientparams=(clientparams::staticarray)

copyfield(*name*, *newname*)
Copies a form field to a new name.

database()

database=(database)

db_keyvalue()

db_keyvalue=(db_keyvalue)

db_lockvalue()

db_lockvalue=(db_lockvalue)

enctype()

enctype=(enctype)

end_rendered()

end_rendered=(end_rendered)

entersubmitblock()

entersubmitblock=(entersubmitblock)

error_code()

error_lang()

error_lang=(error_lang)

errorclass()

errorclass=(errorclass::string)

errors()

Returns an array with fields that have input errors, or empty array if no errors or form has not been validated

errors=(errors)

exceptionfieldtypes()

exceptionfieldtypes=(exceptionfieldtypes::map)

fields()

fields=(fields::array)

fieldset()

fieldset=(fieldset)

fieldsource()

fieldsource=(fieldsource)

formaction()

formaction=(formaction)

formbutton()

formbutton=(formbutton)

formid()

formid=(formid)

formmode()

Returns add or edit after form -> init has been called

formmode=(formmode)

getbutton()

Returns what button was clicked on the form on the previous page. Assumes that submit buttons are named button_add etc. Returns add, update, delete, cancel or any custom submit button name that begins with button_.

getlabel(*name::string*)

Returns the label for a form field.

getvalue(*name::string*, *-index::integer =?*)

getvalue(*name::string*, *index::integer =?*)
Returns the current value of a form field. Returns an array for repeated form fields.

id()

id=(id)

init(*-get =?*, *-post =?*, *-keyvalue =?*)

init(*get =?*, *post =?*, *keyvalue =?*)
Initiates form to grab keyvalue and set formmode if we have a database connected to the form. Does nothing if no database is specified.

isvalid()
Returns the result of form -> validate (true/false) without performing the validation again (unless it hasn't been performed already)

keyparamname()

keyparamname=(keyparamname::string)

keys()
Returns an array of all field names

keyvalue()

legend()

legend=(legend)

loadfields(*-params =?*, *-post =?*, *-get =?*, *-inlinename =?*, *-database =?*)

loadfields(*params =?*, *post =?*, *get =?*, *inlinename =?*, *database =?*)
Overwrites all field values with values from either database, action_params or explicit -params. Autodetects based on current lasso_currentaction.

Parameters:n -params (optional)

Array or map to take field values from instead of database or submit (using dbnames)

-get (optional flag)

Only getparams will be used

-post (optional flag)

Only postparams will be used

-inlinename (optional)

The first record in the result from the specified inline will be used as field values

-database (optional)

If a database object is specified, the first record from the latest search result of the database object will be used. If -database is specified as flag (no value) and the form object has a database object attached to it, that database object will be used.

lockvalue()

lockvalue_decrypted()

method()

method=(method)

name()

name=(name)

noautoparams()

noautoparams=(noautoparams)

noscript()

noscript=(noscript)

onconvert()
Outputs the form data in very basic form, just to see what it contains

oncreate(*-formaction =?*, *-method =?*, *-name =?*, *-id =?*, *-raw =?*, *-actionpath =?*, *-fieldset::boolean =?*, *-legend =?*, *-entersubmitblock =?*, *-noautoparams =?*, *-template::string =?*, *-buttontemplate::string =?*, *-required::string =?*, *-class::string =?*, *-errorclass::string =?*, *-unsavedmarker::string =?*, *-unsavedmarkerclass::string =?*, *-unsavedwarning::string =?*, *-keyparamname::string =?*, *-noscript =?*, *-database =?*)

oncreate(*formaction =?*, *method =?*, *name =?*, *id =?*, *raw =?*, *actionpath =?*, *fieldset::boolean =?*, *legend =?*, *entersubmitblock =?*, *noautoparams =?*, *template::string =?*, *buttontemplate::string =?*, *required::string =?*, *class::string =?*, *errorclass::string =?*, *unsavedmarker::string =?*, *unsavedmarkerclass::string =?*, *unsavedwarning::string =?*, *keyparamname::string =?*, *noscript =?*, *database =?*)

Parameters:n

- formaction (optional)

  The action attribute in the form html tag

- method (optional)

  Defaults to post

- name (optional)
- id (optional)
- raw (optional)

  Anything in this parameter will be put in the opening form tag

- actionpath (optional)

  Knop action path

- fieldset (optional)
- legend (optional string)

  Legend for the entire form - if specified, a fieldset will also be wrapped around the form

- entersubmitblock (optional)
- noautoparams (optional)
- template (optional string)

  Html template, defaults to #label# #field##required#<br>

- buttontemplate (optional string)

  Html template for buttons, defaults to #field# but uses -template if specified

- required (optional string)

  Character(s) to display for required fields (used for #required#), defaults to *

- class (optional string)

  CSS class name that will be used for the form element, default none

- errorclass (optional string)

  CSS class name that will be used for the label to highlight input errors, if not defined style="color: red" will be used

- unsavedmarker (optional string)

  ID for html element that should be used to indicate when the form becomes dirty.

- unsavedmarkerclass (optional string)

  Class name to use for the html element. Defaults to "unsaved".

- unsavedwarning (optional string)
- keyparamname (optional)
- noscript (optional flag)

  If specified, don't inject any javascript in the form. This will disable all client side functionality such as hints, focus and unsaved warnings.

- database (optional database)

  Optional database object that the form object will interact with

`process`(*user =?*, *lock =?*, *keyvalue =?*)
Automatically handles a form submission and handles add, update, or delete. Requires that a database object is specified for the form

`raw`()

`raw=(raw)`

`removeField`(*-name::string*)

`removeField`(*name::string*)
Removes all form elements with the specified name from the form

`render_fieldset2_open`()

`render_fieldset2_open=(render_fieldset2_open)`

`render_fieldset_open`()

`render_fieldset_open=(render_fieldset_open)`

`renderform`(*-name::string =?*, *-from =?*, *-to =?*, *-type =?*, *-excludetype =?*, *-legend::string =?*, *-start::boolean =?*, *-end::boolean =?*, *-onlyformcontent::boolean =?*, *-xhtml::boolean =?*)

`renderform`(*name::string =?*, *from =?*, *to =?*, *type =?*, *excludetype =?*, *legend::string =?*, *xhtml::boolean =?*, *onlyformcontent::boolean =?*)
Outputs HTML for the form fields, a specific field, a range of fields or all fields of a specific type. Also inserts all needed javascripts into the page. Use form -> setformat first to specify the html format, otherwise default format #label# #field##required#<br> is used.

Parameters:

- name (optional)

  Render only the specified field

- from (optional)

Render form fields from the specified number index or field name. Negative number count from the last field.

- to (optional)

  Render form fields to the specified number index or field name. Negative number count from the last field.

- type (optional)

  Only render fields of this or these types (string or array)

- excludetype (optional)

  Render fields except of this or these types (string or array)

- legend (optional)

  Groups the rendered fields in a fieldset and outputs a legend for the fieldset

- start (optional)

  Only render the starting <form> tag

- end (optional)

  Only render the ending </form> tag

- xhtml (optional flag)

  XHTML valid output

renderformend(*xhtml::boolean =?*)

renderformstart(*xhtml::boolean =?*)

renderhtml(*-name =?, -from =?, -to =?, -type =?, -excludetype =?, -legend::string =?, -xhtml::boolean =?*)

renderhtml(*name::string =?, from =?, to =?, type =?, excludetype =?, legend::string =?, xhtml::boolean =?*)

Outputs form data as plain HTML, a specific field, a range of fields or all fields of a specific type. Some form field types are excluded, such as submit, reset, file etc. Use form -> setformat first to specify the html format, otherwise default format #label#: #field#<br> is used.

Parameters:

- name (optional)

  Render only the specified field

- from (optional)

  Render fields from the specified number index or field name

- to (optional)

  Render fields to the specified number index or field name

- type (optional)

  Only render fields of this or these types (string or array)

- excludetype (optional)

  Render fields except of this or these types (string or array)

- legend (optional)

  Groups the rendered fields in a fieldset and outputs a legend for the fieldset

- xhtml (optional flag)

  XHTML valid output

required()

required=(required::string)

reseterrors()
Empties the error array as if no errors was found

resetfields()
Resets all form field values to their initial values

search_type()

search_type=(search_type)

searchfields(*-sql::boolean =?*, *-params::boolean =?*)

searchfields(*sql::boolean =?*, *params::boolean =?*)
Returns an array with fieldname = value, or optionally SQL string to be used in a search inline. form -> loadfields must be called first.

setformat(*-template::string =?*, *-buttontemplate::string =?*, *-required::string =?*, *-legend::string =?*, *-class::string =?*, *-errorclass::string =?*, *-unsavedmarker::string =?*, *-unsavedmarkerclass::string =?*, *-unsavedwarning::string =?*)

setformat(*template::string =?*, *buttontemplate::string =?*, *required::string =?*, *legend::string =?*, *class::string =?*, *errorclass::string =?*, *unsavedmarker::string =?*, *unsavedmarker-class::string =?*, *unsavedwarning::string =?*)
Defines a html template for the form.

Parameters:

-template (optional string)

Html template, defaults to #label# #field##required#<br>

-buttontemplate (optional string)

Html template for buttons, defaults to #field#

-required (optional string)

Character(s) to display for required fields (used for #required#), defaults to *

-legend (optional string)

Legend for the entire form - if specified, a fieldset will also be wrapped around the form

-class (optional string)

CSS class name that will be used for the form element, default none

-errorclass (optional string)

CSS class name that will be used for the label to highlight input errors, if not defined style="color: red" will be used

-unsavedmarker (optional string)

-unsavedmarkerclass (optional string)

-unsavedwarning (optional string)

setparam(*-name::string*, *-param::string*, *-value*, *-index::integer =?*)

setparam (*name::string*, *param::string*, *value*, *index::integer =?*)
Sets the param content for a form field.

setvalue (*name*, *value =?*, *index::integer =?*)
Sets the value for a form field. Either `form -> (setvalue:  fieldname = newvalue)` or `form -> (setvalue:  -name = fieldname, -value = newvalue)`

start_rendered ()

start_rendered=(start_rendered)

template ()

template=(template::string)

unsavedmarker ()

unsavedmarker=(unsavedmarker)

unsavedmarkerclass ()

unsavedmarkerclass=(unsavedmarkerclass)

unsavedwarning ()

unsavedwarning=(unsavedwarning::string)

updatefields (*sql::boolean =?*)
Returns a pair array with fieldname = value, or optionally SQL string to be used in an update inline. form -> loadfields must be called first.

validate ()
Performs validation and fills a transient array with field names that have input errors. form -> loadfields must be called first.

validfieldtypes ()

validfieldtypes=(validfieldtypes::map)

version ()

version=(version)

CHAPTER

FIVE

KNOP_GRID

class **`knop_grid`**

Custom type to handle data grids (record listings).

`addfield` (*-name::string =?*, *-label::string =?*, *-dbfield::string =?*, *-width::integer =?*, *-class::string =?*, *-raw =?*, *-url::string =?*, *-keyparamname::string =?*, *-defaultsort =?*, *-nosort::boolean =?*, *-template =?*, *-quicksearch =?*)

`addfield` (*name::string =?*, *label::string =?*, *dbfield::string =?*, *width::integer =?*, *class =?*, *raw =?*, *url =?*, *keyparamname::string =?*, *defaultsort =?*, *nosort::boolean =?*, *template =?*, *quicksearch =?*)

deprecated use insert instead

`addurlarg` (*field::string*, *value =?*)

`class` ()

`class=(class::string)`

`database` ()

`database=(database)`

`dbfieldmap` ()

`dbfieldmap=(dbfieldmap::map)`

`defaultsort` ()

`defaultsort=(defaultsort::string)`

`error_lang` ()

`error_lang=(error_lang)`

`fields` ()

`fields=(fields::array)`

`footer` ()

`footer=(footer::string)`

`insert` (*-name::string =?*, *-label::string =?*, *-dbfield::string =?*, *-width::integer =?*, *-class::string =?*, *-raw =?*, *-url::string =?*, *-keyparamname::string =?*, *-defaultsort =?*, *-nosort::boolean =?*, *-template =?*, *-quicksearch =?*)

`insert` (*name::string =?*, *label::string =?*, *dbfield::string =?*, *width::integer =?*, *class =?*, *raw =?*, *url =?*, *keyparamname::string =?*, *defaultsort =?*, *nosort::boolean =?*, *template =?*, *quicksearch =?*)

Adds a column to the record listing.

Parameters:

- name (optional) Name of the field. If not specified, the field will be omitted from the grid. Useful to be able to quicksearch in fields not shown in the grid. In that case -dbfield must be specified.
- label (optional) Column heading
- dbfield (optional) Corresponding database field name (name is used if dbfield is not specified)
- width (optional) Pixels (CSS width)
- url (optional) Columns will be linked with this url as base. Can contain #value# for example to create clickable email links.
- keyparamname (optional) Param name to use instead of the default -keyvalue for edit links
- defaultsort (optional flag) This field will be the default sort field
- nosort (optional flag) The field header should not be clickable for sort
- template (optional) Either string to format values, compound expression or map containing templates to display individual values in different ways, use -default to display unknown values, use #value# to insert the actual field value in the template.

  If a compound expression is specified, the field value is passed as param to the expression and can be accessed as params. Example expressions:

  ```
  {return: params} to return just the field value as is
  {return: (date: (field: "moddate")) -> (format: "%-d/%-m")} to return a specific fi
  ```
- quicksearch (optional flag) If specified, the field will be used for search with quicksearch. If not a boolean the value will be used as the searchfield name

(Previously called addfield)

lang()
Returns a reference to the language object

lang=(lang)

lastpage()

nav()

nav=(nav)

nosort()

nosort=(nosort)

numbered()

numbered=(numbered)

onassign(*value*)

oncreate(*-database::knop_database*, *-nav =?*, *-quicksearch =?*, *-rawheader::string =?*, *-class::string =?*, *-id::string =?*, *-nosort =?*, *-language::string =?*, *-numbered =?*, *-rowsorting::boolean =?*)

oncreate(*database::knop_database*, *nav =?*, *quicksearch =?*, *rawheader::string =?*, *class::string =?*, *id::string =?*, *nosort =?*, *language::string =?*, *numbered =?*, *rowsorting::boolean =?*)

Parameters:

- database (required database) Database object that the grid object will interact with
- nav (optional nav) Navigation object to interact with

- quicksearch (optional) Label text for the quick search field
- rawheader (optional) Extra html to be inserted in the grid header
- class (optional) Extra classes to be inserted in the grid header. The standard class "grid" is always inserted
- id (optional) Creates a custom id used for table, quicksearch and quicksearch_reset
- nosort (optional flag) Global setting for the entire grid (overrides column specific sort options)
- language (optional) Language to use for the grid, defaults to the browser's preferred language
- numbered (optional flag or integer) If specified, pagination links will be shown as page numbers instead of regular prev/next links. Defaults to 6 links, specify another number (minimum 6) if more numbers are wanted. Can be specified in ->renderhtml as well.

page()

page=(page::integer)

page_skiprecords(*maxrecords::integer*)

Converts current page value to a skiprecords value to use in a search.

Parameters:

- maxrecords (required integer) Needed to be able to do the calculation. Maxrecords_value can not be taken from the database object since that value is not available until after performing the search

qs_id()

qs_id=(qs_id::string)

qsr_id()

qsr_id=(qsr_id::string)

quicksearch(*-sql::boolean =?*, *-contains::boolean =?*, *-value::boolean =?*, *-removedotbackticks::boolean =?*)

quicksearch(*sql::boolean =?*, *contains::boolean =?*, *value::boolean =?*, *removedotbackticks::boolean =?*)

Returns a pair array with fieldname = value to use in a search inline. If you specify several fields in the grid as -quicksearch (visible or not), they will be treated as if they were one single concatenated field. Quicksearch will take each word entered in the search field and search for them in the combined set of quicksearch fields, performing a "word begins with" match (unless you specify -contains when calling -> quicksearch).

So if you enter dev joh it will find records with firstname = Johan, occupation = Developer.

If you're familiar with how FileMaker performs text searches, this is the way quicksearch tries to behave.

Parameters:

- sql (optional flag) Return an SQL string for the search parameters instead.
- contains (optional flag) Perform a simple contains search instead of emulating "word begins with" search
- value (optional flag) Output just the search value of the quicksearch field instead of a pair array or SQL string
- removedotbackticks (optional flag) Use with -sql for backward compatibility for fields that contain periods. If you use periods in a fieldname then you cannot use a JOIN in Knop.

quicksearch=(quicksearch::string)

quicksearch_fields()

quicksearch_fields=(quicksearch_fields::array)

quicksearch_form()

quicksearch_form=(quicksearch_form)

quicksearch_form_reset()

quicksearch_form_reset=(quicksearch_form_reset)

rawheader()

rawheader=(rawheader::string)

renderfooter(*-end::boolean =?*, *-numbered =?*, *-xhtml::boolean =?*)

renderfooter(*end::boolean =?*, *numbered =?*, *xhtml::boolean =?*)
Outputs the footer of the grid with the prev/next links and information about found count. Automatically included by ->renderhtml

Parameters:

- end (optional flag) Also output closing </table> tagn
- numbered (optional flag or integer) If specified, pagination links will be shown as page numbers instead of regular prev/next links. Defaults to 6 links, specify another number (minimum 6) if more numbers are wanted.

renderheader(*-start::boolean =?*, *-xhtml::boolean =?*, *-startwithfooter::boolean =?*)

renderheader(*start::boolean =?*, *xhtml::boolean =?*, *startwithfooter::boolean =?*)
Outputs the header of the grid with the column headings. Automatically included by ->renderhtml.

Parameters:

- start (optional flag) Also output opening <table> tag

renderhtml(*-inlinename =?*, *-xhtml::boolean =?*, *-numbered =?*, *-startwithfooter::boolean =?*)

renderhtml(*inlinename =?*, *xhtml::boolean =?*, *numbered =?*, *startwithfooter::boolean =?*)
Outputs the complete record listing. Calls renderheader, renderlisting and renderfooter as well. If 10 records or more are shown, renderfooter is added also just below the header.

Parameters:

- inlinename (optional) If not specified, inlinename from the connected database object is used
- numbered (optional flag or integer) If specified, pagination links will be shown as page numbers instead of regular prev/next links. Defaults to 6 links, specify another number (minimum 6) if more numbers are wanted.

renderlisting(*inlinename =?*, *xhtml::boolean =?*)
Outputs just the actual record listing. Is called by renderhtml.

Parameters:

- inlinename (optional) If not specified, inlinename from the connected database object is used

rowsorting()

rowsorting=(rowsorting)

sortdescending()

sortdescending=(sortdescending)

sortfield()

sortfield=(sortfield::string)

sortparams(*-sql::boolean =?*, *-removedotbackticks::boolean =?*)

sortparams(*sql::boolean =?*, *removedotbackticks::boolean =?*)

Returns a Lasso-style pair array with sort parameters to use in the search inline.

Parameters:

- sql (optional)
- removedotbackticks (optional flag) Use with -sql for backward compatibility for fields that contain periods. If you use periods in a fieldname then you cannot use a JOIN in Knop.

tbl_id()

tbl_id=(tbl_id::string)

urlargs(*except =?*, *prefix =?*, *suffix =?*)

Returns all get params that begin with - as a query string, for internal use in links in the grid.

Parameters:

- except (optional) Exclude these parameters (string or array)
- prefix (optional) For example ? or &amp; to include at the beginning of the querystring
- suffix (optional) For example &amp; to include at the end of the querystring

version()

version=(version)

CHAPTER

SIX

KNOP_LANG

class **`knop_lang`**

A knop_lang object holds all language strings for all supported languages. Strings are stored under a unique text key, but the same key is of course used for the different language versions of the same string. Strings can be separated into different variables if it helps managing them for different contexts.

When the language of a string object is set, that language is used for all subsequent requests for strings until another language is set. All other instances on the same page that don't have a language set will also use the same language.

If no language is set, knop_lang uses the browser's preferred language if it's available in the knop_lang object, otherwise it defaults to the first language (unless a default language has been set for the instance).

`_unknowntag` (*-language =?*, *-replace =?*)

`_unknowntag` (*language =?*, *replace =?*)

Returns the language string for the specified text key

= shortcut for getstring.

Parameters:

- language (optional) see getstring(-language).
- replace (optional) see getstring(-replace).

`addlanguage` (*-language::string*, *-strings::map =?*)

`addlanguage` (*language::string*, *strings::map =?*)

Adds a map with language strings for an entire language. Replaces all existing language strings for that language.

Parameters:

- language (required) The language to add.
- strings (required) Complete map of key = value for the entire language.

`browserlanguage` ()

Autodetects and returns the most preferred language out of all available languages as specified by the browser's accept-language q-value.

`currentlanguage` ()

`currentlanguage=(currentlanguage::string)`

`defaultlanguage` ()

`defaultlanguage=(defaultlanguage::string)`

`getstring` (*-key::integer*, *-language::string =?*, *-replace =?*)

`getstring` (*key*, *language::string =?*, *replace =?*)
Returns a specific text string in the language that has previously been set for the instance.If no language has been set, the browser's preferred language will be used unless another instance on the same page has a language set using ->setlanguage.

If the string is not available in the chosen language and -fallback was specified, the string for the language that was first specified for that key will be returned.

Parameters:

- key (required) textkey to return the string for.
- language (optional) to return a string for a specified language (temporary override).
- replace (optional) single value or array of values that will be used as substitutions for placeholders #1#, #2# etc in the returned string, in the order they appear. Replacements can be compund expressions, which will be executed. Can also be map or pair array, and in that case the left hand element of the map/array will be replaced by the right hand element.

`getstring` (*key::integer*, *-language::string =?*, *-replace =?*)

`getstring` (*key::string*, *-language::string =?*, *-replace =?*)

`insert` (*language::string*, *key::string*, *value::string*)
Adds an individual language string.

Parameters:

- language (required) The language for the string.
- key (required) Textkey to store the string under. Replaces any existing key for the same language.
- value (required) The actual string (can also be compound expression). Can contain replacement tokens #1#, #2# etc.

`keys` ()
Returns array of all text keys in the string object.

`keys=(keys)`

`languages` (*language =?*)
Returns an array of all available languages in the string object (out of the languages in the -language array if specified).

Parameters:

- language (optional) string or array of strings.

`onconvert` ()

`oncreate` (*-default::string =?*)

`oncreate` (*default::string =?*, *fallback::boolean =?*, *debug::boolean =?*)
Creates a new instance of knop_lang.

Parameters:

- default (optional) Default language.
- fallback (optional) If specified, falls back to default language if key is missing.

`setlanguage` (*language::string*)
Sets the current language for the string object. Also affects other instances on the same page that do not have an explicit language set.

strings()

strings=(strings::map)

validlanguage(*-language::string*)

validlanguage(*language::string*)
Checks if a specified language exists in the string object, returns true or false.

validlanguage(*void*)

version()

version=(version::date)

CHAPTER

SEVEN

KNOP_NAV

class knop_nav
Experimental new leaner version of custom type to handle site navigation menu

acceptDeserializedElement(*d::serialization_element*)
Called when a knop_user object is retrieved from a session

actionconfigfile()
Shortcut to filename: actcfg.

actionconfigfile_didrun()

actionconfigfile_didrun=(actionconfigfile_didrun::string)

actionfile()
Shortcut to filename: act.

actionpath()
Returns current path as array.

actionpath=(actionpath::string)

addchildren(*-path::string*, *-children::knop_nav*)

addchildren(*path::string*, *children::knop_nav*)
Add nav object as children to specified key path, replacing the current children if any.

children(*-path*)

children(*path =?*)
Return reference to the children of the current navigation object map, or for the specified path

class()

class=(class::string)

configfile()
Shortcut to filename: cfg.

contentfile()
Shortcut to filename: cnt.

currentclass()

currentclass=(currentclass::string)

currentmarker()

currentmarker=(currentmarker::string)

data(*-path::string =?*, *-type::string =?*)

data (*path::string =?*, *type::string =?*)
Returns data object that can be stored for the current nav location (or specified nav location).

Parameters:

- path (optional)
- type (optional string)

  Force a certain return type. If the stored object doesn´t match the specified type, an empty instance of the type is returned. That way the data can be filtered by type without having to use conditionals to check the type before.

default ()

default=(default::string)

directorytree (*basepath::string =?*, *firstrun::boolean =?*)
Returns a map of all existing knop file paths.

directorytreemap ()

directorytreemap=(directorytreemap::map)

dotrace ()

dotrace=(dotrace::boolean)

error_lang ()

error_lang=(error_lang::knop_lang)

filename (*-type::string*, *-path::string =?*)

filename (*type::string*, *-path::string =?*)

filename (*type::string*, *path::string =?*)
Returns the full path to the specified type of precissing file for the current navigation.

Parameters: -type (required)

lib, act, cnt, cfg, actcfg

filenaming ()

filenaming=(filenaming::string)

fileroot ()

fileroot=(fileroot::string)

findnav (*path::array*, *navitems::array*)

getargs (*-index::integer*)

getargs (*index::integer =?*)
Path arguments = the leftover when we found a matching path, to be used for keyvalue for example.

Parameters:

- index (optional integer)

  Specifies which leftover path item to return, defaults to all path items as a string

getlocation (*-setpath::string*, *-refresh::boolean =?*)

getlocation (*setpath::string =?*, *refresh::boolean =?*)
Grabs path and actionpath from params or urlhandler, translates from url to path if needed. This must be called before using the nav object.

Parameters:

- setpath (optional)

  forces a new path

`getlocation_didrun``()`

`getlocation_didrun=(getlocation_didrun::boolean)`

`getnav``(`*-path*`)`

`getnav``(`*path =?*`)`
Return reference to the current navigation object map, or for the specified path.

`haschildren``(`*-navitem::map*`)`

`haschildren``(`*navitem::map*`)`
Returns true if nav object has children that are not all -hide.

`include``(`*-file::string*, *-path::string =?*`)`

`include``(`*file::string*, *path::string =?*`)`
Includes any of the files for the current path, fails silently if file does not exist.

Parameters: -file (required)

lib, act, cnt, cfg, actcfg or library, action, config, actionconfig, content, or any arbitrary filename

`insert``(`*-key::string*, *-label =?*, *-default =?*, *-url =?*, *-title =?*, *-id =?*, *-template =?*, *-children =?*, *-param =?*, *-class =?*, *-filename =?*, *-disabled::boolean =?*, *-after =?*, *-target =?*, *-data =?*, *-hide::boolean =?*`)`

`insert``(`*key::string*, *label =?*, *default =?*, *url =?*, *title =?*, *id =?*, *template =?*, *children =?*, *param =?*, *class =?*, *filename =?*, *disabled::boolean =?*, *after =?*, *target =?*, *data =?*, *hide::boolean =?*`)`
Inserts a nav item into the nav array

`label``(`*-path::string*`)`

`label``(`*path::string =?*`)`
Returns the name of the current (or specified) nav location

Parameters: -path (optional)

`library``(`*-file::string*, *-path =?*`)`

`library``(`*file::string*, *path =?*`)`
includes file just as ->include, but returns no output.

`libraryfile``()`
Shortcut to filename: lib.

`linkparams``(`*-navitem::map*`)`

`linkparams``(`*navitem::map*`)`
Returns an array for all parameters that should be sent along with nav links

`navitems``()`

`navitems=(navitems::array)`

`navmethod``()`

`navmethod=(navmethod::string)`

`onconvert``()`
Shortcut to renderhtml

`oncreate` (*-template::string =?*, *-class::string =?*, *-currentclass::string =?*, *-currentmarker::string =?*, *-default::string =?*, *-root::string =?*, *-fileroot::string =?*, *-navmethod::string =?*, *-filenaming::string =?*, *-trace::boolean =?*)

`oncreate` (*template::string =?*, *class::string =?*, *currentclass::string =?*, *currentmarker::string =?*, *default::string =?*, *root::string =?*, *fileroot::string =?*, *navmethod::string =?*, *filenaming::string =?*, *trace::boolean =?*)

Parameters:

- default (optional)

  Key of default navigation item

- root (optional)

  The root path for the site section that this nav object is used for

- fileroot (optional)

  The root for include files, to be able to use a different root for physical files than the logical root of the site. Defaults to the value of -root.

- navmethod (optional)

  path or param. Path for "URL designed" URLs, otherwise a -path parameter will be used for the navigation.

- filenaming (optional)

  prefix (default), suffix or extension, specifies how include files are named

- trace (optional flag)

  If specified debug_trace will be used. Defaults to disabled for performance reasons.

- template (optional)

  html template used to render the navigation menu

- class (optional)

  default class for all navigation links

- currentclass (optional)

  class added for the currently active link

- currentmarker (optional)

  character(s) show to the right link of current nav (typically &raquo;)

`path` (*-path::string*)

`path` (*path::string =?*)

Returns url or key path for the current or specified location

Parameters:

- path (optional)

`path=(path::string)`

`pathargs` ()

`pathargs=(pathargs::string)`

`patharray` ()

Returns current path as array.

patharray=(patharray::array)

pathmap()

pathmap=(pathmap::map)

renderbreadcrumb(*-delimiter::string =?*, *-home::boolean =?*, *-skipcurrent::boolean =?*, *-plain::boolean =?*)

renderbreadcrumb(*delimiter::string =?*, *home::boolean =?*, *skipcurrent::boolean =?*, *plain::boolean =?*)

Shows the current navigation as breadcrumb trail.

Parameters:

- delimiter (optional)

  Specifies the delimiter to use between nav levels, defaults to ” > ” if not specified

- home (optional flag)

  Show the default navigation item (i.e. “home”) first in the breadcrumb (unless already there).

renderhtml(*-items::array =?*, *-keyval::array =?*, *-flat::boolean =?*, *-toplevel::boolean =?*, *-xhtml::boolean =?*, *-patharray =?*, *-levelcount::integer =?*)

renderhtml(*items::array =?*, *keyval::array =?*, *flat::boolean =?*, *toplevel::boolean =?*, *xhtml::boolean =?*, *patharray =?*, *levelcount::integer =?*)

Render hierarchial nav structure.

Parameters:

- renderpath (optional)

  Only render the children of the specified path (and below)

- flat (optional flag)

  Only render one level

- expand (optional flag)

  Render the entire expanded nav tree and not just the currently active branch

- xhtml (optional)

  XHTML valid output

renderhtml_levels()

renderhtml_levels=(renderhtml_levels::integer)

rendernav(*-active::string =?*)

root()

root=(root::string)

sanitycheck()

Outputs the navigation object in a very basic form, just to see what it contains

scrubKeywords(*input*)

scrubKeywords(*input::trait_queriable*)

Pinched from Kyles inline definitions. Needed to have keywords, like -path act like regular pairs

serializationElements()

Called when a knop_user object is stored in a session

setformat (*-template::string =?*, *-class::string =?*, *-currentclass::string =?*, *-currentmarker::string =?*)

setformat (*template::string =?*, *class::string =?*, *currentclass::string =?*, *currentmarker::string =?*)
Sets html template for the nav object, use #items# #item# #/items# or more elaborate #items# #link##label##current##/link##children# #/items# as placeholders.

Parameters:

- template (optional string)

  Html template, defaults to <ul>#items#<li>#item#</li>#/items#</ul>

- class (optional string)

  Css class name that will be used for every navigation link

- currentclass (optional string)

  Css class name that will be added to the currently active navigation link (defaults to crnt)

- currentmarker (optional string)

  String that will be appended to menu text of currently active navigation link

setlocation (*-path::string*)

setlocation (*path::string*)
Sets the current location to a specific nav path or url

template ()

template=(template::string)

url (*-path::string =?*, *-params =?*, *-urlargs::string =?*, *-getargs::boolean =?*, *-except =?*, *-topself::knop_nav =?*, *-autoparams::boolean =?*, *...*)

url (*path::string =?*, *params =?*, *urlargs::string =?*, *getargs::boolean =?*, *except =?*, *topself::knop_nav =?*, *autoparams::boolean =?*, *...*)
Returns full url for current path or specified path. Path parameters can be provided and overridden by passing them to this tag.

Parameters:n

- path (optional)
- params (optional)

  Pair array to be used in url instead of plain parameters sent to this tag

- urlargs (optional)

  Raw string with url parameters to append at end of url and -params

- getargs (optional flag)

  Add the getargs (leftover path parts) to the url

- except (optional)

  Array of parameter names to exclude (or single parameter name as string)

- topself (optional nav)

  Internal, needed to call url from renderhtml when rendering sublevels

-autoparams (optional flag)

Enables the automatic passing of action_params that begin with “-“

urlmap()

urlmap=(urlmap::map)

urlparams()

urlparams=(urlparams::array)

version()

version=(version)

CHAPTER

EIGHT

KNOP_USER

class `knop_user`

Purpose:

- Maintain user identity and authentication
- Handle database record locking more intelligently, also to be able to release all unused locks for a user
- Authenticating user login
- Restricting access to data
- Displaying specific navigation options depending on type of user

lets add some date handling in there too like time of last login and probably the IP that the user logged in from.

Some options to handle what happens when a user logs in again whilst already logged in. ie one could:

•disallow second login (with a message explaining why)

•automatically log the first session out (with a message indicating what happened)

•send a message to first session: "Your username is attempting to log in again, do you wish to close this session, or deny the second login attempt?"

•allow multiple logins (from the same IP address)

•allow multiple logins from any IP address

All of these could be useful options, depending of the type of app.

And different types of user (ie normal, admin) could have different types of treatment.

Handling for failed login attempts:

•Option to set how many tries can be attempted;

•Option to lock users out permanently after x failed attempts?

•Logging (to database) of failed logins / successful logins

Password recovery system (ie emailing a time sensitive link to re-set password) By "password recovery" I'm not thinking "email my password" (hashed passwords can't be emailed...) but rather to email a short lived link that gives the user an opportunity to change his password. How is this different from "password reset"? Yes, that is an accurate description of what I had in mind, except for the bit about emailing a short-lived link. Instead I imagined having the user reset their password 100% on the web site through the use of "Security Questions", much like banks employ.

I like the idea of more info attached to the user. Like login attempts, locking a user temporarily after too many failed attempts etc.

The setup is more or less that I have users and groups.

I'm thinking that Knop shouldn't do any session handling by itself, but the knop_user variable would be stored in the app's session as any other variable. Knop should stay as lean as possible...

Other things to handle:

- Prevent session sidejacking by storing and comparing the user's ip and other identifying properties.
- Provide safe password handling with strong one-way salted encryption.

consider having a separate table for auditing all user actions, including logging in, logging out, the basic CRUD actions, searches

The object have to handle situations where no user is logged in. A guest can still have rights to some actions. Modules that can be viewed. Forms that could be sent in etc. That the added functions don't slow down the processing. We already have a lot of time consuming overhead in Knop.

Features:

1. Authentication and credentials
    - Handle the authentication process
    - Keep track of if a user is properly logged in
    - Optionally keep track of multiple logins to same account
    - Prevent sidejacking
    - Optionally handle encrypted/hashed passwords (with salt)
    - Prevent brute force attacks (delay between attempts etc)
    - Handle general information about the user
    - Provide accessors for user data
2. Permissions and access control
    - Keep track of what actions a user is allowed to perform (the "verbs")
    - Tie into knop_nav to be able to filter out locations based on permissions
3. Record locks
    - Handle clearing of record locks from knop_database
4. Audit trail/logging
    - Optionally log login/logout actions
    - Provide hooks to be able to log other user actions

Future additions:

- Keep track of what objects and resources a user is allowed to act on (the "nouns")
- Provide filtering to use in database queries
- What groups a user belongs to
- Mechanism to update user information, password etc
- Handle password recovery

Permissions can be read, create, update, delete, or application specific (for example publish)

`_unknowntag`(...)

acceptDeserializedElement(*d::serialization_element*)
Called when a knop_user object is retrieved from a session

addlock(*-dbname*)

addlock(*dbname*)
Called by database object, adds the name of a database object that has been locked by this user.

allowsidejacking()

allowsidejacking=(allowsidejacking::boolean)

auth()
Checks if user is authenticated, returns true/false

clearlocks()
Clears all database locks that has been set by this user.

client_fingerprint()

client_fingerprint=(client_fingerprint::string)

client_fingerprint_expression()
Returns an encrypted fingerprint based on client_ip and client_type.

cost()

cost=(cost::boolean)

costfield()

costfield=(costfield::string)

costsize()

costsize=(costsize::integer)

data()

data=(data::map)

dblocks()

dblocks=(dblocks::set)

description()

description=(description)

encrypt()

encrypt=(encrypt::boolean)

encrypt_cipher()

encrypt_cipher=(encrypt_cipher::string)

fields()

fields=(fields::array)

getdata(*field::string*)
Get field data from the data map

getpermission(*permission::string*)
Returns true if user has permission to perform the specified action, false otherwise

groups()

groups=(groups::array)

id_user()
Return the user id

id_user=(id_user)

keys()
Returns all keys for the stored user data.

logdatafield()

logdatafield=(logdatafield::string)

logdb()

logdb=(logdb::knop_database)

logeventfield()

logeventfield=(logeventfield::string)

login(*-username =?*, *-password =?*, *-searchparams::array =?*, *-force::string =?*)

login(*username =?*, *password =?*, *searchparams::array =?*, *force::string =?*)
Log in user. On successful login, all fields on the user record will be available by -> getdata.

Parameters:

- username (required) Optional if -force is specified
- password (required) Optional if -force is specified
- searchparams (optional) Extra search params array to use in combination with username and password
- force (optional) Supply a user id for a manually authenticated user if custom authentication logics is needed

loginattempt_count()

loginattempt_count=(loginattempt_count::integer)

loginattempt_date()

loginattempt_date=(loginattempt_date::date)

logobjectfield()

logobjectfield=(logobjectfield::string)

logout()
Logout the user

loguserfield()

loguserfield=(loguserfield::string)

oncreate(*-userdb::knop_database*, *-encrypt =?*, *-cost =?*, *-useridfield::string =?*, *-userfield::string =?*, *-passwordfield::string =?*, *-saltfield::string =?*, *-costfield::string =?*, *-logdb =?*, *-loguserfield::string =?*, *-logeventfield::string =?*, *-logobjectfield::string =?*, *-logdatafield::string =?*, *-singleuser::boolean =?*, *-allowsidejacking::boolean =?*)

oncreate(*userdb::knop_database*, *encrypt =?*, *cost =?*, *useridfield::string =?*, *userfield::string =?*, *passwordfield::string =?*, *saltfield::string =?*, *costfield::string =?*, *logdb =?*, *loguserfield::string =?*, *logeventfield::string =?*, *logobjectfield::string =?*, *logdatafield::string =?*, *singleuser::boolean =?*, *allowsidejacking::boolean =?*)

Parameters:

- encrypt (optional flag or string) Use encrypted passwords. If a value is specified then that cipher will be used instead of the default RIPEMD160. If -saltfield is specified then the value of that field will be used as salt.
- singleuser (optional flag) Multiple logins to the same account are prevented (not implemented)

`passwordfield``()`

`passwordfield=(passwordfield::string)`

`permissions``()`

`permissions=(permissions::map)`

`removedata` (*field::string*)
Remove field data from the data map

`saltfield``()`

`saltfield=(saltfield::string)`

`serializationElements``()`
Called when a knop_user object is stored in a session

`setdata` (*field*, *value =?*)
Set field data in the data map. Either -> setdata(-field='fieldname', -value='value') or -> setdata('fieldname'='value')

`setpermission` (*permission::string*, *value =?*)
Sets the user's permission to perform the specified action (true or false, or just the name of the permission

`singleuser``()`

`singleuser=(singleuser::boolean)`

`uniqueid``()`

`uniqueid=(uniqueid::string)`

`userdb``()`

`userdb=(userdb::knop_database)`

`userfield``()`

`userfield=(userfield::string)`

`useridfield``()`

`useridfield=(useridfield::string)`

`validlogin``()`

`validlogin=(validlogin::boolean)`

`version``()`

`version=(version)`

CHAPTER

NINE

KNOP_UTILS

class **knop_timer**

Utility type to provide a simple timer Usage:

```
Initialise  var(timer = knop_timer)
Read        $timer
Math        100 + $timer or $timer + 100
                        100 - $timer or $timer - 100
```

For other integer handling wrap it in integer first `integer($timer)`

+(rhs::integer)

-(rhs::integer)

asstring()

micros()

micros=(micros::boolean)

oncreate()

oncreate(*-micros::boolean*)

oncreate(*micros::boolean*)

resolution()

time()

timer()

timer=(timer::integer)

version()

version=(version::date)

knop_affected_count()

Adding a affected_count method pending a native implementation in Lasso 9 Used in sql updates, deletes etc returning number of rows affected by the change

knop_blowfish(*-string::string*, *-mode::string*)

knop_blowfish(*string::string*, *-mode::string*, *-key::string*)

knop_blowfish(*string::string*, *mode::string*, *key::string =?*)

knop_client_param(*param::string*, *-count::boolean =?*)
Returns the value of a client GET/POST parameter

Example usage:

```
knop_client_param('my');
knop_client_param('my', 2);
knop_client_param('my', 'get');
knop_client_param('my', 2, 'post');
knop_client_param('my', -count);
knop_client_param('my', 'get', -count);
```

Inspired by Bil Corrys lp_client_param Lasso 9 version by Jolle Carlestam

knop_client_param(*param::string*, *index::integer*, *method::string =?*, *-count::boolean =?*)

knop_client_param(*param::string*, *method::string*, *-count::boolean =?*)

knop_client_params(*-method::string =?*)

knop_client_params(*method::string =?*)
Returns a static array of GET/POST parameters passed from the client. An optional param "method" can direct it to return only post or get params

Example usage:

```
knop_client_params;
knop_client_params('post');
knop_client_params(-method = 'get');
```

Based on same code as action_params but without the inline sensing parts.

knop_crypthash(*string::string*, *-cost::integer =?*, *-saltlength::integer =?*, *-hash::string =?*, *-salt =?*, *-cipher::string =?*, *-map::boolean =?*)

knop_crypthash(*string::string*, *cost::integer =?*, *saltlength::integer =?*, *hash::string =?*, *salt =?*, *cipher::string =?*, *map::boolean =?*)

knop_encodesql_full(*text::string*)

knop_encrypt(*data*, *-salt =?*, *-cipher::string =?*)

knop_encrypt(*data*, *salt =?*, *cipher::string =?*)
Encrypts the input using digest encryption, optionally with salt.

knop_foundrows()

knop_IDcrypt(*value::integer*, *seed::string =?*)
Encrypts or Decrypts integer values

knop_IDcrypt(*value::string*, *seed::string =?*)

knop_math_dectohex(*base10::integer*)
Returns a base16 string given a base10 integer.

knop_math_hexToDec(*base16::string*)
Returns a base10 integer given a base16 string.

knop_normalize_slashes(*path::string*)

knop_response_filepath()
Safer than using Lasso 9 response_filepath when dealing wit hone file systems on Apache

knop_seed()

knop_stripbackticks(*input*)

knop_stripbackticks(*input::bytes*)

knop_stripbackticks(*input::string*)
Remove backticks (‘) from a string to make it safe for MySQL object names

knop_unique()
Original version Returns a very unique but still rather short random string. Can in most cases be replaced by the Lasso 9 version of lasso_unique since it’s safer than the pre 9 version.

knop_unique9(*-prefix::string =?*)

knop_unique9(*pre::string =?*)

CHAPTER

TEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*