Configuration Reference

Table of Contents

1. Interchange Table Editor.....1
1.1. Calling the table editor.....2
1.2. Attributes and attribute quoting.....3
1.3. Templating.....3
1.4. Metadata.....4
1.5. Extended settings.....8

1. Interchange Table Editor

Interchange has a powerful, highly−configurable table editor application implemented via its `[table−editor] ...` tag.

It is called in an Interchange page as simply as:

```
[table−editor cgi=1]
```

Given that call, it reads the passed CGI query information and builds a table editor for an interchange table.

Each field within the table editor is completely configurable for HTML widget type, label, help links, and more. These configurations can be saved in the `mv_metadata` database, or can be specified in the table−editor tag call itself.

The table editor is portable. It will work with any DBI/SQL database, with LDAP databases, and with Interchange DBM and plain−file databases.

Much of the Interchange administrative user interface (UI) is built around the table editor.

Its features include:

Complete range of widgets and data filters

Interchange has 18 different HTML widget types with data filters to condition the data.

Link fields from any table

Though the table editor uses one table as its base, fields from other tables can be brought in, and entire sets of records relationally linked to the base record can be edited within the table editor.

Tabbed display

Interchange automatically builds a tabbed interface from your fields specification.

"Wizard" mode

The table editor has a "wizard" mode that can collect information for accomplishing installation or setup tasks, with Next, Back, Cancel and Finish modes.

Templatable setup

You can completely control the way the table editor displays the widgets without interfering with its functionality.

Complex data structures

Interchange can build arbitrarily−deep data structures from form input. The collected data is serialized with the equivalent of Perl's Data::Dumper and stored in a single database field.

1.1. Calling the table editor

In its simplest form, table−editor is called in an ITL page with:

```
[table−editor table=products key=os28004][/table−editor]
```

That will edit the table products using its default configuration, for the SKU `os28004`.

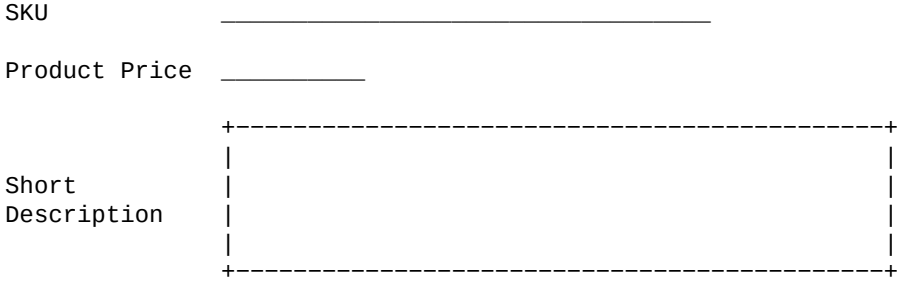
If no metadata is defined for the table, all fields are edited. To limit it with the tag call:

```
[table−editor
    table=products
    key=os28004
    fields="sku price description" ][/table−editor]
```

To specify that the field description should have a different widget type, height, and width, you can specify:

```
[table−editor
    table=products
    key=os28004
    fields="sku price description"
    widget.description=textarea
    width.description=50
    height.description=10
][/table−editor]
```

If you do this with the default `foundation` demo catalog, you will see:



Note that the labels are pulled from the `mv_metadata` definition −− any attributes not specified in the options do that. You can override each in turn −− to change SKU to "Part Number" you can do:

```
[table−editor
    table=products
    key=os28004
    fields="sku price description"

    label.sku="Part number"

    widget.description=textarea
    width.description=50
    height.description=10
][/table−editor]
```

To change the style of the label column, you can set the style information with:

```
[table−editor
```

```
      table=products
      key=os28004
      fields="sku price description"

      label_cell_style="font-weight: bold"

      label.sku="Part number"
      widget.description=textarea
      width.description=50
      height.description=10
  ][/table-editor]
```

This should **bold** the label text.

These are just a few small examples. There are more than 150 options for table editor which we will discuss below.

1.2. Attributes and attribute quoting

The [table-editor] is capable of accepting a large number of attributes. It uses standard ITL tag quoting, explained in the Interchange Tag Reference.

You can quote with single-quote (<'>), double-quote ("), backtick (`), or pipe (|). Material placed in backticks is run through a safe Perl interpreter. In fact, it is the equivalent of using the [calc] [/calc] tag pair except that contained ITL tags are not interpolated.

Pipe-quoting has the attribute of stripping trailing and leading whitespace; it is often convenient when specifying JavaScript (which uses both single and double quotes frequently) in the various `*_extra` parameters.

1.3. Templating

The [table-editor] is templated on several levels. In the most basic use, where you rely on it to build the table rows, there is the row_template option. By default, it is:

```
<td$opt->{label_cell_extra}>
  {BLABEL}{LABEL}{ELABEL}{META_STRING}
</td>
<td$opt->{data_cell_extra}>
  <table cellspacing=0 cellmargin=0 width="100%">
    <tr>
      <td$opt->{widget_cell_extra}>
        {WIDGET}
      </td>
      <td$opt->{help_cell_extra}>
         {TKEY}
         {HELP?}<i>{HELP}</i>{/HELP?}
         {HELP_URL?}<BR><A HREF="{HELP_URL}">help</A>{/HELP_URL?}
      </td>
    </tr>
  </table>
</td>
```

The values of $opt->{*_cell_extra} are constructed from the *_cell_class,

- _cell_width, *_cell_valign, *_cell_align, *_cell_style, and *_cell_extra options. You can watch the effect by trying different settings:

```
[table-editor
          table=products
          key=os28004
          label_cell_class=myclass
          label_cell_width=10%
          label_cell_valign=top
          label_cell_extra=|bgcolor="cyan"|
          ][/table-editor]
```

The values specified with {LABEL}, {WIDGET}, etc. are what are used to substitute the widget values constructed from the metadata. A perfectly functional template would be:

<td>{LABEL}</td><td>{WIDGET}</td>

That would show the label and widget without any help being shown (even if it is available) and using the default styles for at table data cell.

There is also the overall template, which is passed as the container text for [table-editor]. Something equivalent to the default can be achieved with:

{TOP_OF_FORM} {HIDDEN_FIELDS} <table> <tr> <td> </td> <td>{TOP_BUTTONS}</td> </tr> {:REST} <tr> <td> </td> <td>{BOTTOM_BUTTONS}</td> </tr> </table> {BOTTOM_OF_FORM}

There are two other templates -- the break_template and the combo_template. See "Templates".

1.4. Metadata

Interchange's foundation demonstration catalog and UI rely on a table named `mv_metadata`, which contains the definitions for table and field appearance. This table is supported with a table definition editor (pages/admin/db_metaconfig) and a field definition editor (pages/admin/meta_editor).

The `mv_metadata` table has the following fields:

code

The key for the table. Normally, it takes the form

```
table::column
```

where table is the table the field is contained in, and column is the table column name. It can also take the forms:

```
view::table::column::key
view::table::column
table::column::key
```

Each is checked in turn to see if it exists, then applied. If none of the above is found, then the field is displayed with a default widget (a text box with size 50).

type

The widget type. The following, at least, are supported:

Type	*Name*	*Description*
text	Text entry	The normal HTML text entry field.
textarea	Textarea	The normal HTML textarea entry for putting in multiple lines of data.
select	Select box	Also known as as a dropdown menu. Interchange has many ways to populate the options via automatic database lookup, and you can specify options to add or replace a lookup.
yesno	Yes/No (Yes=1)	A dropdown/select looking for a Yes (1) or No (0) answer.
noyes	No/Yes (No=1)	A dropdown/select looking for a Yes (0) or No (1) answer.
yesno radio	Yes/No (radio)	Same as the yesno widget except implmented with a radio box.
noyes radio	No/Yes (radio)	Same as the noyes widget except implmented with a radio box.
multiple	Multiple Select	A dropdown/select with SIZE greater than 1.
combo	Combo Select	A dropdown/select with a preceding text entry field that can add a new entry. Needs the `nullselect` filter; usually combined with a lookup.
reverse_combo	Reverse Combo	A dropdown/select with a following text entry field that can add a new entry. Needs the `last_non_null` filter; usually combined with a lookup.
move_combo	Combo move	A dropdown select that sends clicked items to a textarea.
display	Text of option	Displays the label (only) for a select/radio choice.
hidden_text	Hidden (show text)	Shows the value of a field and includes a hidden field to put the value in the form. Usually used when you want to display a key for a record but not give the opportunity to change it (and create a new record).
radio	Radio box	Select one of many options with a check box. Usually can be used instead of a select; can be grouped in matrices.
radio_nbsp	Radio (nbsp)	Select one of many options with a check box. Usually can be used instead of a select; can be grouped in matrices. This version puts no spaces in the outputted HTML, guaranteeing no wrap. (You can use the newer nowrap styles in CSS instead, often.)
checkbox	Checkbox	Select one or more options with a checkbox. Usually can be used instead of a multiple select.
check_nbsp	Checkbox (nbsp)	Select one or more options with a checkbox. Usually can be used instead of a multiple select. This version puts no spaces in the outputted HTML, guaranteeing no wrap. (You can use the newer nowrap styles in CSS instead, often.)
imagedir	Image listing	Shows a list of already existing images in a directory, with a link to a dialog to upload a new one.
imagehelper	Image	Combines input of an image name along with upload of the image.

	upload	
date	Date selector	Selects a date, with optional time. Used in combination with the `date_change` filter.
value	Value	Simply shows the value of the field, with no widget to set it.
option_format	Option formatter	
show	Show all options	Shows all options for a select/radio/checbox type input, without a widget to set the value.
uploadhelper	File Uppload	Puts the contents of a file upload in the named variable. Can be used as filter.widget="upload"

These widgets are implemented with the Vend::Form module, and are discussed in more detail later in this document.

width

The width of the widget. Meaningful in some way for most types.

height

In the field metadata, it is meaningful for textarea, multiple select types (including the combo widgets), and for groups of radio and checkboxes.

In the table metadata context, defines the number of rows that will be shown on the record select page before a "more" list will be built.

field

The fields for an options lookup query if more than one. Default is none −− the field in "lookup" is used.

db

The table to do the lookup query in. Default is the same table as the column is in.

name

The name of the generated HTML form element. Default is the same name as the column for the widget.

outboard

Catchall field used in several ways by different widgets. Normally used to specify a foreign key, it can also contain a directory name or other information needed for a widget.

options

Hard−coded options for the select, checkbox, and radio box widget types.

attribute

Not normally used for table editor. Used in Interchange in the context of an [item−list ...] for generating option names.

label

The label associated with the field for display in the table editor. In the below example, "Foo" is the label:

```
Foo: <input type=text value="bar" size=30>
```

help

Inline help to be displayed in the table editor.

lookup

A field name to look up options in. Normally this would be the same field as the widget, to generate a list of unique values. The equivalent to the query:

```
SELECT DISTINCT foo FROM table ORDER BY foo
```

is done.

filter

A filter which should be applied to the data coming from a widget before saving in the database. A few examples of the dozens of standard filters are:

nullselect
Select first non−null from HTML fields, used for combo box widget
digits_dot
Helps keep currency symbols and punctuation from polluting decimal values.
uc
Uppercases the data.
NN
Where NN is an integer. A number that limits length of input.

There are many more filters, and it is easy to specify custom filters. See Interchange's filters documentation.

help_url

A URL for extended help on a field.

pre_filter

For advanced use only. Specifies a filter that is run on the data before it is used to set the widget value. Not normally used.

lookup_exclude

A regular expression that can exclude certain values from a lookup list.

prepend

append

Valuse that are prepended and appended to the widget HTML, perhaps to call an external formatter or tool.

display_filter

Not normally used.

default

The default value that should be given to a column when it is of length zero.

extended

The repository for the serialized extended values set in metadata.

1.5. Extended settings

Interchange's meta editors use the table editor's serialization capability to set many more than just the fields mentioned above. There are over 150 different metadata settings, and it would be impossible to have each occupy a field in a table.

Many of these settings can be passed in a CGI query string if the cgi=1 option is specified.

across

The number of label−widget pairs which should be placed on each table row. The default is 1. Not passable by CGI.

action

The Interchange form action to use for the generated form. In normal mode, the default is "set". In wizard mode, the default is "return". Not passable by CGI.

action_click

An mv_click action that should be run on the Wizard next function. The default is "ui_override_next", which is usually a no−op on most systems. Not passable by CGI.

all_errors

Specifies that all form field entries should be checked for errors and their label fields set to the CONTRAST setting if an error is found.

all_opts

An ITL tag option that specifies that the table−editor options should be retrieved from one source.

If the option is a HASH reference, it will be used directly as a structure that will set all options.

If it is a scalar value, it will be used as a key to select the mv_metadata record which contains the options.

append

Active for every column in ui_data_fields. A value containing HTML which should be appended to the widget HTML.

auto_secure

Instructs Interchange to build a write enable only for the tables, columns, and keys that are specified in the table−editor call. Note that you can allow unfettered writes by setting the scratch variable `mv_data_enable`, but that it is rarely right to do so.

This prevents people from hacking together a duplicate of the [table−editor] form and writing columns or records they shouldn't.

This option is automatically disabled if the `cgi` option is enabled. Still, you should pay attention to what you are allowing users to write to your database.

back_text

The text that should be used in the Wizard **Back** button. Automatially translated for locale.

bottom_buttons

Indicates that buttons should always be only on the bottom. Normally, `[table−editor]` provides a top row of buttons if more than four rows are in the table.

break_cell_extra

Extra HTML attributes for the table cell in the standard break_row template. You might pass a valign, align, class, or other attribute:

```
break−cell−extra=|class="myclass"|
```

break_row_extra

Extra HTML attributes for the table row in the standard break_row template. You might pass a valign, align, class, or other attribute:

```
break−row−extra=|class="myclass"|
```

break_template

The HTML template that is used to present a break row. Default is:

```
<tr$opt−>{break_row_extra>
        <td colspan=$span $opt−>{break_cell_extra> </td>
</tr>
```

cancel_button_style

The HTML style for the cancel button.

cancel_text

The text placed in the "Cancel" button in both editor and wizard mode. Default is "Cancel".

cell_span

The number of cells that are in the span of a normal widget−label pair. The default is two, which is appropriate for the label in one table cell and the widget in the other. If you have a row_template like:

```
<td>{LABEL}</td><td> −−&gt; </td><td>{WIDGET}</td>
```

you would want a cell_span of three. For the row_template

```
<td align=left>
        <b>{LABEL}</b><br>
        {WIDGET}
</td>
```

you would want a cell_span of 1.

This allows the formatter to build the right number of cells for spacers and whole_row templates.

cgi

Signifies that some options may come from the URL calling the page where the `[table−editor]` resides.

This allows a simple:

```
[table−editor cgi=1][/table−editor]
```

to be active for any table and key that are called with a URL like:

```
http://your.catalog.url/cat/admin/flex_editor?mv_data_table=products?item_id=1
```

The `auto−secure` option is turned off if this option is set, for it would then be possible for people to call a table and set their own security.

check

A hash option that is active for every field. Defines a profile check that should be run on the field before the record will be allowed to be set (or the wizard allowed to go to the **Next** option.

This call

```
[table−editor
        table=products
        item_id="[cgi sku]"
        check.description=required
][/table−editor]
```

ensures that the description field will be non−blank and non−zero before the record is written.

Works in conjunction with the `process_filter` profile provided as a part of the UI. If using the table editor outside the province of the UI, you will need to make sure you get this profile included; it normally resides in lib/UI/profiles.

clear_image

In cases where the table−editor templates need a transparent image for display padding, can set the path where that image is. Default is `bg.gif`, which comes with the Interchange UI.

color_fail

The color to set failure messages to in HTML. Default Red.

color_success

The color to set success messages to in HTML. Default Green.

data_cell_class

data_cell_style

data_cell_valign

data_cell_width

data_cell_align

data_cell_extra

The settable parameters allowing change of the HTML appearance of the data cell in the standard `[table−editor]` presentation. The default is `data_cell_class` set to *cdata*.

The easiest thing to do to alter the look is define the CSS class for *cdata* how you want it. But you can individually set the width, style, and alignments; and you can attach scripting events or other CSS calls to the cell as well with `data_cell_extra`.

The following table−editor call:

```
[table−editor
        table=products
        item_id="[cgi sku]"
        data_cell_class=newclass
        data_cell_style="color:red"
        data_cell_extra=|onMouseOver="alert('move that mouse!'"|
        data_cell_valign=top
][/table−editor]
```

will result in a templated row of:

```
<td class=clabel>{LABEL}</td>
<td class="myclass"
```

```
            style="color:red"
            valign="top"
            onMouseOver="alert('move that mouse!')">
                    {WIDGET}{HELP?}{HELP} ... (rest of template)
    </td>
```

database

A hash attribute with a key associated with every field. Specifies the database table that will be used for a lookup (if any).

default

A hash attribute with a key associated with every field. Specifies the default value that will be used in the field if none is found in the table or otherwise available via the **default_ref**.

This is only active if the `defaults` flag is set.

default_ref

The hash reference that is used to set the defaults for the fields if there is no data in the table for that row. Default is $Values, the Interchange values hash.

You can set this in options with:

```
[table-editor default_ref=`$CGI`]
```

Note the backticks. This calls Perl, which returns the $CGI reference.

If you have previously collected some defaults in a scratch variable, you could use that with:

```
[table-editor default_ref=`$Scratch->{myhash}`]
```

Not settable in metadata, and it *must* be a hash reference or there will be a fatal error.

This is only active if the `defaults` flag is set.

defaults

Allows defaults to be set from the `default` hash above. When used in combination with `force_defaults=1` forces the passed defaults to override any data previously residing in the table record.

enctype

The encoding type for the form generated by [table-editor]. If the `file_upload` option is set, it defaults to `multipart/form-data`, otherwise uses the form default for the browser (which is normally `application/x-www-form-urlencoded`).

extra

A hash attribute with a key associated with every field. Specifies some extra HTML parameter(s) which will be attached to the widget for the field, based on the behavior of Vend::Form.

Normally used to pass an onChange or other scripting event.

field

A hash attribute with a key associated with every field. Specifies the fields to be used as a value−label pair for a lookup (if there is one). Works in conjunction with the `lookup` and `database` tags.

To build a list of products for selection by SKU in the `foo` widget, without having to use the SKU as the label, you can do:

```
[table−editor
      foo.widget=select
      foo.lookup=1
      foo.database=products
      foo.field="sku,description"
/]
```

Essentiallly the same as:

```
[table−editor
      foo.widget=select
      foo.lookup_query="select sku,description from products order by description"
/]
```

file_upload

Specifies that file upload should be enabled by changing the form encoding type. Forms using the `imagehelper` widget need this set.

filter

A hash attribute with a key associated with every field. Specifies a filter that will be applied to the widget result data before it is put in the database. To make sure that an integer value doesn't have extraneous whitespace that could cause an error, do:

```
[table−editor
      foo.widget=text_5
      foo.label="Seconds before we should become impatient and beep"
      foo.filter=digits
/]
```

Filters can be chained; they are the normal Interchange filters active in many situations.

Works in conjunction with the `process_filter` profile provided as a part of the UI. If using the table editor outside the province of the UI, you will need to make sure you get this profile included; it normally resides in lib/UI/profiles.

force_defaults

Causes the entries in the "defaults" hash to be used to set the initial value of fields, disabling the preference for data coming from an existing record in the database.

form_extra

Extra information (usually scripting event calls) for the form. For instance, if you are maintaining a series of event monitors in and want to make sure the user knows that some changes will be lost, you can set up a routine named `check_change()` in JavaScript. To check the change and ask for a confirmation before submission, you can do:

```
<script>
var changed = new Array;
function is_changed (element) {
      if(element != undefined)
              changed[changed.length] = element.name;
}

function check_change () {
      if(changed.length > 0)
        return
              confirm('the ' + changed.join(',') + ' elements were changed. Continue?');
      return true;
}
</script>

[table-editor
      foo.widget=text_50
      foo.label="Important stuff"
      foo.extra='onChange="is_changed(this)"
      form_extra='onSubmit="return check_change()"'
/]
```

You could also pass style information if that is ever appropriate. For form name, action, and target,

form_name

The name of the form. Do not include the `NAME=` portion, that is provided. Results in outputted HTML of:

```
      <FORM ACTION="$opt->{form_action}" METHOD=POST NAME="$opt->{form_name}">
```

get

Sets the method for the form to GET -- the default is POST.

height

A hash attribute with a key associated with every field. Sets the height of the widget if that is appropriate -- for example, it sets the ROWS parameter of a TEXTAREA, the SIZE parameter of a SELECT, etc. Also acts on the combination widgets according to the behavior of Vend::Form.

If you want to set the height of a cell, you should use the `*_extra` parameters.

help

A hash attribute with a key associated with every field. Specifies inline help that should be shown in the right-hand (data) portion of a widget row.

help_anchor

Sets the text that is used to anchor the `help_url` link. Default is `help`. Can also be used to set an image if you want to use that instead:

```
[table-editor
      table=products
      key=os28004
      help_anchor='<img src="question_mark.jpg" border=0>'
/]
```

help_cell_class

help_cell_style

help_cell_valign

help_cell_width

help_cell_align

help_cell_extra

Sets the help cell attributes. See **data_cell_extra**.

help_url

A URL which leads to extended help on a field. For example:

```
[table-editor
      table=products
      key=os28004
      widget.foo=text_50
      label.foo="Important stuff"
      help.foo="This is a complex field."
      help_anchor="Search for more help"
      help_url.foo="http://www.google.com/search?q=foo"
/]
```

hidden

Allows setting of extra hidden variables in a form. This is a hash attribute, with one key for every hidden variable you need to set.

```
[table-editor
      table=products
      key=os28004
      hidden.foo=bar
      hidden.buz=baz
/]
```

The above will include

```
      <input type="hidden" name="foo" value="bar">
      <input type="hidden" name="buz" value="baz">
```

href

Sets the action for the form. By default it is VendURL/ui, which checks for authorization for setting table records. If the `secure=1` parameter is set, SecureURL will be used instead. parameter is set, the SecureURL the default

include_before

include_form

inner_table_width

item_id

js_changed

keep_errors

label

label_cell_extra

label_cell_width

layer_panel_style

layer_tab_style

left_width

link_before

link_extra

link_fields

link_key

link_label

link_sort

link_table

link_template

link_view

lookup

lookup_query

mailto

message_label

meta

meta_anchor

meta_anchor_specific

meta_append

meta_class

meta_extra

meta_prepend

meta_style

method

mv_auto_export

mv_blob_field

mv_blob_label

mv_blob_nick

mv_blob_only

mv_blob_pointer

mv_blob_title

mv_cancelpage

Specifies the destination page if you hit the cancel−button.

mv_data_auto_number

mv_data_fields

mv_data_function

mv_data_table

mv_failpage

mv_nextpage

mv_prevpage

next_text

You can customize an alternative text for the OK/Next button.

Note! If your alternative text is long, and you get the text chopped, be sure to specify a wider button with the ok_button_style setting (eg. ok_button_style=|font−weight: bold; width: 80px; text−align: center|)

no_bottom

By default, form buttons are displayed at the top and bottom of the table. Specifying no_bottom=1 prevents the bottom set of buttons from being displayed.

```
no_bottom=1
```

no_meta

no_table_meta

no_top

By default, form buttons are displayed at the top and bottom of the table. Specifying no_top=1 prevents the top set of buttons from being displayed.

```
no_top=1
```

nodelete

noexport

nosave

notable

ok_button_style

options

orig_back_text

orig_cancel_text

outboard

override

panel_append

panel_height

Specifies the height of the data panels used in the tabbed diaplay. The default is 600.

```
panel_height=600
```

panel_id

Specifies the prefix used to identify the panels used in the tabbed display. The default is mvpan.

```
panel_id=mvpan
```

panel_prepend

panel_style

Specifies the CSS style applied to the data panels used in the tabbed display. The default is:

```
font-family: sans-serif;
font-size: smaller;
padding: 0;
border: 2px;
border-color:#999999;
border-style:outset;
```

panel_width

Specifiesthe width of the data panels used in the tabbed diaplay. The default is 800.

```
panel_width=800
```

passed

pre_filter

prepend

promiscuous

reload

reparse

restrict_allow

row_template

save_meta

secure

show_reset

simple_row

start_at

start_at_index

tab_bgcolor_template

Controls the iteration over the range of bgcolor attributes of the tabs in the tabbed display. If tab_bgcolor_template="#xx0000" the tabs will be set to "#ff0000" "#ee0000", etc. The default is "#xxxxxx" and the values are set to "#ffffff" "#eeeeee",etc. This option allows one to specify a range of colored tabs.

```
tab_bgcolor_template="#xx0000"
```

tab_height

Specifies the height of the tabs in the tabbed display. The default is 20px.

```
tab_height=20
```

tab_horiz_offset

Specifies the number of pixels that additional rows of tabs are shifted to the right. The default is 10px.

```
tab_horiz_offset=10
```

tab_style

Specifies the CSS style applied to the tabs in the tabbed display. The default is:

```
text-align:center;
font-family: sans-serif;
line-height:150%;
font-size: smaller;
border:2px;
border-color:#999999;
border-style:outset;
border-bottom-style:none;
```

tab_vert_offset

Specifies the number of pixels that additional rows of tabs are shifted upward. The default is 20px.

```
tab_vert_offset=20
```

tab_width

Specifies the width of the tabs in the tabbed display. The default is 120px.

```
tab_width=120
```

tabbed

This option specifies that the tabbed display mode is to be used. The tabbed=1 option displays sections as DHTML tabbed panels.

table

This option specifies the table as the source / destination of the form fields. The table is specified in the form: table=products.

table_width

td_extra

template

ui_blob_hidden

ui_blob_widget

ui_break_before

ui_break_before_label

ui_clone_id

ui_clone_tables

ui_data_fields

Specifies the database fields that a form collects. This is a quoted space delimited list of column names from the database table.

```
ui_data_fields="field1 field2 field3"
```

ui_data_fields_all

ui_data_key_name

ui_display_only

A quoted space delimited list of fields to be displayed but not edited.

```
ui_display_only="property_id"
```

ui_hide_key

Hides the key−field in the form. Remember that if you use the **fields** option, then you have to include the key−column in that list. Otherwise the form will fail to update the record.

ui_meta_specific

ui_meta_view

ui_new_item

ui_profile

ui_profile_success

ui_wizard_fields

Specifies the fields that a form collects in wizard mode.

```
ui_wizard_fields="wiz_field1 wiz_field2 wiz_field3"
```

widget

widget_cell_extra

widgets_only

width

wizard

Specifies that the form collects data as session variables as opposed to database fields. Data fields are specified using the **ui_wizard_fields** option.

wizard_cancel

wizard_next
