HelperForm

Table of contents

HelperForm

This helper is used to generate an edition form for an object of type ObjectModel. Example: editing the client's profile.

Form declaration

Fields inside [brackets] are optional as per the HTML standard.
Values between {curly braces} list the possible values for this field.

```
$this->fields_form = array(
  'legend' => array(
    'title' => $this->l('Edit carrier'),                    // This is the name of the fieldset, which can
contain many option fields
    'image' => '../img/admin/icon_to_display.gif'           // The icon must, if there is one, must be of the
size 16*16
  ),
  'input' => array(
    array(
      'type' => {'text', 'select', 'textarea', 'radio', 'checkbox', 'file', 'shop', 'asso_shop', 'free',
'color'},
      ['label'] => $this->l('Shipping method'),             // Theoretically optional, but in reality each
field has to have a label
      'name' => 'shipping_method',                          // The name of the object property from which we
get the value
      ['required'] => {true, false},                        // If true, PrestaShop will add a red star next to
the field
      ['desc'] => $this->l('Description displayed under the field'),
      ['hint'] => $this->l('Invalid characters:').' <>;=#{}' // This is displayed when the mouse hovers the
field.
      ['suffix'] => 'kg'                                    // This is displayed after the field (ie. to
indicate the unit of measure)
      ['options'] => array(                                 // This is only useful if type == select
        'query' => $array_of_rows,                          // $array_of_rows must contain an array of arrays,
inner arrays (rows) being mode of many fields
        'id' => 'id_carrier',                               // The key that will be used for each option
"value" attribute
      ),
      ['values'] => array(                                  // This is only useful if type == radio
        array(
          'id' => 'active_on',
          'value' => 1,
          'label' => $this->l('Enabled')
        ),
        array(
          'id' => 'active_off',
          'value' => 0,
          'label' => $this->l('Disabled')
        )
      ),
      [is_bool] => {true, false},                           // This is only useful if type == radio. It
display a "yes or no" choice.
      ['empty_message'] => $this->l('To be displayed when the field is empty'),
      ['lang'] => {true, false},                            // Is the field multilang?
     ),
    array(
      //another field
    ),
  ),
  'submit' => array(
    'title' => $this->l('   Save   '),                      // This is the button that saves the whole
fieldset.
    'class' => 'button'
  )
);
```

⚠️ If you want to use the "color" type, you can add the `"color mColorPickerInput"` classes

Basic declaration

Removing all the optional fields, this is how to build a basic HelperForm element:

```
$this->fields_form = array(
  'legend' => array(
    'title' => $this->l('Edit carrier'),
    'image' => '../img/admin/icon_to_display.gif'
  ),
  'input' => array(
    array(
      'type' => 'text',
      'name' => 'shipping_method',
     ),
  ),
  'submit' => array(
    'title' => $this->l('Save'),
    'class' => 'button'
  )
);
```

This specific code generates this HTML code (simplified here for readability reasons):

```
<form id="_form">
  <fieldset id="fieldset_main_conf">
    <legend>
      <img alt="Edit carrier" src="../img/admin/icon_to_display.gif">Edit carrier
    </legend>
    <div class="margin-form">
      <input type="text" class="" value="" id="shipping_method" name="shipping_method">
    </div>
    <div class="clear"></div>
    <div class="margin-form">
      <input type="submit" class="button" name="" value="Save" id="_form_submit_btn">
    </div>
  </fieldset>
</form>
```

Generating specific elements

The 'input' variable of the form declaration takes an array containing the content of your form. Using the various offered possibilities, you can build just about any type of form, and be assured that it will comply with PrestaShop's style and form processing.

You can use as many element arrays as necessary for your form, one after the other.

Text input

Here is how to generate a basic `<input>` element:

```
array(
  'type'     => 'text',                              // This is a regular <input> tag.
  'label'    => $this->l('Name'),                    // The <label> for this <input> tag.
  'name'     => 'name',                              // The content of the 'id' attribute of the <input> tag.
  'size'     => 50,                                  // The content of the 'size' attribute of the <input>
tag.
  'required' => true,                                // If set to true, this option must be set.
  'desc'     => $this->l('Please enter your name.')  // A help text, displayed right next to the <input> tag.
),
```

Selector

Here is how to generate a `<select>` element:

```
array(
  'type' => 'select',                             // This is a <select> tag.
  'label' => $this->l('Shipping method:'),        // The <label> for this <select> tag.
  'desc' => $this->l('Choose a shipping method'), // A help text, displayed right next to the <select> tag.
  'name' => 'shipping_method',                    // The content of the 'id' attribute of the <select> tag.
  'required' => true,                             // If set to true, this option must be set.
  'options' => array(
    'query' => $options,                          // $options contains the data itself.
    'id' => 'id_option',                          // The value of the 'id' key must be the same as the key for
'value' attribute of the <option> tag in each $options sub-array.
    'name' => 'name'                              // The value of the 'name' key must be the same as the key
for the text content of the <option> tag in each $options sub-array.
  )
),
```

The content of the selector is stored in the `$options` variable, which is an array of arrays. It must contain two keys: id and name.

`$options` can take this value:

```
$options = array(
  array(
    'id_option' => 1,                 // The value of the 'value' attribute of the <option> tag.
    'name' => 'Method 1'              // The value of the text content of the  <option> tag.
  ),
  array(
    'id_option' => 2,
    'name' => 'Method 2'
  ),
);
```

...but of course, you would be better off generating such an array of arrays yourself, from the data stored in PrestaShop. For instance, here is how to display a gender (social title) selector:

```
$options = array();
foreach (Gender::getGenders((int)Context::getContext()->language->id) as $gender)
{
  $options[] = array(
    "id" => (int)$gender->id,
    "name" => $gender->name
  );
}
```

Checkbox

Here is how to generate a `<input>` of type "checkbox":

```
array(
  'type'    => 'checkbox',                        // This is an <input type="checkbox"> tag.
  'label'   => $this->l('Options'),               // The <label> for this <input> tag.
  'desc'    => $this->l('Choose options.'),       // A help text, displayed right next to the <input> tag.
  'name'    => 'options',                         // The content of the 'id' attribute of the <input> tag.
  'values'  => array(
    'query' => $options,                          // $options contains the data itself.
    'id'    => 'id_option',                       // The value of the 'id' key must be the same as the key for
'value' attribute of the <option> tag in each $options sub-array.
    'name'  => 'name'                             // The value of the 'name' key must be the same as the key
for the text content of the <option> tag in each $options sub-array.
  ),
),
```

Just as for a selector input, check boxes take an array of arrays as the value of `$options`.

Radio button

Here is how to generate a `<input>` of type "radio":

```
array(
  'type'      => 'radio',                                 // This is an <input type="checkbox"> tag.
  'label'     => $this->l('Enable this option'),          // The <label> for this <input> tag.
  'desc'      => $this->l('Are you a customer too?'),     // A help text, displayed right next to the <input> tag.
  'name'      => 'active',                                // The content of the 'id' attribute of the <input> tag.
  'required'  => true,                                    // If set to true, this option must be set.
  'class'     => 't',                                     // The content of the 'class' attribute of the <label>
tag for the <input> tag.
  'is_bool'   => true,                                    // If set to true, this means you want to display a yes
/no or true/false option.
                                                          // The CSS styling will therefore use green mark for
the option value '1', and a red mark for value '2'.
                                                          // If set to false, this means there can be more than
two radio buttons,
                                                          // and the option label text will be displayed instead
of marks.
  'values'    => array(                                   // $values contains the data itself.
    array(
      'id'    => 'active_on',                             // The content of the 'id' attribute of the <input>
tag, and of the 'for' attribute for the <label> tag.
      'value' => 1,                                       // The content of the 'value' attribute of the <input>
tag.
      'label' => $this->l('Enabled')                      // The <label> for this radio button.
    ),
    array(
      'id'    => 'active_off',
      'value' => 0,
      'label' => $this->l('Disabled')
    )
  ),
),
```

Note that you have to use the `"t"` CSS class on your labels in order to have the proper styling (but you can redefine that class using the "class" variable).

Other HTML elements

The `type` variable of the element declaration makes it possible to generate just about any kind of `<input>` element: `text`, `select`, `textarea`, `radio`, `checkbox`, `file` and many others! See the list of available types here: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Input
You can also use some PrestaShop specific: `shop`, `asso_shop`, `free`, `color`. Try them out!