Bs_Form Package:
form building and handling

Version:	1.3 / 2003-03-03
Updated for:	bs-4.3
Author:	Andrej Arn
Copyright:	blueshoes.org

Intro

This manual shows you the possibilities and features of the Bs_Form package. It does not describe all possible options in great detail, to avoid help-text-dublication. Instead, it often points you to `Class->vars` and `Class->methods()` where you can read on.

The Bs_Form package is located in core/html/form. Whenever the manual points to examples, they can be found in the /examples subdirectory, and can be viewed online at
http://www.blueshoes.org/_bsCore/html/form/examples/example_file_name.

If you have not checked out the website of this package yet, do that now:
http://www.blueshoes.org/en/framework/html/form/
Also you should keep a browser open with the apidoc:
http://developer.blueshoes.org/phpdoc/

Setting up a basic form

Here we are building the simple.php example form step by step. You may look at the source code or run it. Or view it online at
http://www.blueshoes.org/_bsCore/html/form/examples/simple.php. For descriptions of the class vars please read the apidoc.

Creating the Bs_Form instance

Creating the instance and setting the names. Never forget the ampersand when creating new instances in PHP4!

```
$form =& new Bs_Form();
$form->internalName          = 'simpleForm';
$form->name                  = 'simpleForm';
```

Set some options:

```
$form->mustFieldsVisualMode = 'starRight';
$form->useAccessKeys        = TRUE;
$form->useJsFile            = TRUE;
$form->jumpToFirstError     = TRUE;
$form->buttons              = FALSE;
$form->language             = 'en';
$form->mode                 = 'add';
$form->onEnter              = 'tab';
```

The advancedStyles are used to highlight fields where inputs are missing or not accepted. See the css definition for `.formError` in the example source.

```
$form->advancedStyles       = array(
  'captionMust'      => '',
  'captionMustOkay'  => '',
  'captionMustWrong' => 'formError',
  'captionMay'       => '',
  'captionMayOkay'   => '',
  'captionMayWrong'  => 'formError',
  'fieldMust'        => '',
  'fieldMustOkay'    => '',
  'fieldMustWrong'   => '',
  'fieldMay'         => '',
  'fieldMayOkay'     => '',
  'fieldMayWrong'    => '',
);
```

Creating instances of Bs_FormElement

We don't create all elements here, just a few. Look at the example source for more.

At first we create a container. I use unset($var) before creating new containers because i like to reuse the same var name. I have to do it cause references on the object exist.

```
unset($container);
$container =& new Bs_FormContainer();
$container->name          = "containerNonFields";
$container->caption       = 'Non-Field Form Elements';
$container->mayToggle     = TRUE;
$form->elementContainer->addElement($container);
```

Then we create a text element (not a field). Here i unset the $element after assigning it to the $container so i can reuse the var. addElement takes the object by reference, that is why.

```
$element =& new Bs_FormText();
$element->name           = 'textBlock';
$element->text           = 'This is a text block. Any text
goes here.';
$container->addElement($element);
unset($element);
```

Then we create a text field.

```
$element =& new Bs_FormFieldText();
$element->name           = 'textField';
$element->caption        = 'text';
$element->editability    = 'always';
$element->must           = TRUE;
$container->addElement($element);
unset($element);
```

Doin' It

This happens in the HEAD of the html document:

```
$formRet = $form->doItYourself();
if (is_array($formRet)) {
      echo $form->includeOnceToHtml($formRet['include']);
      echo $form->onLoadCodeToHtml($formRet['onLoad']);
}
```

Bs_Form->doItYourself() does it all for us. It sees if the form is being displayed for the first time, or if it has been submitted. It validates the inputs, and decides if the form has to be displayed again, or if everything was ok and we can proceed (with storing data to a db, sending mails, whatever). Read `$form->doItYourself()`, also to know about the possible return values. You do not need to use doItYourself(), you may need the extra-flexibility by doing all this in your user code.

Later down this happens in the BODY of the html document. It spits out the information about input errors (if available), spits out the form (if needed), or if the form has been submitted successfully and everything was accepted then the string "Done!" along with a dump of $_POST are spitted out.

```
if (is_array($formRet)) {
      if (isSet($formRet['errors'])) echo $formRet['errors'];
      echo $formRet['form'];
} else {
      echo 'Done!<br><br>';
      echo 'dumping $_POST:<br>';
      dump($_POST);
}
```

What next? Have a look at `Bs_Form->getValuesArray()` to process the values.

Presentation

There are many options to change the way the forms look. Here are the ways and vars that can help you.

Layouts

By default all form elements are arranged in a standard way. Captions are on the left, form elements on the right. Elements are on top of each other. To produce forms with fully customized layout, templates can be used.

For an example have a look at:
http://www.blueshoes.org/_bsCore/html/form/examples/withTemplate1/

Set `Bs_Form->useTemplate` to `TRUE`, and specify `Bs_Form->templatePath`. Then put your templates in there. They have to be named according to the rules described in `Bs_FormTemplateParser->loadTemplate()`. If a template cannot be found, does not exist etc... it will fall back to standard output mode.

Note: templates can be used for forms AND for element containers (`Bs_FormContainer`).

The following tags can be used in your template code (see header doc of `Bs_FormTemplateParser` class):

bs_form:
All form elements then go between these tags. Mode can be the usual things, add, edit, delete etc.
Example: `<bs_form name="formName" mode="add"> ... </bs_form>`

bs_input
name= the internally used element name. has to be set.
type= one of 'element', 'caption', 'text', 'error', 'help'. default is 'element'.
elementlist= list of elements you want to have in the returned output. useful for radio buttons if you don't want all of them, or just one. give the keys of the elements you want.
examples:

```
<bs_input name="elementName" type="element"/>
<bs_input name="elementName" type="element"
elementlist="us,ca,de"/>
```

bs_formbuttons
example:
`<bs_formbuttons />`

bs_formerrors
properties: [caption]
example: `<bs_formerrors caption="Input Errors:"/>`

for an example of such an element layout look at the form examples.

elementLayout

When no template is used then the fields are spitted out in a standard way from the element container. An html table is used, and the caption is in a td on the left side while the element is in a td on the right side.
How this is done in detail is specified in the var `Bs_Form->elementLayouts`. Every element type (like a text field or a submit button) can have a different elementLayout. In Addition, every unique element can overwrite the layout using `Bs_FormElement->elementLayout`.

For an example have a look at:
http://www.blueshoes.org/_bsCore/html/form/examples/elementLayout.php

elementStringFormat

Using `Bs_FormElement->elementStringFormat` you can wrap the form element (without the caption, so only the __ELEMENT__ part of the elementLayout!) in some html code or text string.

For an example have a look at:
http://www.blueshoes.org/_bsCore/html/form/examples/elementLayout.php

hideCaption

The caption of form elements can be hidden (in different ways). That is the default behavior for some elements, namely the non-field elements. See `Bs_FormElement->hideCaption`. (some form elements overwrite it)

orderId

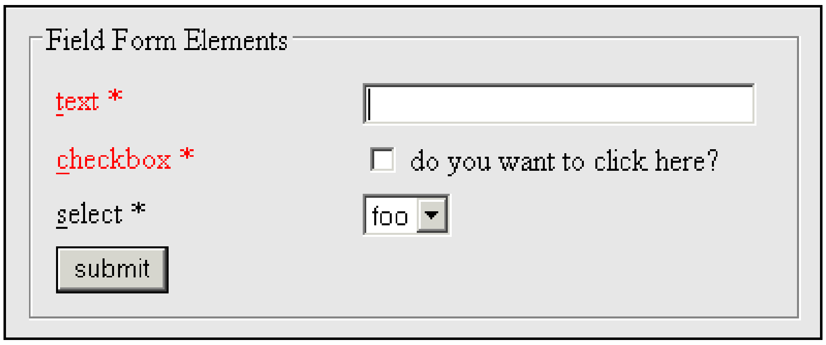
When no templates are used then, again, the elements are spitted out on top of each other. The order is the same as they were added to the form [container]. But using `Bs_FormElement->orderId` you can change it.

styles

Css styles can be used for form fields. See `Bs_FormField->styles`.

advancedStyles

If you want to use css styles based on the fact if a form field is a must field or not, and based on the fact if it has been filled in incorrectly (or was left out) then you can use advancedStyles. It is very easy for example to make fields appear in red color when there was an input error. See `Bs_Form->advancedStyles` and `Bs_FormField->advancedStyles`.



This feature has been used in the http://www.blueshoes.org/_bsCore/html/form/examples/simple.php example.

errorTableLayout

When errors occred then they are spitted out on top of the form in an html table. Very basic layouting can be used for that, see `Bs_Form->errorTableLayout`.

Elements

Check the website http://www.blueshoes.org/en/framework/html/form/ to see an overview of all form elements, and screenshots of how the elements look. There are not only the standard HTML elements. Every element has its own apidoc page too: http://developer.blueshoes.org/phpdoc/

Variables

There are some special vars to note:

```
Bs_Form->mode
Bs_Form->step
Bs_Form->level
```

Values, Input Manipulation and Validation

Values

Every form field can have up to 4 versions of the values:

```
Bs_FormField->valueDefault
Bs_FormField->valueReceived
Bs_FormField->valueInternal
Bs_FormField->valueDisplay
```

Input Manipulation

When the form has been submitted, and before the values get validated, there may be some input manipulation. Example: trim the values. See:

```
Bs_FormField->trim
Bs_FormField->remove
Bs_FormField->removeI
Bs_FormField->replace
Bs_FormField->replaceI
Bs_FormField->case
```

Based on `Bs_FormField->bsDataType` and `Bs_FormField->bsDataInfo` there may be a modification on the data type, for example a conversion to bool.

Example:
http://www.blueshoes.org/_bsCore/html/form/examples/inputManipulation.php

Input Validation

The user-supplied values can be validated against rules specified, for example the string minlength can be checked. Or if the data type of the field is set to 'email' then it should look like an email. See:

```
Bs_FormField->bsDataType          Bs_FormField->mustStartWith
Bs_FormField->bsDataInfo          Bs_FormField->notStartWith
Bs_FormField->enforce             Bs_FormField->mustEndWith
Bs_FormField->must                Bs_FormField->notEndWith
Bs_FormField->mustIf              Bs_FormField->mustContain
Bs_FormField->mustOneOf           Bs_FormField->notContain
Bs_FormField->mustOneOfIf         Bs_FormField->equalTo
Bs_FormField->onlyOneOf           Bs_FormField->notEqualTo
Bs_FormField->onlyIf              Bs_FormField->regularExpression
Bs_FormField->onlyOneOfIf         Bs_FormField->additionalCheck
Bs_FormField->minLength
Bs_FormField->maxLength
```

Every field where something is wrong will have `Bs_FormField->errorMessage` and `Bs_FormField->errorType` set. The default error messages are defined in text files in core/html/form/lang/ and are used according to the current language that is set in `Bs_Form->language`. But you can overwrite error messages using `Bs_FormField->defaultErrorMessage`.

Example:
http://www.blueshoes.org/_bsCore/html/form/examples/inputValidation.php

Triggers & Co.

Read about:

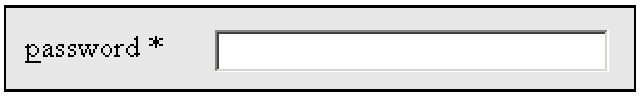
```
Bs_Form->postLoadTrigger()
Bs_FormField->codePostLoad
Bs_FormField->codePostReceive
Bs_FormField->codePostManipulate
```

Features

mustfieldsVisualMode

How must fields should be 'highlighted' in the browser. By default a star * is used. See `Bs_Form->mustFieldsVisualMode`.

Screenshot:



Access Keys

Access Keys are "short cuts" to jump to a field. You may know this form applications on your operating system. When a text has an underlined character like this or that then you can jump to it by hitting alt-character, for example alt-t or alt-h.

This is now possible with form fields in html too, at least in Internet Explorer. The html code for this looks like: `<input type="text" accesskey="x">`
When pushing alt-x the cursor goes into that field.

To make use of this feature in Bs_Form you have two options:

You can set `Bs_Form->useAccessKeys` to `TRUE`. Then the first character of the caption is used as access key for each form field automatically. Very handy.

For more control, you can define an access key for each form field itself, like this:

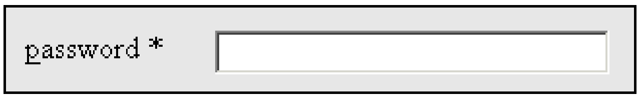
```
$myField->styles['accessKey'] = 'x';
$myField->styles['accessKey'] = array('en'=>'x', 'de'=>'y');
```

You can use option one, and overwrite the access key for some fields using option two.

The caption of the form field will then automatically look like caption with an underlined character.

When two fields have the same access key, the browser may act differently. As far as i know: IE (up to 6) jumps from one field to the next with the same access key if you repeat it. Mozilla jumps to the first or last field, and stays there even if you hit it again. No chance to move on to the others.

Screenshot:



jumpToFirstError

When the form has been submitted but something is wrong in a field, the cursor can be set into the first faulty field automatically using javascript.
See `Bs_Form->jumpToFirstError`.

onEnter

Browsers act differently when one hits enter in an input field. Usually the form gets submitted immediatly. This behavior may not be desired.
The options are:

- ignore the pressing of enter
- jump to next field (act like tab)
- submit form
- execute a userdefined javascript function

See `Bs_Form->onEnter`.

bs_after_formopen_tag and bs_before_formclose_tag

In the generated html code of the form output there are two special tags:
`<bs_after_formopen_tag/>` right after the <form> tag.
`<bs_before_formclose_tag/>` right before the </form> tag.
They can be replaced with anything you like, for example html code.

buttons

Buttons can be added as you like using the `Bs_FormFieldSubmit`, `Bs_FormFieldButton`, `Bs_FormFieldReset` and `Bs_FormFieldImage` classes.
Often forms need the same buttons. So using Bs_Form->buttons you can tell the form to use buttons in a standard way, based on the current language and mode it's in.
See `Bs_Form->buttons`.

Multilanguage support

The Bs_Form package is strong on language support. This allows to use exactly the same form for different languages of your web site.

Often variables can be assigned strings or arrays. Examples are Bs_FormElement->caption and Bs_FormField->valueDefault. Such vars usually include the sentence in the documentation header: "usually a string, can be a hash because it's language dependant."

That means: When specifying a string like:
`caption = 'Hello';` that string is used. When specifying an array like:
`caption = array('en'=>'Hello', 'de'=>'Hallo');` the current language is used (see `Bs_Form->language`).

There are always fallbacks. Examples: If 'en-uk' is not there but 'en' is, 'en' is used. If the currently used language is not defined in the array then the first array element is used.

Multilevel forms

Multilevel forms can be used to avoid showing huge forms at once. Using "next" and "back" buttons the user can move around. See the vars `Bs_Form->level` and `Bs_FormContainer->level`.

The supplied user data from previous levels is passed along in hidden fields until the final level is reached. This may slow down the process when using lots of data (blobs). On the other hand, it simplifies things, for example sessions that time out don't cause lost data. File uploads are not supported yet when using multilevel forms (except in the last level).

Note: You cannot use `Bs_Form->doItYourself(),` the handling has to be done a bit special. Use `Bs_Form->getAll()` etc.

Example: http://www.blueshoes.org/_bsCore/html/form/examples/multiLevel.php

Upload Forms

Using the feature-rich Bs_FormFieldFile class you can create forms with upload functionality. When uploading, the html form tag has to have a different encryption type: `<form enctype="multipart/form-data">`
The var `Bs_Form->encType` can be set if you like. If you don't then the package detects the needed enctype itself.

Example: http://www.blueshoes.org/_bsCore/html/form/examples/upload.php

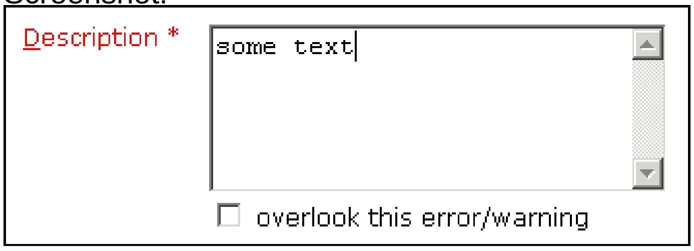
enforce

Validation rules may be restrictive. Mostly they really need to be, sometimes we just want to make sure the user thinks about what he's filling in.

There are moments when we want to warn a user about his input, but don't want to absolutely require him to change something.

For example the developer register form (that you have hopefully filled in :-) has a textarea field where we ask developers to write about the plans they have with blueshoes. The field has a minLength of 40 chars. And when a user fills in less, he'll see the form again with a warning about the minimum length. BUT: he can check the "ignore" checkbox and thus does not have to fill up the field with "asdf asdf asdf asdf". Altough most people do that. Be warned, people don't read. You are here, you do, congratulations. :-)

Screenshot:



This feature is demonstrated in:
http://www.blueshoes.org/_bsCore/html/form/examples/inputValidation.php

includeOnce, onLoadCode and head stuff.

When working with html one often faces this problem: You're in the process of writing body output, for example a web form, and realize you need to add some javascript. But by definition javascript really should go into the html head. But the head has already been created, or even worse, spitted out. That's why we use collecting of to-include javascript libraries, javascript code that has to execute in the onLoad event of the document, and other stuff that has to go into the head.

When using `Bs_Form->getAll()` or `Bs_Form->doItYourself()` (which itself uses `getAll()`) then we don't just receive the html code of the form, but also includeOnce etc stuff. That's returned in a hash, read the manual.

It is then up to you to place everything where it needs to be. Also see:

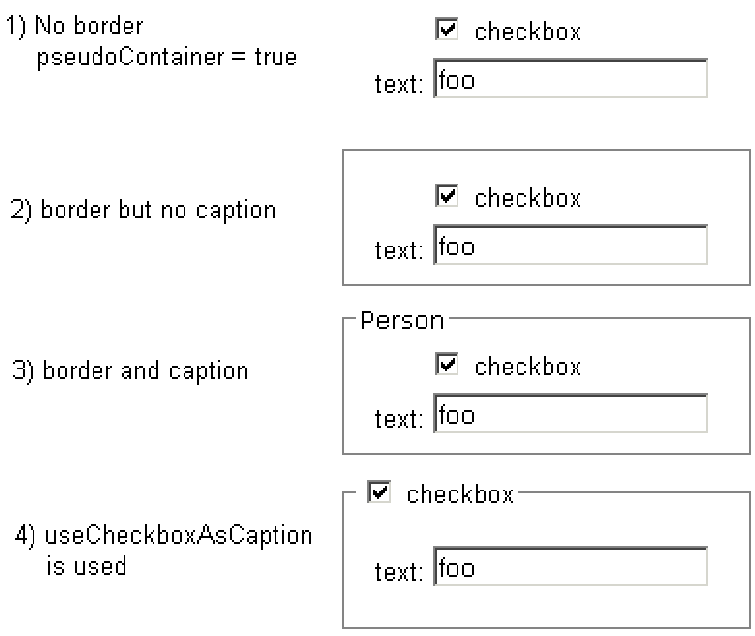
```
Bs_Form->addIncludeOnce
Bs_Form->getIncludeOnce
Bs_Form->includeOnceToHtml
Bs_Form->addOnLoadCode
Bs_Form->getOnLoadCode
Bs_Form->onLoadCodeToHtml
Bs_Form->addIntoHead
Bs_Form->getInHeadCode
```

Special Cases

useCheckboxAsCaption

An element container can have borders and a caption, as you can see in the examples 1-3 on the right side.

Using the property `Bs_FormContainer->useCheckboxAsCaption` a caption that is used in the container can be set as caption of the container directly, as example 4 shows. Very fancy. :-)



Example:
http://www.blueshoes.org/_bsCore/html/form/examples/useCheckboxAsCaption.php

explodable Fields

Very special and subject to change. No documentation at this time.