# Bs_FormItAble:
## *make your objects editable*

**Andrej Arn**
**Sam Blume**

blueshoes

BS_FORMITABLE (FIA) IS ABLE TO MAKE PHP OBJECTS EDITABLE USING WEB FORMS ON THE FLY.

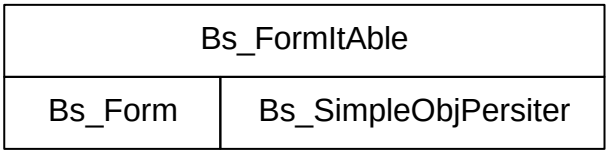
In order to understand this document, you should already have a basic knowledge of the object persister: Bs_SimpleObjPersiter; otherwise read the Bs_SimpleObjPersister.howTo.pdf.
FIA is not dependent on the object persister, but it definitely makes sense using the two packages together.
FIA is based on the Bs_Form package, so you may want to have a look at that too: http://www.blueshoes.org/en/framework/html/form/

**Intro:**
Say your boss would like to have an address book with an HTML form to fill in the data. The first step would be to crate a "person" class that represents the people. That is a 5 min task. But the following steps are much more cumbersome. These include:

A. Creating an HTML-form handling the input and adding error handling for the "dumb" users who do not fill in the form correctly.
B. Storing the data into the object and add/delete/modify the data in the DB.

This can easily cost you a day of boring work.

So BlueShoes came up with two packages to solve both points. For A) we wrote Bs_Form and for B) we wrote Bs_SimpleObjPersiter. Note that these packages are independent but complement each other when it comes to situations as the sample above. This is where FIA comes in. It combines the two packages. The result is much easier and shorter code to write and maintain. Isn't that formidable? :-)



**Source Location:**

| | |
|---|---|
| class: | core/storage/objectpersister/Bs_SimpleObjPersister.class.php |
| example: | core/storage/objectpersister/examples/simpleObjPersister.php |

Load the example file. We are going through it step by step here, but not every line is pasted into that document.

Now let us set up the class named 'PersonRecord'

Following steps have to be preformed:

1. Identify the class attributes that must be persisted and give them a default value.
2. Use the constructor to init the FIA-hint-hash.
3. Add the two call-back functions `bs_sop_getHints()` and `bs_fia_getHints($fiaAgent)`

```
class PersonRecord {

    var $ID           = 0;  // The ID. Used in the DB
    var $gender       = 0;  // field: 0 = unknown, 1 = male, 2 = female
    var $givenName    = ''; //
    var $familyName   = ''; //
    var $emailAddress = ''; //
    var $comments     = ''; //

    var $theFiaHints; // hints for the formitable class.
```

1) At first we need to identify which attributes are properties of the person, and thus should be persisted and editable:

- The ID field is the record ID in the database.
- The fields gender, givenName, familyName, emailAddress and comments belong to the person too.
- The fiaHints attributes will not be persistable nor editable.

2) Use the constructor to init the FIA-hint-hash.

```
/**
* Constructor
* - inits the FIA hint hash with default values
*/
function PersonRecord () {
  $this->initFiaHints();
}
```

3.a) Add the call-back function `bs_sop_getHints()`

Following the rules of the object persister (Bs_SimpleObjPersister) we must define a call-back function that returns the persit-hints-hash:

```
    /**
    * callback function for the object persister.
    */
    function bs_sop_getHints() {
        static $hint_hash = array (
            'primary' => array (
                'ID' => array('name'=>'id', 'type'=>'auto_increment'),
            ),
            'fields' => array (
                'gender'       => array('metaType'=>'integer'),
                'givenName'    => array('metaType'=>'string', 'size'=>40),
                'familyName'   => array('metaType'=>'string', 'size'=>40),
                'emailAddress' => array('metaType'=>'string', 'size'=>120),
                'comments'     => array('metaType'=>'blob',   'size'=>10000),
            ),
        );
        return $hint_hash;
    }
```

3.b) Add the callback function `bs_fia_getHints($fiaAgent)`

Also FIA requires us to write a callback function that must return the FIA-hint-hash. The FIA-hint-hash describes how the form will look like.
Note that we return $this->theFiaHints that was initialised in the constructor. If you wish to change the default behaviour of the form you're allowed to modify the `$theFiaHints` directly in your program.

```
    /**
    * callback function of Bs_FormItAble.
    * Return the hints used to build the form.
    */
    function bs_fia_getHints($fiaAgent) {
        return $this->theFiaHints;
    }
```

Everything that can be set to Bs_Form can be set in the 'props' array here. For example the 'language' is set to English, but can as well be set to German or ... whatever.

```
/**
* We keep the init of the FIA hints in this separate function.
*/
function initFiaHints() {
  // First define an access rights array (used below)
  $accessOmit = array(
        'guest'            => 'omit',
        'admin'            => 'normal',
  );

  $this->theFiaHints = array(
    'props' => array(
        'internalName'       => 'fiaForm',
        'name'               => 'fiaForm',
        'mode'               => 'add',
        'language'           => 'en',
        'user'               => 'guest', // set guest user as default
        'useAccessKeys'      => TRUE,
        'useTemplate'        => FALSE,
        'templatePath'       => $_SERVER['DOCUMENT_ROOT'] . './templates/',
        'jumpToFirstError'   => TRUE,
        'buttons'            => 'default',
        'advancedStyles'     => array(
            'captionMustWrong' => 'formError', // 'formError' is a CSS style
            'captionMayWrong'  => 'formError',
        ),
    ),
```

The groups are containers that are used later on in fields. We only use one container here that will have the caption 'Data'.

```
    'groups' => array(
        'grpData' => array(
                    'caption'   => 'Data',
                    'mayToggle' => FALSE,
        ),
    ),
```

The field definitions: The keys (ID and givenName) in the 'fields' array are the names of the class attributes. Then in the sub-arrays, the key 'name' is the form field name. The key 'group' defines to which container (defined above) this field belongs to. The fieldType is the
Bs_Form field implementation, all field types can be used of course.
All properties that can be set to the form field classes can be set in that array too. For example you could add 'size' => 30 to the givenName.

```
'fields' => array(
    'ID' => array(
        'name'            => 'ID',
        'group'           => 'grpData',
        'fieldType'       => 'Bs_FormFieldHidden',
        'must'            => FALSE,
        'editability'     => 'always',
    ),
    'givenName' => array(
        'name'            => 'givenName',
        'caption'         => array('en'=>'Given name', 'de'=>'Vorname'),
        'group'           => 'grpData',
        'fieldType'       => 'Bs_FormFieldText',
        'must'            => TRUE,
        'minLength'       => 2,
        'editability'     => 'always',
    ),

    #   :
    # snipp (There are more fields here; see the examples code)
    #   :

    'comments' => array(
        'name'            => 'comments',
        'caption'         => array('en'=>'Comments', 'de'=>'Bemerkungen'),
        'accessRights'    => $accessOmit, // accessRights setting
        'group'           => 'grpData',
        'fieldType'       => 'Bs_FormFieldTextarea',
        'must'            => FALSE,
        'editability'     => 'always',
        'cols'            => 30,
    ),
```

The 'comments' field above has an accessRights setting:
This means that for 'guest' users, the field is being omitted. For 'admin' users it is editable 'normal'.

NOTE: We have set the user as ‘guest’ by default in the `theFiaHints` above. So to see the comment field in the form you will have to modify the `theFiaHints` by setting

```
theFiaHints['props']['user'] = 'admin';
```

whenever an admin is viewing the form. This is a default Bs_Form feature, no hokus pokus.

Well that was all of the class. Let's go on with the code.

Make a DB connection, create an instance of the object persister, and pass db object to the persister.

```
$dsn = array ('name'=>'test', 'host'=>'localhost', 'port'=>'3306', 'socket'=>'',
                            'user'=>'root', 'pass'=>'',
                            'syntax'=>'mysql', 'type'=>'mysql'
);
if (isEx($dbAgent =& getDbObject($dsn))) {
    $dbAgent->stackDump('echo');
    die();
}
$objPersister = new Bs_SimpleObjPersister();
$objPersister->setDbObject($dbAgent);
```

Then create an instance of our new class. And if we got an ID in the request (get or post) then load the record with that ID.

```
$personRecord =& new PersonRecord();
if (!empty($_REQUEST['ID'])) {
    //let's load it
    $personRecord->ID = $_REQUEST['ID'];
    $objPersister->load($personRecord);
}
```

If we got an ID in the request, we know that we want to 'edit' a record, and not 'add' (default).
Also it is possible (in our example) that the user is set to 'admin' in the query string. Of course you would use a login and take this information from the session.

```
if (!empty($_REQUEST['ID'])) {
    $personRecord->theFiaHints['props']['mode'] = 'edit';
}
if (!empty($_REQUEST['user'])) {
    $personRecord->theFiaHints['props']['user'] = $_REQUEST['user'];
}
```

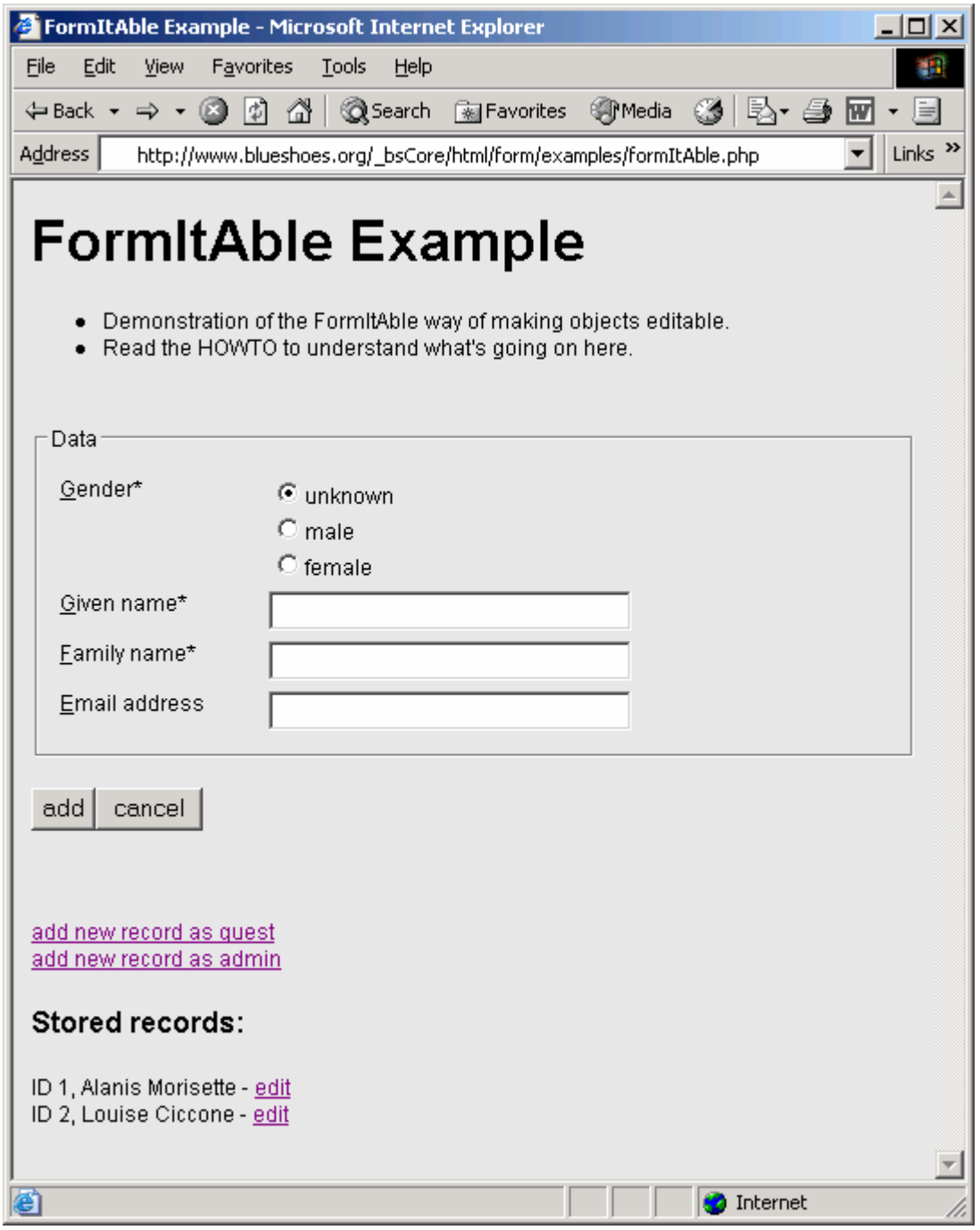
The next step is to create an instance of FIA, and then let it to the work. If it returns with bool TRUE then we know that we've got a successful submit. And all the variables in our object $personRecord will be updated from the user input. So we can use the persister to store the object. Otherwise, we got an array with information about the form that has to be displayed in the browser. The array includes keys like 'form', 'errors', 'include' and 'onLoad'. This is normal Bs_Form behavior.

```
$fia     =& new Bs_FormItAble();
$fiaOut = $fia->doItYourself($personRecord);
if ($fiaOut === TRUE) {
    //successful submit, let's store it.
    $objPersister->store($personRecord);
    $outBody .= "<b>Successfully stored as ID: " . $personRecord->ID . '</b>';
} else {
    $form =& new Bs_Form();
    $outHead .= $form->includeOnceToHtml($fiaOut['include']);
    $outHead .= $form->onLoadCodeToHtml($fiaOut['onLoad']);
    if (!empty($fiaOut['errors'])) $outBody .= $fiaOut['errors'];
    $outBody .= $fiaOut['form'];
}
```

The rest of the file is HTML code and a little bit of PHP, and we show a list of all persisted records to the user to make them editable:

```
$objs = $objPersister->loadAll('PersonRecord');
if (is_array($objs)) {
    $outBody .= "<h4>Stored records:</h4>";
    foreach ($objs as $obj) {
        $outBody .= "ID {$obj->ID}, {$obj->givenName} {$obj->familyName} - <a
href='{$_SERVER['PHP_SELF']}?ID={$obj->ID}&user=admin'>edit</a><br>\n";
    }
}
```

And this is how it looks in the browser.




Have you noticed that the 'comments' field is not here? Remember that the default user is set to 'guest', and for 'guest' we set the field to 'omit'. Edit an existing record, or add a new one as user 'admin' and the field will be there.

(For details on the Bs_SimpleObjPersister callback see the object persister HOWTO)