

ESIform v2.0 Documentation

Idea & created by Tino Ehrich

Visit my site: www.efides.com

May, 10th 2006 » © Tino Ehrich 2008

01. Introduction

[+] A [-] B [-] C

I designed ESIform to satisfy all needs for a "fast" formular creation as well as the creation of all the follow-ups (e.g. automatic email follow-up to the user).

Because of some clients who request really frequently new formulars I had to create new formulars which have to be automatically server-side validated and after this saving the data to an database (crm). So I created a simply form generator first. Important was that the design came from the clients. So it was necessary to create the formular-elements wherever you want - not just each field below the other. Also important was the error-messaging.

I think all together customizeability is the best word for this.

This worked well for the first time, but soon I had a request for a socket follow-up to an ssl-server. On the one hand the caught data should be saved in the database (crm) and then send on to this sslserver. Also the name of the data-fields and some data-values should be different. So it was time to create a new formular generator which will satisfy all this formular management actually really easy. ESIform was born!

What about other form-generators?

I was searching for other form-generators, but they didn't satisfy my needs. Either they just supported one field below each other, so you couldn't really customize your formular or they couldn't manage the automatical setting of required fields or they didn't support follow-ups etc. So it was more easy for me to write my own form-generator and bring in all my experiences.

Features

- **Full customizeability:** There are no design-limits
- **Easy array based formular creation:** You don't need to be a pro to set up your form
- **Supporting all fields:** text-fields, drop-downs, checkbox- and radio-buttons
- **Field-Activation:** activate non-required fields dynamicly
- **Easy field validation:** if necessary you can match the values with your defined regex-code
- **Supporting multiple customizable follow-ups:** e-mail, sockets, database, file or all together and how many as you want
- **Supporting fallback:** If there occurs an error by trying to follow-up your data, the system notice this and saves your data on your local-server - so, no data get lost!
- ... and much more - just find it out!

02. Structure

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

The structure consists out of 3 parts:

■ config (array)

This is one of the parts which is not necessary to create and validate your form. By this array you can define things like the error-message, the color of the error-messages, if there are follow-ups etc.

■ fields (array)

Here you will define all your form-fields. Each field will be defined by a new array which includes all the necessary and optional field-definitions.

■ follow-ups (array)

This is the other part which is not really necessary to create and validate your form. The follow-ups array contains all your follow-ups which should be called after a successful validation of your form. Your data can be send/saved by the following follow-ups-types:

- Mail
- Socket (Http-Request)
- MySQL (Database)
- File

After the call of your follow-ups, you will be redirected to a before defined url.

■ template (string-text / file)

Put here your form-template. Either by putting your complete template-code or by putting the path to your template. The template has to be in text-format (html).

Take a look at the structure

```
1. <?php
2.
3.     /* load class-file */
4.     require("esiform.class.php");
5.
6.     /* call class */
7.     new esiform(
8.
9.         "config"      => array(),
10.        "fields"       => array(),
11.        "follow-ups"  => array(),
12.        "template"    => "foo bar text",
13.
14.    );
15.
16. ?>
```

03. Config definitions

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Following you will find all the possible definitions and their explanation for this array:

- **lib-path:** Sometimes the directory of your form-definitions will be different to the directory where ESiform resides. Therefore it's necessary to set the webserver-relative-path to the directory where ESiform resides. ESiform tries to determine this path by itself.
Examples:
 - "lib-path" => "/www/esiform"
 - "lib-path" => "/www/your-path-to-the-esiform-package"

- **ajax-post:** If you define this value, then your post-action will be processed in the background and the result will be passed on to a pre-defined field-element such as an layer. The value consists out of an array. The array requires the ID of the mentioned field-element.
Examples:
 - "ajax-post" => array("update-element-id" => "your-element-id")

- **error-color:** Sets the error-color for your form. The value has to be in ASCII. The default color is #f00 and looks like this.
Examples:
 - "error-color" => "#f00" /* default */
 - "error-color" => "#0000ff"
 - or whatever ...

- **error-message:** Sets the error-message which replaces your placeholder "{error-message}" in your template. There is an default message which goes like this: "Please correct / fill the red marked fields!".
Examples:
 - "error-message" => "Please correct / fill the red marked fields!" /* default */
 - "error-message" => "Here goes your message ..."

- **follow-ups:** Define here if you wanna use follow-ups for your form. The default value is "true" because in genereal everybody wanna use follow-ups. If you dont wanna use follow-ups, define this value to "false" in your config definitions.
Examples:
 - "follow-ups" => "true" /* default */
 - "follow-ups" => "false"

- **method-type:** Define here the method type of your form. The default type is set to "post".
Examples:
 - "method-type" => "post" /* default */
 - "method-type" => "get"

- **required-sign:** Set here the sign which should mark a required field. The default sign is

set to `[*]`. This shows a required field like this: Example-Field-Lable *

Examples:

- `"required-sign" => "[*]" /* default */`
- `"required-sign" => "#"`
- `"required-sign" => "+"`
- or whatever ...

- **required-sign-pos:** Set here the position of the required-sign. The default position is "front". By this position, the required-sign will be set in front of the field-label. The other possible position is the "back"-position.

Examples:

- `"required-sign-pos" => "front" /* default */`
- `"required-sign-pos" => "back"`

- **return-type:** Decide by this key, if you want to return your form as a printed version or a non-printed version. The non-printed version will return the rendered template. The default value is set to "print".

Examples:

- `"return-type" => "print" /* default */`
- `"return-type" => "return"`

- **sexyness:** If you set this value to true, ESIform will include the [scriptaculous-Javascript-Class](#). This class is needed if you want a nice, sexy animation for fields which will activated dynamically. Probably I will use it also for other things - but so far that's all.

Examples:

- `"sexyness" => "false" /* default */`
- `"sexyness" => "true"`

- **xhtml:** Decide by this setting if your field should be xhtml-valid. If so, your field will for example look like this: `<input type="text" name="foo" value="" />`

Examples:

- `"xhtml" => "true" /* default */`
- `"xhtml" => "false"`

04. Template definition

[+] A [-] B [-] C

Here you define your template-code. You can do this by putting your template-code directly into the definition or by putting a the path to your template-file. In both cases you have to define each field by at least 2 of 3 placeholders. Everything has to be inside of the form-tags.

Here is an example:

```
1.
2. <form action="index.php#error" method="post">
3.   <p>
4.     {firstname:label}<br />{firstname:field}
5.     <small>{firstname:description}</small>
6.   </p>
7. </form>
8.
```

The placeholders will be automatically replaced by the defined fields. For the mentioned example there has to be defined field with the key "firstname". The placeholders are always ":label", ":field" and ":description". By the possibility to set the placeholders everywhere where you want to, it's perfect to fit it to your desired design. There shouldnt be any limit.

05. Field definitions

[+] A [-] B [-] C

Let's say that this part is the heart of ESIform. Here you have to define all your desired form-fields. Each field has to be defined by a new array.

Here is an simple example:

```
1. <?php
2.
3.     "fields"    => array(
4.
5.         "firstname"    => array(
6.             "type"      => "text",
7.             "label"     => "Firstname:",
8.             "required"  => true,
9.         )
10.     )
11. )
12.
13. ?>
```

Following I will explain you first of all the elements which can be defined for each field.

After the explanation of the "Elements" you can find some more detailed descriptions for each field-type - starting with [Text-Fields](#).

05.01. Elements

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Elements are needed to describe and build your field just as you need it. But not all elements can be used for each field. For example: an "options"-array doesn't make sense for a "text"-field. For the smaller elements (like "label", "description" ...) I didn't put any examples in those descriptions but you can find the examples in other parts from this documentation.

For more information click on the following links.

05.01.01. type label describ error-text required value

■ type:

Define here the type of your field:

- checkbox
- date
- email - like an text-field, but so ESIform will automatically parse the field for a syntactly correct e-mail-address
- hidden
- password
- radio
- select
- text
- textarea

■ label:

Specify here your label-text for your field. The "lable" will be automaticly set to the css-class "esiform-label". Define this class to format your labels just as you desire.

■ describ:

Sometimes you need an additional explanation for your field. Put it here. The "describ" will be automaticly set to the css-class "esiform-describ". Define this class to format your descriptions just as you desire.

■ error-text:

Define here your individual field-error-text. This text will be displayed as soon as an error for the certain field occurs. Also the text will be marked in the defined "error-color" (see config-definitions). The "error-text" will be automaticly set to the css-class "esiform-error-text". Define this class to format your error texts just as you desire.

■ required:

Set this value to "true" or "false" depends if you want this field to be filled.

■ value:

Set here the pre-defined value of an text-/textarea-field.

05.01.02. attributes

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Here you can define the attributes for your field. This will be realized by an array with a key - value definition.

Example of all valid attributes:

```
1. <?php
2.
3.     "attributes"    => array(
4.         "size"      => 5,
5.         "maxlength" => 20,
6.         "multiple"  => true,
7.         "disabled"  => true,
8.         "tabindex"  => 2,
9.     )
10.
11. ?>
```

Attributes like "value", "selected" or "checked" will be set seperatly.

05.01.03. styles

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Here you can format your field with CSS-Style-Sheets just as you wish to.

Example of some possible values:

```
1. <?php
2.
3.     "styles"    => array(
4.         "width"        => "100px",
5.         "height"       => "200px",
6.         "background-color" => "#ffc",
7.         "color"        => "#00f",
8.         "font-family"  => "arial",
9.     )
10.
11. ?>
```

05.01.04. events

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Sometimes it's necessary to bind your field to Javascript-Events. If so, do it by this array.

Example of some possible values:

```
1. <?php
2.
3.     "events"    => array(
4.         "onchange"    => array(
5.             "if(this.value == 'foo') alert('bar');",
6.             "alert('ready!!');",
7.         ),
8.         "onclick"    => array(
9.             "alert('click, clack!');",
10.        ),
11.    )
12.
13. ?>
```

As you may have noticed, you can split your script-commands by each array-entry. ESiform will join your lines before the form will be print out. I just thought that this helps you to get an better overview of your scripts. In addition to this, you can also define more than one event at once.

05.01.05. options

[+] A [-] B [-] C

This field is for fields with more than one option to choose - DropDown- and Check-boxes as well as Radio-Buttons. This array contains the following sub-keys / -arrays:

- **keys:** This key is an boolean. Set it to true to tell ESiform that you use a key-value pair for your values.
- **pattern:** This field is necessary to coordinate your build-up of Checkbox-/ and Radiobutton-Fields. You easily define your fields by two placeholders called {name} and {field}. This is kind of important for these kind of fields, because in this way you can decide how you wanna build you fields. I think I will give you some examples, so you see what I mean.

Pattern example Nr.1 for an checkbox-Field:

```

1. <?php
2.
3.     "type"      => "checkbox",
4.     "options"   => array(
5.         "pattern" => "{field}{name}&nbsp;",
6.         "values"  => array(
7.             "value 1",
8.             "value 2",
9.         ),
10.    )
11.
12. ?>
```

Would return your fields like this: value 1 value 2

Pattern example Nr.2 for an checkbox-Field:

```

1. <?php
2.
3.     "type"      => "checkbox",
4.     "options"   => array(
5.         "keys"   => true,
6.         "pattern" => '
7.             <span style="width:45px">{name}</span>
8.             &#187; {field}<br />
9.         ',
10.        "values"  => array(
11.            "1" => "Sport",
12.            "2" => "Movies",
13.        ),
14.    )
15.
16. ?>
```

Would return your fields like this:

Sport » (Code: <input type="checkbox" name="" value="1">)

Movies » (Code: <input type="checkbox" name="" value="2">)

By this you are really free to define your field-design just as you desire. Dont forget that this pattern-design will replace your ":field"-placeholder for the certain field.

- **values:** This is an array of all possible field-options.

Values examples for an DropDown-Field:

```
1. <?php
2.
3.     /* simple value set */
4.     "options" => array(
5.         "values" => array(
6.             "value 1",
7.             "value 2",
8.         ),
9.     )
10.
11.     This will result in such an DropDown-Field:
12.
13.     <select>
14.         <option value="value 1">value 1</option>
15.         <option value="value 2">value 2</option>
16.     </select>
17.
18. ?>
```

The other method is a key-value pair:

```
1. <?php
2.
3.     /* key-value pair set */
4.     "options" => array(
5.         "keys" => true,
6.         "values" => array(
7.             "111111111" => "value 1",
8.             "222222222" => "value 2",
9.         ),
10.     )
11.
12.     This will result in such an DropDown-Field:
13.
14.     <select>
15.         <option value="111111111">value 1</option>
16.         <option value="222222222">value 2</option>
17.     </select>
18.
19. ?>
```

You see the difference? For the first method the value and key pair are exactly the same. For the second method you can define both independent from each other. For the second method it's important to set the "key"-field to true. Else ESiform doesnt know which type you use.

- **select:** Define by this array your pre-selected / -checked values. Each array entry represents one value-key. If there is pre-selection, ESiform will automatically take the first value of the option-values.

Example:

```
1. <?php
2.
3.     "options"    => array(
4.         "values"    => array(
5.             "value 1",
6.             "value 2",
7.         ),
8.         "select"    => array(
9.             "value 1"
10.        )
11.    )
12.
13. ?>
```

05.01.06. valid

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

This array helps you with your field-validation. It has some nice helpers which will be explained as followed:

- **match:** Place here an REGEX-String which should match the entered value. If this results in an false-match, your field wont be validated successfull. An example for this would be an special formatted telephone-number.

Example:

```
1. <?php
2.
3.     /* special telephone-number match */
4.     "valid"     => array(
5.         "match"  => '1234\d-00\d-\d99'
6.     )
7.
8. ?>
```

- **replace:** Place here an REGEX-String which will be run automaticly over your entered value. Everything which matches positive to your REGEX-String, will be deleted out of your entered value.

Example:

```
1. <?php
2.
3.     /* replace all integer-values */
4.     "valid"     => array(
5.         "replace" => '\d'
6.     )
7.
8. ?>
```

- **choose-min:** Define here if you need at least "x"-selected fields until your field will be validated positive. This setting can be used for checkbox and DropDown-multiple-Fields.

Example:

```
1. <?php
2.
3.     "valid"     => array(
4.         "choose-min" => 2
5.     )
6.
7. ?>
```

- **choose-max:** Define here if the number of maximum selectable fields. If the you stay in the range of the defined number your field will be validated positive. This setting can be used for checkbox and DropDown-multiple-Fields.

Example:

```
1. <?php
2.
3.     "valid"     => array(
4.         "choose-max" => 2
5.     )
6.
7. ?>
```

- **equal-to**: Pass here an array of other field-ids which have to match exactly the same value as the entered value for the current field.

Example:

If you create an user-account mostly you have to enter your password twice to re-check that you entered the password correctly. Both fields have to match each other. For those kinds of situations I offered this validation-type.

```
1. <?php
2.
3.     "valid"     => array(
4.         "equal-to" => array("password-1")
5.     )
6.
7. ?>
```

05.01.07. activate

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

This array let you activate non-required fields, if your field was successfull validated. You can handle this by an unique value or in general, which means that as soon as your field was successfull validated, all fields in the activation-array will be activated.

Example for an DropDown-Field with a unique-value activation:

```
1. <?php
2.
3.     /* activate through an unique value */
4.     "address" => array(
5.         "type"      => "select",
6.         "label"     => "Address:",
7.         "error-text" => "Please choose:",
8.         "required"  => true,
9.         "options"   => array(
10.            "values" => array(
11.                "Mr",
12.                "Mrs",
13.            )
14.        ),
15.        "activate" => array(
16.            "if-value" => array(
17.                "Mr" => array("foo-field", "bar-field")
18.            )
19.        )
20.    )
21.
22. ?>
```

Example for an DropDown-Field with a general activation:

```
1. <?php
2.
3.     /* activate through an unique value */
4.     "address" => array(
5.         "type"      => "select",
6.         "label"     => "Address:",
7.         "required"  => true,
8.         "options"   => array(
9.            "values" => array(
10.                "Mr",
11.                "Mrs",
12.            )
13.        ),
14.        "activate" => array(
15.            "foo-field",
16.            "bar-field"
17.        )
18.    )
```

- 19.
20. ?>

Auto-hide / Auto-appear option: A feature to this method is, that the fields will be hidden before they get activated. But this works only, if you implement an special DOM-ID in your template - for each of your fields which has to be activated. If this special ID doesnt appear to be in the template, there wont be any auto-hides / auto-appears for your fields. As soon as your field gets activated, it will be appear immediatly.

Let's say your field which has to be activated is called "foo-field". To make use of the "auto-hide-option" enter your DOM-ID in your template like you can see it for the following example:

- 1.
2. `<div id="field-box-foo-field">`
3. `{foo-field:label}
{foo-field:field}`
4. `</div>`
- 5.

As you can see, we just added a short string `field-box-` for out special id. ESiform is trying to match such an ID and if this turns out successfull, ESiform will auto-hide or auto-appear your field. Just depends on the given action.

If you miss to set your ID, your activation-field will be visible all the time.

05.02. Text-Fields

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Following you will find an example with all possible settings of an text-field. This is just an example to bring you the theory more closely. Remember that not all settings are necessary.

```
1. <?php
2.
3.     "firstname" => array(
4.
5.         "type"      => "text",
6.         "label"     => "Firstname:",
7.         "describ"   => "
8.             <small>Put here your firstname</small>
9.         ",
10.        "attributes" => array(
11.            "maxlength" => 20
12.        ),
13.        "styles"     => array(
14.            "width"      => "300px",
15.            "background-color" => "#ffc",
16.        ),
17.        "events"     => array(
18.            "onblur"    => array(
19.                "alert(this.value)"
20.            )
21.        ),
22.        "required"   => true,
23.        "valid"      => array(
24.            "replace"   => '\\d',
25.            "match"     => '^John'
26.        ),
27.        "activate"   => array(
28.            "lastname", "city"
29.        )
30.    )
31. )
32.
33. ?>
```

Instead of using "text" for your "type"-setting, you can also use "email", "password" or "hidden". The last both field types should be known. The "email"-type is actually the same as an "text"-type field but it comes with the benefit, that ESIform will automaticly check by an regex-match, if the entered value is an syntactly correct e-mail-address. Only if this will return an positiv match, the field is successfully validated.

05.03. Textarea-Fields

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Following you will find an example with all possible settings of an textarea-field. This is just an example to bring you the theory more closely. Remember that not all settings are necessary.

```
1. <?php
2.
3.     "message" => array(
4.
5.         "type"      => "textarea",
6.         "label"     => "Message:",
7.         "describ"   => "
8.             <small>Put here your message</small>
9.         ",
10.        "attributes" => array(
11.            "tabindex" => 2
12.        ),
13.        "styles"     => array(
14.            "width"      => "300px",
15.            "height"    => "400px",
16.            "background-color" => "#ffc",
17.        ),
18.        "events"     => array(
19.            "onblur" => array(
20.                "alert(this.value)"
21.            )
22.        ),
23.        "required"   => true,
24.        "valid"      => array(
25.            "replace" => '\d',
26.        ),
27.        "activate"   => array(
28.            "more-message"
29.        )
30.    )
31. )
32.
33. ?>
```

05.04. Checkbox-/ Radiobutton-Fields

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Following you will find an example with all possible settings of an checkbox-/ radiobutton-field. This is just an example to bring you the theory more closely. Remember that not all settings are necessary.

ESForm sets for each field an id-value. For the checkbox- / radiobutton-fields this id-value is combined out of the **field-type**, **field-id** and **the field-value**. This is important if you want to access those fields by javascript calls. Pay attention that there are no white-spaces, special-chars etc. in your field-value. Ok, lets see how the ids looks like:

- In the following example the field-id is named "dislikes" and the type is a "checkbox"
- By looking at the field-values the generated ids would look like this:

1. checkbox-field-dislikes-Rain
2. checkbox-field-dislikes-Spiders
3. checkbox-field-dislikes-Snakes

Example for an Checkbox-Field:

For the following example the user can choose out of 3 dislikes. If he chooses the dislike "Snakes", another field called "zoo" will be activated (This field is not listed in this example).

```
1. <?php
2.
3.     "dislikes" => array(
4.
5.         "type"      => "checkbox",
6.         "label"     => "Dislikes:",
7.         "describ"   => "
8.             <small>Choose your dislikes!</small>
9.         ",
10.        "attributes" => array(
11.            "tabindex" => 1
12.        ),
13.        "styles"     => array(
14.            "background-color" => "#ffc",
15.        ),
16.        "events"     => array(
17.            "onclick" => array(
18.                "alert(this.value)"
19.            )
20.        ),
21.        "required"   => true,
22.        "options"    => array(
23.            "pattern" => "{field}&nbsp;{name}&nbsp;|",
24.            "values"  => array(
25.                "Rain",
26.                "Spiders",
27.                "Snakes",
28.            ),
```

```

29.         "select" => array(
30.             "Rain", "Snakes",
31.         )
32.     ),
33.     "valid"      => array(
34.         "choose-min" => 2,
35.         "choose-max" => 3,
36.     ),
37.     "activate"  => array(
38.         "if-value"   => array(
39.             "Snakes" => array("zoo")
40.         )
41.     )
42. )
43. )
44.
45. ?>

```

Example for an Radiobutton-Field:

```

1. <?php
2.
3.     "dislikes" => array(
4.
5.         "type"      => "radio",
6.         "label"     => "Dislikes:",
7.         "describ"   => "
8.             <small>Choose your dislikes!</small>
9.         ",
10.        "attributes" => array(
11.            "tabindex" => 1
12.        ),
13.        "styles"     => array(
14.            "background-color" => "#ffc",
15.        ),
16.        "events"     => array(
17.            "onclick" => array(
18.                "alert(this.value)"
19.            )
20.        ),
21.        "required"   => true,
22.        "options"    => array(
23.            "pattern" => "{field}&nbsp;{name}&nbsp;|",
24.            "values"  => array(
25.                "Rain",
26.                "Spiders",
27.                "Snakes",
28.            ),
29.            "select"  => array(
30.                "Spiders"
31.            )
32.        ),
33.        "activate"   => array(
34.            "if-value" => array(
35.                "Snakes" => array("zoo")

```

```
36.          )
37.          )
38.
39.      )
40.
41. ?>
```

05.05. Drop-Down-Fields

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Following you will find an example with all possible settings of an drop-down-field. This is just an example to bring you the theory more closely. Remember that not all settings are necessary.

```
1. <?php
2.
3.     "dislikes" => array(
4.
5.         "type"      => "select",
6.         "label"     => "Dislikes:",
7.         "describ"   => "
8.             <small>Choose your dislikes!</small>
9.         ",
10.        "attributes" => array(
11.            "multiple" => true,
12.            "size"     => 10
13.        ),
14.        "styles"     => array(
15.            "background-color" => "#ffc",
16.        ),
17.        "events"     => array(
18.            "onclick" => array(
19.                "alert(this.value)"
20.            )
21.        ),
22.        "required"   => true,
23.        "options"    => array(
24.            "keys"    => true,
25.            "values"  => array(
26.                "r" => "Rain",
27.                "b" => "Bad weather",
28.                "s" => "Storm",
29.                "t" => "Traffic jam",
30.                "e" => "Empty account",
31.            ),
32.            "select" => array(
33.                "r", "e",
34.            )
35.        ),
36.        "valid"      => array(
37.            "choose-min" => 2,
38.        ),
39.        "activate"   => array(
40.            "if-value" => array(
41.                "e" => array("money")
42.            )
43.        )
44.    )
45. )
46.
```

05.06. Date-Fields

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Following you will find an example with all possible settings of an date-field. This is just an example to bring you the theory more closely. Remember that not all settings are necessary.

```
1. <?php
2.
3.     "day-of-arrival" => array(
4.
5.         "type"      => "date",
6.         "label"     => "Day of arrival:",
7.         "required"  => true,
8.         "format"    => "%Y/%m/%d",
9.         "lang"      => "en"
10.    )
11.
12.
13. ?>
```

The following languages are available:

1. DA = danish
2. DE = german
3. EN = english /* default */
4. ES = spanish
5. FR = french
6. HU = hungarian
7. IT = italian
8. NL = dutch
9. PT = portuguese

06. Follow-Up definitions

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Follow-ups will send on or save your data. The following types are supported:

- Mail
- Socket (Http-Request)
- MySQL
- File

The definition of a follow-up consists out of the type + individual name and the actually definition of the individual follow-up which is done by an array.

You will also be able to manipulate and add data to your processed post-data - individual for each type you wanna use. It's also possible to add data which will be returned by an socket-follow-up or mysql-follow-up.

```
1. <?php
2.
3.     /* syntactical example */
4.     "type-individual_name" => array()
5.
6.     /* example */
7.     "mail-my-example"      => array()
8.
9. ?>
```

The individual name is necessary for the case that you need to define more than one follow-up of the same type because an array can't have a same key twice. Else the following follow-up would overwrite the one before.

What else? You can influence each follow-up by setting individual conditions like "equal" or "match". If the condition results in an true value, then your follow-up will be processed.

Finally you can set an redirect-url. This URL will be called automatically after all follow-ups has been successfull processed. For example: You can send the user to an "Thank you..." page.

Let's have a look on a quick follow-up-example:

Note that all follow-ups has to be defined within the "types"-Array. At the end you can set your redirect-url - if desired.

```
1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
5.         "socket-crm" => array(
6.             "host" => "www.xyz.com",
7.             "port" => 80,
8.             "path" => "/subscribe.php",
9.         ),
10.
11.         "mail-response" => array(
```

```
12.         "to"           => array( "{email}" ),
13.         "bcc"          => array( "office@you.com" ),
14.         "from"         => "office@you.com",
15.         "from-name"    => "Office",
16.         "subject"      => "Welcome!",
17.         "text"         => "
18.             Hi {firstname-format}!
19.             Your e-mail-address is: {email}
20.         ",
21.     )
22. ),
23.     "redirect" => "http://www.your-success-url.com",
24. ),
25. ?>
```

06.01. Types

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Here you will find the explanation of all possible follow-up-types. Just click the on the name of the type to get to know it.

06.01.01. Mail

[+] A [-] B [-] C

This follow-up let you send e-mails after a successful validation of your form. E-Mails can be send to infinite to-, cc- as well as bcc-addresses. Furthermore you can send e-mails as multipart (Html/Text) and with attachments. You can also add embedded attachments such as images.

The following list will show you the possible settings for a mail follow-up and their explanation:

- **to:** array with all to-addresses
 - You can use here the placeholders too
Example: `array("{email}")`
- **cc:** array with all cc-addresses
 - You can use here the placeholders too
Example: `array("{email}")`
- **bcc:** array with all bcc-addresses
 - You can use here the placeholders too
Example: `array("{email}")`
- **from:** string with the sender-address (placeholders are allowed)
- **from-name:** string with the sender-name (placeholders are allowed)
- **subject:** your e-mail-subject (placeholders are allowed)
- **text:** your e-mail text-version. Here you can work with placeholders to insert the entered form-data.
- **html:** your e-mail html-version. Here you can work with placeholders to insert the entered form-data.
- **attachments:** an array where you can define all your attachments. Each attachment has to be defined in an new array. This array contains the following settings:
 - **path:** path to your attachment
 - **name:** the name of your attachment which will be used in the e-mail
 - **type:** this have to be the MIME-Type of your attachment. Example: If you send an jpg-image your mime-type will be "image/jpeg". » [List of all mime-types](#)
- **embedded:** an array where you can define all your embedded-attachments. Embedded-attachments are for example images which are included in your html-version. As soon as you open your e-mail, your images will be displayed because they are part of your e-mail. Each attachment has to be defined in an new array. This array contains the following settings:
 - **path:** path to your attachment
 - **name:** the name of your attachment which will be used in the e-mail
 - **cid:** name of the placeholder for this attachment. The placeholder has to be in the html-version. In most cases this is probably an image. So you have to build it like this: ``
 - **type:** this have to be the MIME-Type of your attachment. Example: If you send an gif-image your mime-type will be "image/gif". » [List of all mime-types](#)
- **string-attachments:** a "string-attachment" is actually the same as the "attachment". The difference is, that for the "attachment"-type you set the path for your attachment so that ESiform can load the attachment from the given path. A "string-attachment" dont have a path because you pass the attachment (the data) directly as a string.

- **data:** put here your attachment text/data
 - **name:** the name of your attachment which will be used in the e-mail
 - **type:** this have to be the MIME-Type of your attachment. Example: If you send an html-file your mime-type will be "text/html". » [List of all mime-types](#)
- **add, replace & delete:** have a look at [the definition...](#)

Let's see some examples for this follow-up!

Here is an example with two attachments:

```

1. <?php
2.
3. "follow-ups"    => array(
4.     "types" => array(
5.         "mail-test"    => array(
6.             "to"        => array( "{email}", "{email2}" ),
7.             "bcc"       => array( "office@mycompany.com" ),
8.             "from"      => "office@mycompany.com",
9.             "from-name" => "Office",
10.            "subject"   => "Test email",
11.            "text"      => "Hi {firstname-format}!",
12.            "attachments" => array(
13.                array(
14.                    "path"    => "/www/your-image.jpg",
15.                    "name"    => "image.jpg",
16.                    "type"    => "image/jpeg",
17.                ),
18.                array(
19.                    "path"    => "/www/attachments/info.pdf",
20.                    "name"    => "info.pdf",
21.                    "type"    => "application/pdf",
22.                ),
23.            )
24.        )
25.    ),
26.    "redirect" => "http://www.your-success-url.com",
27. )
28.
29. ?>

```

An example with an normal attachment and an embedded-attachment:

```

1. <?php
2.
3. "follow-ups"    => array(
4.     "types" => array(
5.         "mail-test"    => array(
6.             "to"        => array(
7.                 "{email}",
8.                 $_POST["email2"]
9.             ),
10.            "bcc"       => array( "office@mycompany.com" ),
11.            "from"      => "office@mycompany.com",

```

```
12.     "from-name"    => "Office",
13.     "subject"     => "Hi {firstname-format}",
14.     "text"        => "Welcome to tech.news ...",
15.     "html"        => "
16.         <p>
17.             <strong>
18.                 Welcome to the
19.                 new tech.newsletter ...
20.             </strong>
21.         </p>
22.
23.         That's me:<br />
24.         
25.     ",
26.     "attachments" => array(
27.         array(
28.             "path"    => "/www/info.pdf",
29.             "name"    => "info.pdf",
30.             "type"    => "application/pdf",
31.         ),
32.     ),
33.     "embedded"    => array(
34.         array(
35.             "path"    => "/www/images/me.jpg",
36.             "name"    => "thats-me.jpg",
37.             "cid"     => "thats-me",
38.             "type"    => "image/jpeg",
39.         ),
40.     )
41. )
42. ),
43. "redirect" => "http://www.your-success-url.com",
44. )
45.
46. ?>
```

And last but not least a string-attachment. In this case we wanna add a html-page which we generated just before.

String-attachment example:

```
1. <?php
2.
3. /* my string attachment */
4. $my_string_attachment = '
5.     <html><body><h1>Hello World!</h1></body></html>
6. ';
7.
8. "follow-ups"    => array(
9.     "types"     => array(
10.         "mail-test"    => array(
11.             "to"        => array($_POST["email"], $_POST["email2"]),
12.             "bcc"       => array("office@mycompany.com"),
13.             "from"      => "office@mycompany.com",
14.             "from-name" => "Office",
```

```
15.     "subject"      => "Test email",
16.     "text"         => ":auto",
17.     "string-attachments" => array(
18.         array(
19.             "data"     => $my_string_attachment,
20.             "name"     => "html-info.pdf",
21.             "type"     => "text/html",
22.         ),
23.     ),
24. )
25. ),
26.     "redirect" => "http://www.your-success-url.com",
27. )
28.
29. ?>
```

Auto-generated texts

Maybe you noticed something special for the last example. There is a new placeholder for the text-version. When you put there the word ":auto" ESiform will replace this by a self generated list of all passed values. This works even when you dont define the text-version (when you leave the field blanket).

Example

When you have a form with the following fields

- address
- firstname
- lastname

then ESiform will generate a text-version which looks like this:

```
1.
2. {address-id}: {address-format}
3. {firstname-id}: {firstname-format}
4. {lastname-id}: {lastname-format}
5.
```

... and this will be finally rendered and send out like this:

```
1.
2. address: Mr.
3. firstname: Tino
4. lastname: Ehrich
5.
```

06.01.02. Socket

[+] A [-] B [-] C

This follow-up let you send on your data to another Server / Script after a successfull validation of your form. You can send your data also through an SSL-Connection.

The following list will show you the possible settings for a socket follow-up and their explanation:

- **host:** hostaddress of the receiving server. If this is an SSL-Connection please add the "ssl://" in front of the hostaddress.
- **port:** Server-Port of the receiving server. Normally this is 80. For an SSL-Connection it's mostly the port 443.
- **method:** Set here "GET" or "POST" depending of the kind of request you desire.
- **path:** Path to the receiving script (relative from the www-root)
- **add, replace & delete:** have a look at [the definition...](#)

Here is an simple example:

```
1. <?php
2.
3. "follow-ups"    => array(
4.     "types" => array(
5.         "socket-test"    => array(
6.             "host"      => "www.your-server.com",
7.             "port"     => 80,
8.             "method"   => "GET",
9.             "path"     => "/libs/test.php",
10.        )
11.    ),
12.    "redirect" => "http://www.your-success-url.com",
13. )
14.
15. ?>
```

Example for multiple sockets and a condition:

```
1. <?php
2.
3. "follow-ups"    => array(
4.     "types" => array(
5.         "socket-test-1"    => array(
6.             "conditions" => array(
7.                 "not-equal-to" => array(
8.                     "subscription" => "yes"
9.                 )
10.            ),
11.            "host"          => "www.your-server.com",
12.            "port"         => 80,
13.            "method"       => "GET",
14.            "path"         => "/libs/test.php",
15.        ),
16.
17.         "socket-test-2"    => array(
```

```
18.         "conditions" => array(  
19.             "equal-to" => array(  
20.                 "subscription" => "yes"  
21.             ),  
22.             "match" => array(  
23.                 "opinion" => "great"  
24.             )  
25.         ),  
26.         "host" => "ssl://www.my-server.com",  
27.         "port" => 443,  
28.         "method" => "POST",  
29.         "path" => "/libs/subscribe.php",  
30.     )  
31. ),  
32.     "redirect" => "http://www.your-success-url.com",  
33. )  
34.  
35. ?>
```

06.01.03. MySQL

[+] A [-] B [-] C

This follow-up let you store your data to a mysql-database after a successfull validation of your form.

The following list will show you the possible settings for a socket follow-up and their explanation:

- **host:** hostaddress of the mysql-database
- **user:** mysql-user-name
- **password:** password for the given mysql-user
- **name:** name of the to selecting database
- **query:** your database query (placeholders are allowed)
- **add, replace & delete:** have a look at [the definition...](#)

Here is an simple example:

```
1. <?php
2.
3. "follow-ups" => array(
4.   "types" => array(
5.     "mysql-test" => array(
6.       "host" => "www.your-server.com",
7.       "user" => "mysql",
8.       "password" => "mysql",
9.       "name" => "test_database",
10.      "query" => "
11.          insert into leads values('".$_POST["email"]."')
12.      "
13.    )
14.  ),
15.  "redirect" => "http://www.your-success-url.com",
16. )
17.
18. ?>
```

As you see, you can add the POST-Values by the Global-PHP-Variable "\$_POST" or "\$_GET" - depends on your chosen form-method. But you can also add the well known placeholder like: {email} or {email-format}. ESiform will set the reference value before the database-query will be processed.

```
1. <?php
2.
3. "follow-ups" => array(
4.   "types" => array(
5.     "mysql-test" => array(
6.       "host" => "www.your-server.com",
7.       "user" => "mysql",
8.       "password" => "mysql",
9.       "name" => "test_database",
10.      "query" => "
11.          insert into leads values('{name-format}','{email}')
```

```
14.     ),  
15.     "redirect" => "http://www.your-success-url.com",  
16. )  
17.  
18. ?>
```

06.01.04. File

[+] A [-] B [-] C

This follow-up let you store your data as file after a successfull validation of your form. The following list will show you the possible settings for a socket follow-up and their explanation:

- **path & file-name:** here you can determine the path & structure of your file. You can make use of date-placeholders which can be rendered by the php-function "strftime()". For example:
 - %Y = 4-digit year (i.e.: 2006)
 - %m = 2-digit month (i.e.: 06)
 - %d = 2-digit day (i.e.: 01)
 - %H = 2-digit hour (i.e.: 12)
 - %M = 2-digit minute (i.e.: 54)
 - %S = 2-digit seconds (i.e.: 10)
 - for more information have a look at the [php-manual](#)...

Furthermore you can use the known [placeholders](#) of your post-data as well as "%session" to use the unique session-id of the user as part of your filename. This should look something like this: "

Warning: session_start() [[function.session-start](#)]: Cannot send session cookie - headers already sent by (output started at C:\work\intranet\appz\public\esiform\v2.0\doc\index.php:109) in C:\work\intranet\appz\public\esiform\v2.0\doc\06-01-04-file.php on line 18

Warning: session_start() [[function.session-start](#)]: Cannot send session cache limiter - headers already sent (output started at C:\work\intranet\appz\public\esiform\v2.0\doc\index.php:109) in C:\work\intranet\appz\public\esiform\v2.0\doc\06-01-04-file.php on line 18
6fd5e9ddc6be0da2090fa17f9474cc1e"

Make sure that your path-settings are correct. ESIfirm checks only the dynamic parts of your path and file name. For the following example you have to make sure that "/www/subscriptions" exists.

- **add, replace & delete:** have a look at [the definition...](#)

Here is an simple example:

```
1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
5.         "file-test" => array(
6.             "path" => "/www/subscriptions/%Y/",
7.             "file-name" => "%m%d-{email}.txt",
8.         )
9.     ),
10. "redirect" => "http://www.your-success-url.com",
11. )
12.
13. ?>
```

The mentioned file would be saved in the following path:

[/www/subscriptions/2006/0508-email@user.com.txt](#)

06.02. Add, Replace, Delete

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

For each "follow-up"-type (mail, socket, mysql, file) it's possible to add, replace and delete data. This effect will only last for respective follow-up.

Well, there is nothing more to explain. Just have a look at the following examples.

Example: add data

```
1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
5.         "mail-response" => array(
6.             "add" => array(
7.                 "date" => date("d.m.Y"),
8.                 "time" => date("h:i"),
9.             ),
10.            "to" => array($_POST["email"]),
11.            "from" => "office@yourcompany.com",
12.            "subject" => "Welcome!",
13.            "text" => "Your e-mail-address is: {email}",
14.        )
15.    ),
16.    "redirect" => "http://www.your-success-url.com",
17. )
18.
19. ?>
```

Example: replace data

There are two ways to replace data. The first way is a value based replacement. Your data will be only replaced if the passed values matching each other. This is also possible by an regex-matching. Just add "regex:" in front of your matching string. See an example below.

The structure consists out of:

- field-id 1 => conditional-array
 - if field-value => replace value by
 - if field-value => replace value by
 - etc

- field-id 2 => conditional-array
 - if field-value => replace value by
 - etc

```
1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
```

```

5.     "mail-response"    => array(
6.         "replace"    => array(
7.             "if-value" => array(
8.                 "address" => array(
9.                     "Mr." => 1,
10.                    "Mrs." => 2
11.                ),
12.                "name"    => array(
13.                    "regex:tino" => "Quesito"
14.                )
15.            )
16.        ),
17.        "to"            => array($_POST["email"]),
18.        "from"          => "office@yourcompany.com",
19.        "subject"       => "Welcome!",
20.        "text"          => "Your e-mail-address is: {email}",
21.    )
22. ),
23. "redirect" => "http://www.your-success-url.com",
24. )
25.
26. ?>

```

The second way is a simple key replacement. You pass a key value which has to match the field-id. If so, the field will be replaced by the values you passed. The structure consists out of:

- field-id 1 => replace by (string or array)
- field-id 2 => replace by (string or array)

```

1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
5.         "mail-response"    => array(
6.             "replace"    => array(
7.                 "address" => array(1,2),
8.                 "date"    => date(YMD)
9.             ),
10.            "to"            => array($_POST["email"]),
11.            "from"          => "office@yourcompany.com",
12.            "subject"       => "Welcome!",
13.            "text"          => "Your e-mail-address is: {email}",
14.        )
15.    ),
16.    "redirect" => "http://www.your-success-url.com",
17. )
18.
19. ?>

```

Example: delete data

In this case, the placeholder "{email}" wont be replaced because there is no key-value for it - we just deleted it!

```
1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
5.         "mail-response" => array(
6.             "delete" => array(
7.                 "email"
8.             ),
9.             "to" => array($_POST["email"]),
10.            "from" => "office@yourcompany.com",
11.            "subject" => "Welcome!",
12.            "text" => "Your e-mail-address is: {email}",
13.        )
14.    ),
15.    "redirect" => "http://www.your-success-url.com",
16. )
17.
18. ?>
```

06.03. Conditions

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Conditions help you to call a follow-up only if a certain condition is valid. What kind of conditions are available? Take a look:

- **equal**: the passed value has to be equal to the passed field-value
- **match**: the passed regex-string has to match the passed field-value
- **in-array**: the passed value must exist in the passed field-array

For each type exists also an opposite type. To call the opposite-type just add an "not-" in front of the type name:

- **not-equal**
- **not-match**
- **not-in-array**

How is the structure build? First of all you have to define the type of the condition which is an array. For this array exist an array-key and an array-value. The array-key will be the field-id which value has to be checked against the array-value. If both values results in an positive result in respective to the condition-type, your follow-up will be processed.

You can define for each follow-up as much conditions as you want but only if all defined conditions results in an positive result, your follow-up will be processed.

Ok, let's have a look on an example:

If we have an follow-up which should send an e-mail to an user only if he entered an e-mail-address. There's no question for the reason of this because without an e-mail-address you cant send an e-mail to the user, right?!

Check if the user entered an e-mail-address

```
1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
5.         "mail-response" => array(
6.             "conditions" => array(
7.                 "not-equal-to" => array(
8.                     "email" => ""
9.                 )
10.            ),
11.            "to" => array($_POST["email"]),
12.            "bcc" => array("office@yourcompany.com"),
13.            "from" => "office@yourcompany.com",
14.            "from-name" => "Office",
15.            "subject" => "Welcome!",
16.            "text" => "
17.                Hi {firstname-format}!
18.                Your e-mail-address is: {email}
19.            ",
20.        )
21.    ),
22.    "redirect" => "http://www.your-success-url.com",
```

```
23. )  
24.  
25. ?>
```

In this example we just check if the field was filled. It would be also possible to check against a syntactically correct e-mail-address. But if we defined our "email"-field as an email-type-field, then ESiform did this validation already before. So we don't need it here.

Let's have a look on another example:

Check for birthdate on next day and the country

```
1. <?php  
2.  
3. "follow-ups" => array(  
4.     "types" => array(  
5.         "mail-response" => array(  
6.             "conditions" => array(  
7.                 "equal" => array(  
8.                     "birthdate" => date('y-m-d', mktime(0,0,0,1,1,2006))  
9.                 ),  
10.                "match" => array(  
11.                    "country" => '(Germany|Great Britain|France|Italy)'  
12.                )  
13.            ),  
14.            "to" => array($_POST["email"]),  
15.            "bcc" => array("office@yourcompany.com"),  
16.            "from" => "office@yourcompany.com",  
17.            "from-name" => "Office",  
18.            "subject" => "Welcome!",  
19.            "text" => "  
20.                Hi {firstname-format}!  
21.                Today is your birthday!  
22.  
23.                Happy birthday! :)  
24.            ",  
25.        )  
26.    ),  
27.    "redirect" => "http://www.your-success-url.com",  
28. )  
29.  
30. ?>
```

06.04. Placeholder

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Placeholders will be used for the text-versions from the mail-follow-ups, for the mysql-queries as well as the file-follow-ups. ESIform will replace your placeholders by the referenced field-values in a normal or formatted way.

How does a placeholder look like?

A placeholder will be replaced automatically by the referenced field-value. This works because the placeholder holds the id-name of the form-field. For example if you have a form-field called "firstname" your placeholder has to look like this: {firstname}.

Sometimes you also want to format the entered value of this field. For these situations you can add a little word to your placeholder: {firstname-format}. This little addition formats the given string by some characters. Let's say that the user entered his firstname like this: tino quesito. With the format addition ESIform would return the given value in this way: Tino Quesito. You see, it will just pre-format your string in a way you would probably do by your own. Mainly it handles just upper- and lowercase issues.

You will find examples by navigating through the "types"-explanations.

06.05. x-Return-Values

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

x-Return-values give a nice feature. It helps you inserting returned data in your follow-up from a previous follow-up.

An example to get in touch with it:

Let's say you process a socket-follow-up to another server. The script on this server receives your data and inserts your passed data into a subscriber-database. After this the script is able to generate a unique subscriber-id. Now it's possible to return this generated id to ESiform. ESiform will parse the returned data and cache them. From this moment you have the possibility to access those returned data and insert them in your next follow-up. For example: an e-mail with the subscriber-id which will be sent to your subscriber.

Through this feature ESiform solve the communication problem between different systems.

How does it work?

This feature can be used for the following-types:

- **socket:** Ok, it works like this. When ESiform calls your script it also catches all out prints from your script. This also includes the HTTP-Header which will be send through your Webserver. ESiform will parse all headers which look like this: Content-xreturn:.

You can add header manually. If you are using PHP you can add a header by the build-in-function `header()`. But it's important that there is no out print before you add the header (I think, PHP 5 fixed this problem). Your script should look something like this:

```
1. <?php
2.
3.     /*
4.         here your process for the passed data
5.         let's say you insert the data to a database
6.         and you receive the insert-id.
7.     */
8.     $insert_id = mysql_insert_id();
9.
10.    /* generate a sample reference-id */
11.    $ref_id = 'REF-000-2006-' . $insert_id;
12.
13.    /* and let's return this ... */
14.    header("Content-xreturn: <ref-id: ".$ref_id.">");
15.
16. ?>
```

Now you can access this generated "reference-id" also in ESiform. The value will be add with the "key"-value to your other post-values. So you can access it by using a placeholder-tag {ref-id}. **Pay attention that you dont use a key-value which is already in use.**

- **mysql:** If you use an mysql-follow-up then ESiform will automaticly add the `mysql_insert_id()` to your POST-Values. The key-value for this will be constructed by the given "follow-up-name" and the string "-id".

For example: If you name your mysql-follow-up mysql-test2 then the insert-id will be available by the key-value mysql-test2-id. The value will be available in the same way as mentioned before: {mysql-test2-id} (placeholder).

See now a complete example

Following you will see an socket-follow-up combined with an mail-follow-up which wil make use of the returned-value "ref-id".

Nr.1 Follow-up to the script above:

```
1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
5.         "socket-subscribe" => array(
6.             "host" => "your-server.com",
7.             "port" => 80,
8.             "method" => "POST",
9.             "path" => "/libs/subscribe.php",
10.        ),
11.
12.        "mail-message" => array(
13.            "to" => array($_POST["email"]),
14.            "from" => "office@my-company.com",
15.            "subject" => "Welcome {firstname-format}!",
16.            "text" => "
17.                Hi, here is your reference-id for your
18.                subscription.
19.
20.                Reference-ID: {ref-id}
21.
22.                Have fun!
23.                Regards
24.                God himself ;)
25.            "
26.        ),
27.    ),
28.    "redirect" => "http://www.your-success-url.com",
29. )
30.
31. ?>
```

And if you want just add one follow-up more ...

Nr.2 Follow-up to the script above:

```
1. <?php
2.
3. "follow-ups" => array(
4.     "types" => array(
5.         "socket-subscribe" => array(
6.             "host" => "your-server.com",
7.             "port" => 80,
```

```
8.         "method" => "POST",
9.         "path"   => "/libs/subscribe.php",
10.    ),
11.
12.    "mysql-subscribe" => array(
13.        "host"       => "your-server.com",
14.        "user"       => "mysql",
15.        "password"  => "mysql",
16.        "name"      => "subscriber",
17.        "query"     => "
18.            insert into user values(
19.                null,
20.                '{ref-id}',
21.                '{email}'
22.            )
23.        "
24.    ),
25.
26.    "mail-message"   => array(
27.        "to"         => array($_POST["email"]),
28.        "from"       => "office@my-company.com",
29.        "subject"    => "Welcome {firstname-format}!",
30.        "text"       => "
31.            Hi, here is your reference-id for your
32.            subscription.
33.
34.            Reference-ID: {ref-id}
35.            Your database ID: {mysql-subscribe-id}
36.
37.            Have fun!
38.            Regards
39.            God himself ;)
40.        "
41.    )
42. ),
43. "redirect" => "http://www.your-success-url.com",
44. )
45.
46. ?>
```

07. Tutorials

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

For the following points you will find some tutorials. It's supposed to help you understand ESIform. For me, it's often more easy to see just some tutorials or examples to understand the tool instead of reading first everything. Maybe for some of you guys it's the same.

But anyway, you guys need to read the documentation. First of all because I put a lot of work and effort in this :) and secondly it let you discover all the little features in detail.

Have fun with the little selection of tutorials!

07.01. A Contact form

[+] A [-] B [-] C

The intention of this tutorial is to create a small contact form which will be processed by ESiform. There will be two followUps for this form:

- an e-mail to the contact-person
- an confirmation e-mail to the person who just entered his data

At the end the user will be redirected to a page with an success-message.

- » [Download this tutorial](#)
- » [See this tutorial online](#)

Following you will see the script of this tutorial. The html-template is included.

```
1. <?php
2.
3. /*
4.     set here the correct path
5.     to the esiform-class
6. */
7. require("esiform.class.php");
8.
9. $myform = new esiform(
10.
11.     array(
12.
13.         "config"    => array(
14.             "sexyness"    => true
15.         ),
16.
17.         "fields"    => array(
18.
19.             "address"    => array(
20.                 "type"        => "select",
21.                 "label"       => "Your Address:",
22.                 "required"    => true,
23.                 "options"    => array(
24.                     "values"  => array(
25.                         "Mr.",
26.                         "Mrs."
27.                     )
28.                 ),
29.                 "styles"     => array(
30.                     "width"   => "200px",
31.                 )
32.             ),
33.         ),
34.
35.         "name"        => array(
36.             "type"        => "text",
37.             "label"       => "Your Name:",
38.             "required"    => true,
```

```
39.         "styles"      => array(
40.             "width"    => "200px",
41.         )
42.     ),
43.
44.     "email"      => array(
45.         "type"      => "email",
46.         "label"     => "Your E-Mail:",
47.         "required"  => true,
48.         "styles"    => array(
49.             "width"  => "200px",
50.         )
51.     ),
52.
53.     "message"    => array(
54.         "type"      => "textarea",
55.         "label"     => "Your message:",
56.         "required"  => true,
57.         "styles"    => array(
58.             "width"  => "200px",
59.         )
60.     ),
61.
62.     "optional"   => array(
63.         "type"      => "checkbox",
64.         "label"     => "Optional",
65.         "describ"   => "You want another question?",
66.         "options"   => array(
67.             "values" => array(
68.                 "Yes"
69.             )
70.         ),
71.         "activate"  => array(
72.             "last-question"
73.         )
74.     ),
75.
76.     "last-question" => array(
77.         "type"      => "select",
78.         "label"     => "Your age?",
79.         "options"   => array(
80.             "values" => array(
81.                 "< 20",
82.                 "21 - 25",
83.                 "26 - 30",
84.                 "31 - 35",
85.                 "> 40",
86.             )
87.         ),
88.         "styles"    => array(
89.             "width"  => "200px",
90.         )
91.     ),
92. ),
93.
94. ),
```

```
95.
96.     "follow-ups"    => array(
97.         "types"     => array(
98.             "mail-office" => array(
99.                 "to"         => array("you@domain.com"),
100.                "from"        => "{email}",
101.                "from-name"   => "{name}",
102.                "subject"    => "Request: contact form",
103.                "text"       => "
104. A message from {name-format} (Age: {last-question})
105. {message}
106.         ",
107.         ),
108.         "mail-user" => array(
109.             "to"         => array("{email}"),
110.             "from"        => "you@domain.com",
111.             "subject"    => "Your Request",
112.             "text"       => "
113. Hello {address} {name-format},
114.
115. thanks for your request. We will
116. get back to as soon as possible!
117.
118. Regards
119. The Office
120.         ",
121.         ),
122.         "redirect" => "http://www.efides.com/ok.html",
123.     ),
124.     "template" => '
125. <html>
126. <head>
127.     <title>
128.         ESIfirm tutorial: contact form
129.     </title>
130.
131.     <style type="text/css">
132.         body {
133.             font-family:trebuchet ms;
134.             font-size:12px
135.         }
136.
137.         #field-box-last-question {
138.             background-color:#ffc;
139.             padding:10px;width:200px
140.         }
141.     </style>
142. </head>
143. <body>
144.
145.     <form action="" method="post">
146.         <h1>Tutorial: Contact form</h1>
147.
148.
149.
150.
```

```
151.         {error-message}
152.
153.         To contact us please enter your data and your
154.         message.<br /> We will get back to as soon
155.         as possible.
156.
157.         <span style="color:#c60">
158.             * Required fields
159.         </span>
160.
161.         <p>
162.             <span style="width:100px">
163.                 {address:label}
164.             </span>
165.             &nbsp;{address:field}
166.         </p>
167.
168.         <p>
169.             <span style="width:100px">
170.                 {name:label}
171.             </span>
172.             &nbsp;{name:field}
173.         </p>
174.
175.         <p>
176.             <span style="width:100px">
177.                 {email:label}
178.             </span>
179.             &nbsp;{email:field}
180.         </p>
181.
182.         <p>
183.             <span style="width:100px">
184.                 {message:label}
185.             </span>
186.             &nbsp;{message:field}
187.         </p>
188.
189.         <p>
190.             <span style="width:100px">
191.                 {optional:label}
192.             </span>
193.             &nbsp;{optional:field}<br />
194.             <small style="color:#c60">{optional:describ}</small>
195.         </p>
196.
197.         <div id="field-box-last-question">
198.             {last-question:label}<br />{last-question:field}
199.         </div>
200.
201.         <p>
202.             <input type="submit" value="Send now!" />
203.         </p>
204.     </form>
205.
206. </body>
```

```
207.     </html>
208.     '
209.
210.     )
211.
212. );
213.
214. ?>
```

07.02. A Newsletter form

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

The intention of this tutorial is to create a form for a newsletter subscription. All possible Newsletter are pre-selected and the user has to choose at least 2 newsletter. There will be two followUps for this form:

- mysql follow-Up to insert the new address into the database
- a doubleOpt-In Mail to the subscriber

At the end the user will be redirected to a page with an success-message.

- » [Download this tutorial](#)
- » [See this tutorial online](#)

Following you will see the script of this tutorial. The html-template is included.

```
1. <?php
2.
3. /*
4.     set here the correct path
5.     to the esiform-class
6. */
7. require("../..../lib/esiform.class.php");
8.
9. $myform = new esiform(
10.
11.     array(
12.
13.         "config"    => array(
14.             "sexyness"    => true
15.         ),
16.
17.         "fields"    => array(
18.
19.             "newsletter"    => array(
20.                 "type"        => "checkbox",
21.                 "label"       => "Our Newsletter:",
22.                 "describ"     => "
23.                     Please choose at least two
24.                     of our newsletters!
25.                 ",
26.                 "required"    => true,
27.                 "options"     => array(
28.                     "keys"        => true,
29.                     "values"     => array(
30.                         1    => "Sports News",
31.                         2    => "Financial News",
32.                         3    => "Fashion Update",
33.                         4    => "TrashTalk!"
34.                     ),
35.                     "select"     => array(
36.                         1,2,3,4
37.                     ),
```

```
38.         ),
39.         "valid" => array(
40.             "choose-min" => 2
41.         )
42.     ),
43.
44.     "email" => array(
45.         "type" => "email",
46.         "label" => "Your E-Mail:",
47.         "required" => true,
48.         "styles" => array(
49.             "width" => "200px",
50.         )
51.     ),
52.
53.     "personal" => array(
54.         "type" => "checkbox",
55.         "label" => "
56.             I want to personalize my newsletter
57.         ",
58.         "options" => array(
59.             "values" => array(
60.                 "Yes"
61.             )
62.         ),
63.         "activate" => array(
64.             "address",
65.             "firstname",
66.             "lastname"
67.         )
68.     ),
69.
70.     "address" => array(
71.         "type" => "select",
72.         "label" => "Your Address:",
73.         "options" => array(
74.             "values" => array(
75.                 "",
76.                 "Mr.",
77.                 "Mrs."
78.             )
79.         ),
80.         "styles" => array(
81.             "width" => "200px",
82.         )
83.     ),
84. ),
85.
86.     "firstname" => array(
87.         "type" => "text",
88.         "label" => "Your Firstname:",
89.         "styles" => array(
90.             "width" => "200px",
91.         )
92.     ),
93.
```

```
94.         "lastname"    => array(
95.             "type"      => "text",
96.             "label"     => "Your Lastname:",
97.             "styles"    => array(
98.                 "width" => "200px",
99.             )
100.        ),
101.    ),
102. ),
103.
104. "follow-ups" => array(
105.     "types" => array(
106.
107.         "mysql-subscription" => array(
108.             "host"      => "localhost",
109.             "user"      => "root",
110.             "password"  => "",
111.             "name"      => "test",
112.             "query"     => "
113.                 insert into subscriptions values(
114.                     null,
115.                     '{newsletter}',
116.                     '{email}',
117.                     '{address}',
118.                     '{firstname}',
119.                     '{lastname}'
120.                 )
121.             "
122.         ),
123.
124.         "mail-user" => array(
125.             "to"        => array("{email}"),
126.             "from"      => "you@domain.com",
127.             "subject"   => "Please confirm ...",
128.             "text"      => "
129. Dear subscriber,
130.
131. thanks for your interest! Please
132. confirm your subscription by clicking
133. on the following URL:
134.
135. http://www.your-domain.com/confirm.php?id={mysql-subscription-id}
136.
137. Regards
138. The Office
139.             "
140.         ),
141.
142.     ),
143.
144.     "redirect" => "http://www.efides.com/ok.html",
145. ),
146.
147. "template" => '
148. <html>
149. <head>
150.     <title>
```

```
151.         ESiform tutorial: newsletter form
152.     </title>
153.
154.     <style type="text/css">
155.         body {
156.             font-family:trebuchet ms;
157.             font-size:12px;
158.             background-color:#eee;
159.         }
160.
161.         .esiform-label {
162.             font-weight: bold;
163.         }
164.
165.         #field-box-address {
166.             background-color:#ffc;
167.             padding:10px;width:250px
168.         }
169.     </style>
170. </head>
171. <body>
172.
173.     <form action="" method="post">
174.
175.         <h1>Tutorial: Newsletter form</h1>
176.
177.         {error-message}
178.
179.         To subscribe our newsletter please choose at
180.         least one newsletter<br />and enter your
181.         personal data.
182.
183.         <span style="color:#c60">
184.             * Required fields
185.         </span>
186.
187.         <p>
188.             {newsletter:label}<br />
189.             {newsletter:describ}<br />
190.             {newsletter:field}
191.         </p>
192.
193.         <p>
194.             <span style="width:100px">
195.                 {email:label}
196.             </span>
197.             &nbsp;{email:field}
198.         </p>
199.
200.         <p>
201.             {personal:field} ,&nbsp;{personal:label}
202.         </p>
203.
204.         <div id="field-box-address">
205.             <p>
206.                 <span style="width:100px">
```

```
207.         {address:label}
208.     </span>
209.     &nbsp; {address:field}
210. </p>
211.
212. <p>
213.     <span style="width:100px">
214.         {firstname:label}
215.     </span>
216.     &nbsp; {firstname:field}
217. </p>
218.
219. <p>
220.     <span style="width:100px">
221.         {lastname:label}
222.     </span>
223.     &nbsp; {lastname:field}
224. </p>
225. </div>
226.
227. <p>
228.     <input type="submit" value="Send now!" />
229. </p>
230.
231. </form>
232.
233. </body>
234. </html>
235. '
236.
237. )
238.
239. );
240.
241. ?>
```

07.03. An E-Mail-FollowUp with attachments [A] [-] B [-] C

The intention of this tutorial is to show you an example of an e-mail follow-up with attachments. The e-mail will consist out of two attachments:

- an outline attachment
- and an inline-attachment (html)

The user has just to fill in a valid e-mail address. At the end the user will be redirected to a page with an success-message.

» [Download this tutorial](#)

Following you will see the script of this tutorial. The html-template is included.

```
1. <?php
2.
3. /*
4.     set here the correct path
5.     to the esiform-class
6. */
7. require("esiform.class.php");
8.
9. $myform = new esiform(
10.
11.     array(
12.
13.         "fields"     => array(
14.
15.             "email"     => array(
16.                 "type"         => "email",
17.                 "label"        => "Your E-Mail:",
18.                 "required"     => true,
19.             ),
20.
21.         ),
22.
23.         "follow-ups"   => array(
24.             "types"     => array(
25.                 "mail-attachment" => array(
26.                     "to"         => array("{email}"),
27.                     "from"        => "ehrich@finanzpark.de",
28.                     "subject"     => "An e-mail with attachment",
29.                     "text"        => "Howdy! Bye bye",
30.                     "html"       => '
31.                         <p>
32.                             Howdy!<br />
33.                             
34.                         </p>
35.
36.                         Bye bye!
37.                     ',
38.                 "attachments" => array(
```

```
39.         array(
40.             "path"    => "images/attachment-1.jpg",
41.             "name"    => "your-attachment-1.jpg",
42.             "type"    => "image/jpeg"
43.         )
44.     ),
45.     "embedded"      => array(
46.         array(
47.             "path"    => "images/attachment-2.jpg",
48.             "name"    => "your-attachment-2.jpg",
49.             "cid"     => "me",
50.             "type"    => "image/jpeg"
51.         )
52.     )
53. ),
54. ),
55.     "redirect"      => "http://www.efides.com/ok.html",
56. ),
57.
58. "template" => '
59. <html>
60. <head>
61.     <title>
62.         ESiform tutorial: E-Mail attachment follow-Up
63.     </title>
64.
65.     <style type="text/css">
66.         body {
67.             font-family:trebuchet ms;
68.             font-size:12px
69.         }
70.
71.         #field-box-last-question {
72.             background-color:#ffc;
73.             padding:10px;width:200px
74.         }
75.     </style>
76. </head>
77. <body>
78.
79. <form action="" method="post">
80.     <h1>Tutorial: E-Mail attachment follow-Up</h1>
81.
82.     {error-message}
83.
84.     Please enter a valid e-amil-address to
85.     receive an attachment-e-mail.
86.
87.     <span style="color:#c60">
88.         * Required fields
89.     </span>
90.
91.     <p>
92.         {email:label}&nbsp;{email:field}
93.     </p>
94.
```

```
95.         <input type="submit" value="Send now!" />
96.     </form>
97.
98. </body>
99. </html>
100. '
101.
102. )
103.
104. );
105.
106. ?>
```

08. Support

[\[+\] A](#) [\[-\] B](#) [\[-\] C](#)

Feel free to contact me if you have any questions about my work. I really appreciate any note about possible improvements for the class as well as the documentation.

I would also be happy to see for which Projects & Websites ESIFORM is used for. So would be great if you guys can just leave me a quick note about that.

- Updates can be found at:

<http://www.efides.com>

- Contact me at:

ehrich@efides.com