ELOwf Developer’s Guide

[Date: 11.11.2016 | Program version: 10.01.000]

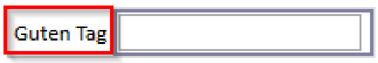
Contents

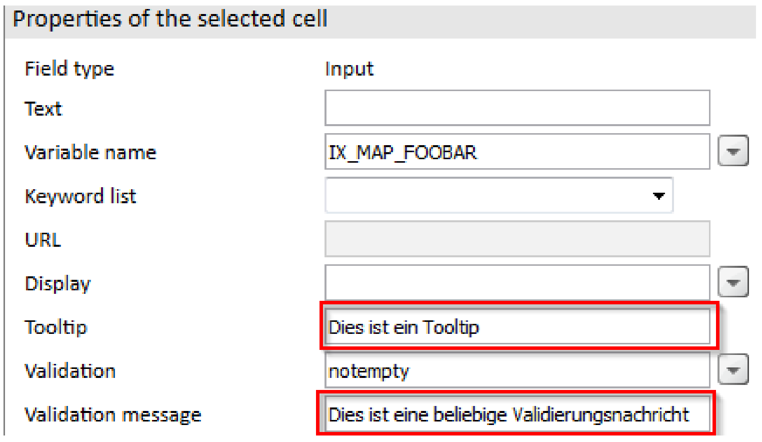
1 Translations / Localization

Several features enable the localization / internationalization of workflow forms, so that labels and tooltips are automatically translated according to the client's language.

1.1 Using translation tables

The "classic way" to internationalize texts is to directly use labels and hints in full text, which will then be automatically replaced using the translation tables available in the clients.

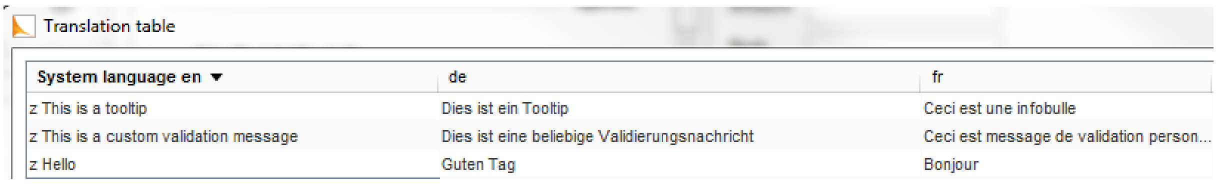




Properties of the selected cell

Field type	Input
Text	
Variable name	IX_MAP_FOOBAR
Keyword list	
URL	
Display	
Tooltip	Dies ist ein Tooltip
Validation	notempty
Validation message	Dies ist eine beliebige Validierungsnachricht

The translation table would look like:

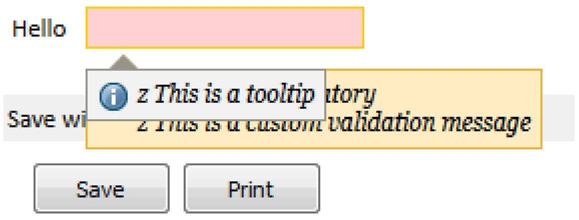


Translation table

System language en	de	fr
z This is a tooltip	Dies ist ein Tooltip	Ceci est une infobulle
z This is a custom validation message	Dies ist eine beliebige Validierungsnachricht	Ceci est message de validation person...
z Hello	Guten Tag	Bonjour

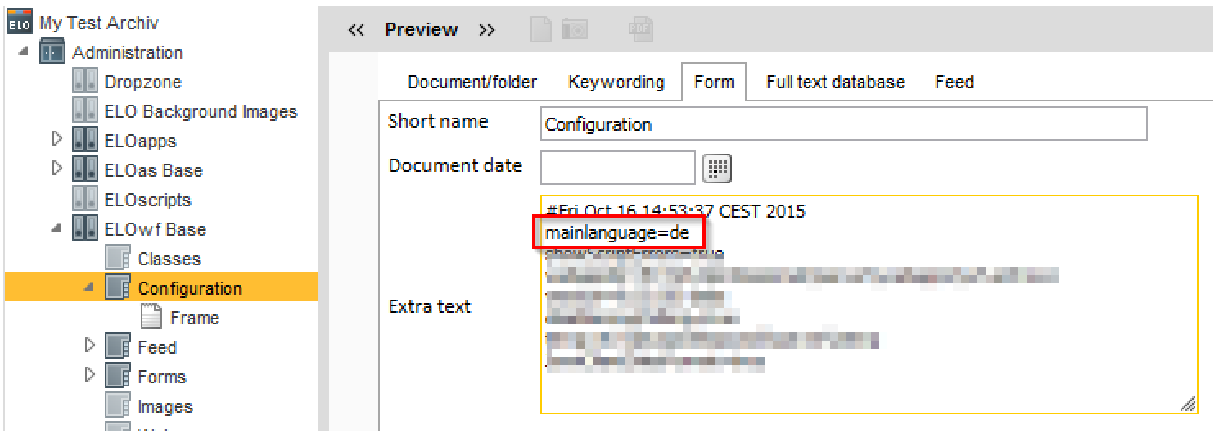
Information: The leading "z" in the screenshot was only used to sort the messages so they can be displayed.

And the result for an English-language client would be:



The screenshot is a bit crowded, but you should get the idea. If no translations are found for a particular text, the original text is kept.

As you also see in this example, the reference language used by ELOwf is not necessarily the system installation language. The standard language is defined as "mainlanguage" in the ELOwf `Configuration` folder:



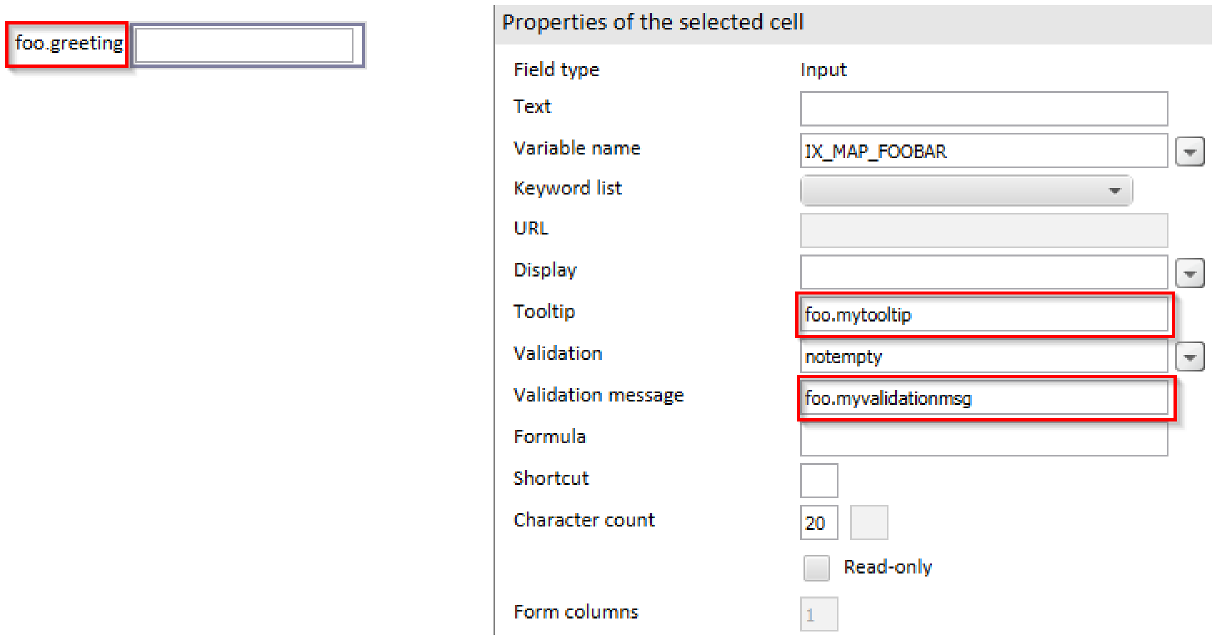
Short name	Configuration
Document date	

Extra text	#Fri Oct 16 14:53:37 CEST 2015
	mainlanguage=de
	[redacted]

This is the language used as the "reference" when searching for translations.

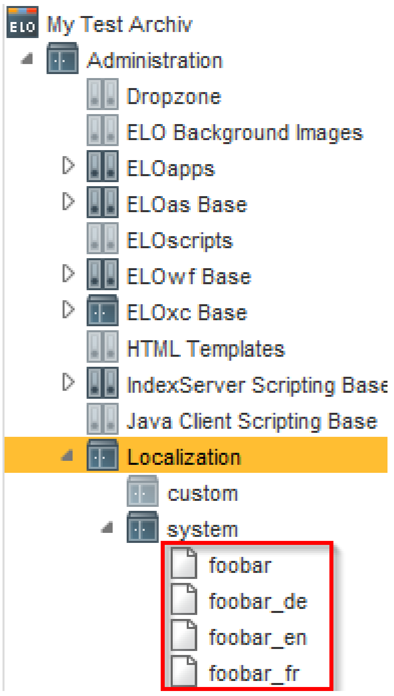
1.2 Using properties files

The "new way" to localize texts is to use translation keys in the form, which uses .property files to deliver the translations. This also enables you to provide custom-translated messages via scripting, which is not otherwise possible.



Field type	Input
Text	
Variable name	IX_MAP_FOOBAR
Keyword list	
URL	
Display	
Tooltip	foo.mytooltip
Validation	notempty
Validation message	foo.myvalidationmsg
Formula	
Shortcut	
Character count	20
	Read-only
Form columns	1

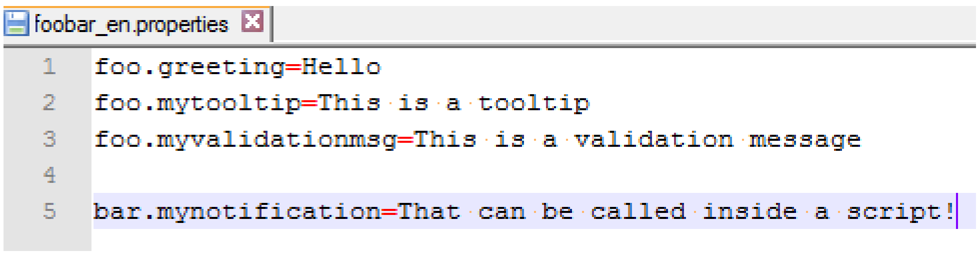
The property files can be stored in the `Administration¶Localization¶system` folder.



Each of these 4 files is a properties file for the corresponding language. The "foobar" without a trailing language abbreviation is used by default if an unexpected/unknown language is requested.

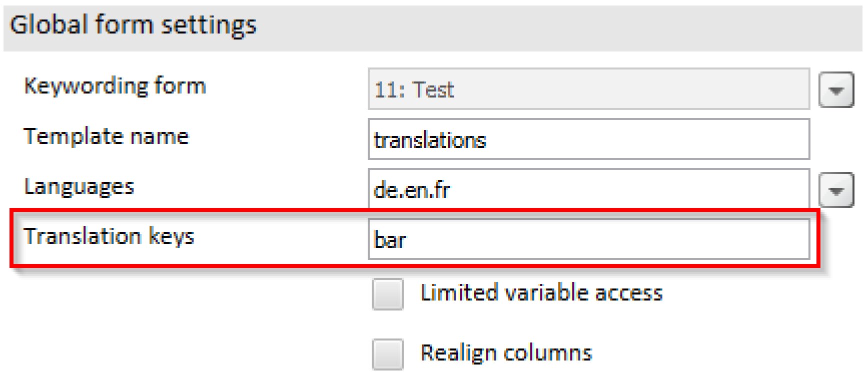
Information: You can use an arbitrary document name (other than the language abbreviation) for language files. All keys are loaded into the same language-specific memory space at runtime.

The content of "foobar_en.properties" may look like this:



```
foo.greeting=Hello
foo.mytooltip=This is a tooltip
foo.myvalidationmsg=This is a validation message

bar.mynotification=That can be called inside a script!
```

You can also import a subset of these properties to make them available through scripting. To achieve this, write the translation key prefix in the property shown below. In this example, all keys starting with "bar" will be made available.



Global form settings	
Keywording form	11: Test
Template name	translations
Languages	de.en.fr
Translation keys	bar
	☐ Limited variable access
	☐ Realign columns

These will be accessible in scripts via "elo.locale.store". In our case, delivering a message box with a translated message would be as simple as:

```
eloAlert(elo.locale.store['bar.mynotification'])
```

1.3 Comparison

As a rule of thumb, if you have just a couple of simple forms requiring translation into another language, using translation tables is perfectly fine. If you have several large forms or several languages, the properties-based solution is advisable.

Using properties files provides the following advantages:

- Scripts can also access translated texts, for example for message boxes
- Sometimes, a same word should be translated into different words in another language, depending on its context. This is not possible with translation tables.
- Properties files are supported by professional translation software.
- Properties files are versioned in the ELO repository.

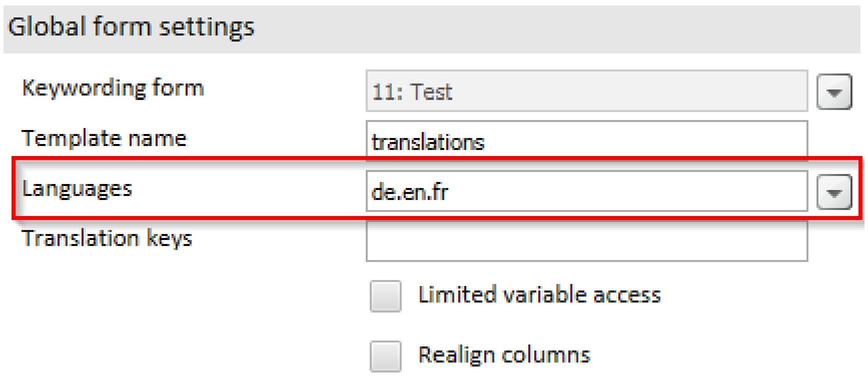
This may be overkill for small repositories with low translation requirements, but become increasingly useful as the need for translations increases.

1.4 When modifying translations

ELOwf caches translations for performance reasons. If you modify the translation tables or the property files, please reload the ELOwf module for the changes to take effect. When modifying the properties files, reload the Indexserver as well.

1.5 Dynamic language selection

Inserting a set of languages in the "languages" form settings:



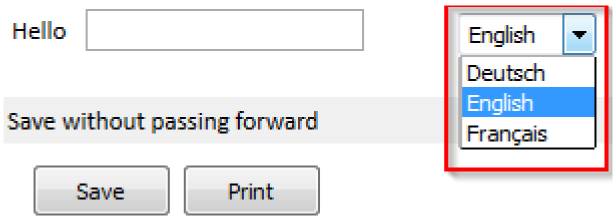
Global form settings

Keywording form	11: Test
Template name	translations
Languages	de.en.fr
Translation keys	

☐ Limited variable access

☐ Realign columns

Adds a combo box to a form to be able to switch the language in the form itself.



2 Date fields

Despite not being obvious at first sight, each "Date" field can also hold additional time information. For example, you can choose "Nov 12, 2015" with the date picker, but you can also add a time if desired, like "Nov 12, 2015 7:08 am". Note that seconds will not be saved.

2.1 The format

- If you use the ELO Java Client, forms will use the same format. This format is either a specific pattern or locale-dependent (the default setting).
- Otherwise, the form will use the German format ("dd.MM.yyyy HH:mm") …except if the language is Hungarian, in which case it will use ("yyyy.MM.dd HH:mm")

Basically, if you don't use the Java Client, you can't use other date/time formats.

When writing, inserting to or updating the date field, it is *recommended* to use the same format. However, it is also possible to use:

- The alternative format specified in the Java Client
- The universal ISO 8601 format (such as "2014-09-08T08:02:17")

2.2 Internationalization & time zones

Note that if you log on with a client in another time zone, the IX_DOCDATE and IX_CREATEDATE will be adapted to the local time.

For example, a document with a creation date like "Nov 12, 2015 7:08" will be adapted into "Nov 12, 2015 10:08 am" if the user is in a time zone 3 hours "later". This applies as well if it was a date "only", which will be considered to be the date at midnight. For example, "Nov 12, 2015" is the same as "Nov 12, 2015, 0:00 am".

All other dates in the form will be left unchanged.

3 Amount and number fields

3.1 Difference

The main difference between amount and number fields is that “amount” fields have thousands separators, while “num” fields do not. Amount fields are typically used for large values or currencies. The thousand and decimal separator are also language-dependent. An amount field like “123.456,78” in German would be displayed as “123,456.78” in English.

3.2 Behavior

Here is how “amount nk:2” behaves (in English locale):

- “12345” -> “12,345.00”   <= displays a thousands separator
- “12,345” -> “12,345.00” <= interprets “,” as thousands separator
- “12.345” -> “12.35”

Here is how “num nk:2” behaves (in English locale):

- “12345” -> “12345.00” <= no thousands separator
- “12,345” -> “12.35”   <= interprets both “,” and “.” as decimal separator
- “12.345” -> “12.35”

Amount fields can accept input with or without thousands separators, but will always display it with them.

If the following flag is set in the header script:

```
ELO.Configuration.Amount.noThousandSep = true;
```

Then “amount” fields will behave similarly to “num” fields and interpret the last separator as decimal separator, whether it’s a dot “.” or a comma “,”.

3.3 Format

What is used as thousand/decimal separators?

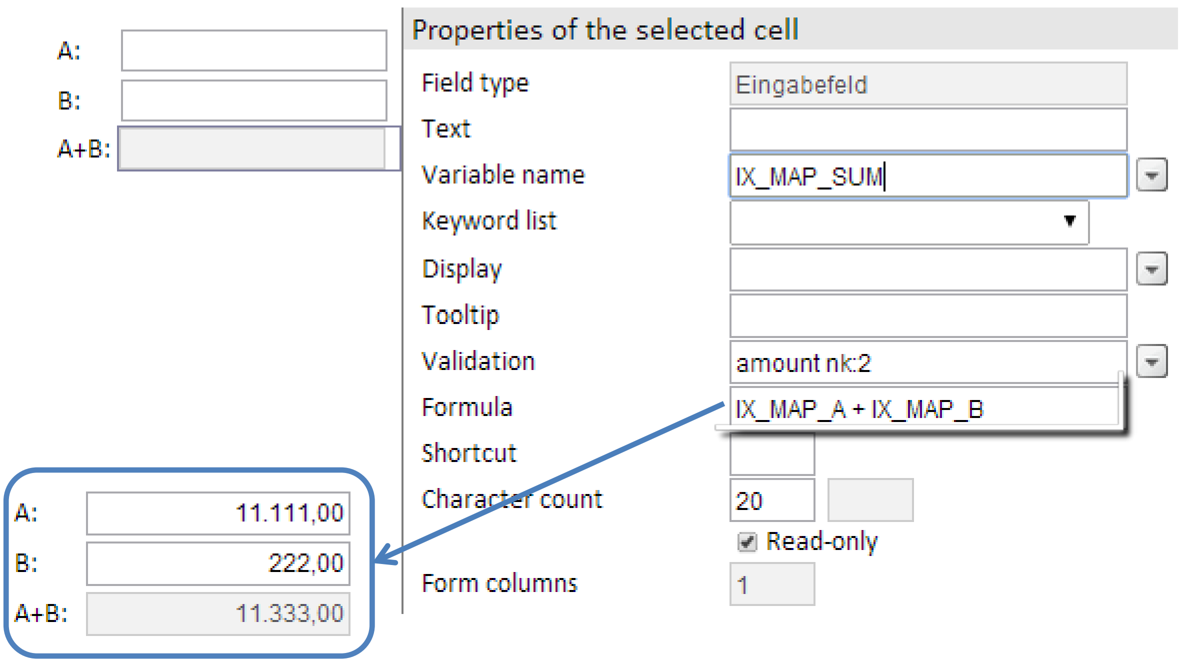
- If using the Java Client, it is defined in the technical presets under “Configuration”. There, you have two options: “Default” (recommended, which is also language-dependent). You can also specify custom separators (not recommended, as the separator stays the same even if you change to another language)
- Otherwise, the locale’s default is used

If you are unsure of which separator is used, you can verify the setting in the form header. These are stored in the following variables:

```
ELO.Configuration.Amount.ThousandSep = …
ELO.Configuration.Amount.DecimalSep = …
```

3.4 Formulas

Numeric input fields can be computed using arbitrary formulas.



A:
B:
A+B:

Properties of the selected cell

Field type	Eingabefeld
Text	
Variable name	IX_MAP_SUM
Keyword list	
Display	
Tooltip	
Validation	amount nk:2
Formula	IX_MAP_A + IX_MAP_B
Shortcut	
Character count	20
	✓ Read-only
Form columns	1

A:	11.111,00
B:	222,00
A+B:	11.333,00

The special function ESum(IX_MAP_N) as a formula can be used to sum all values. It is the equivalent of IX_MAP_N1 + IX_MAP_N2 + ...

3.5 On-the-fly formulas

In forms, you can also type "=" then any arithmetic formula like "= 0.21 * 1234 + 57" in order to compute something on the fly. Simply end it with "Enter" or leave the field; the result will automatically be computed.

These inline formulas are limited to numerical computations. Other variables are not accessible.

3.6 Scripting

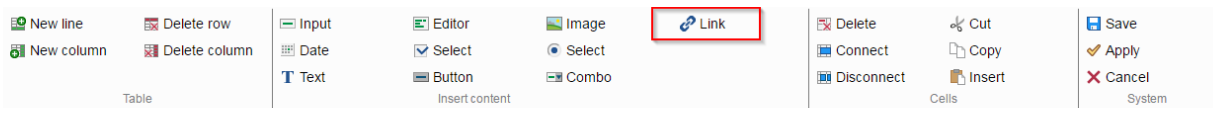
When retrieving values from fields through scripting, be sure to use "$num(...)" and not "$val(...)". The first returns a float, while the latter returns a string in "ELO format": no thousands separator and a comma "," as decimal separator, regardless of the requested language!

If the input contains the value "12,345.67", $val("IX_...") would return the String "12345,67".

In “ELO_PARAMS”, number and amount values are always stored as strings in “ELO format” as well.

4 Link fields

Starting from ELO 10, you can also place links in forms.

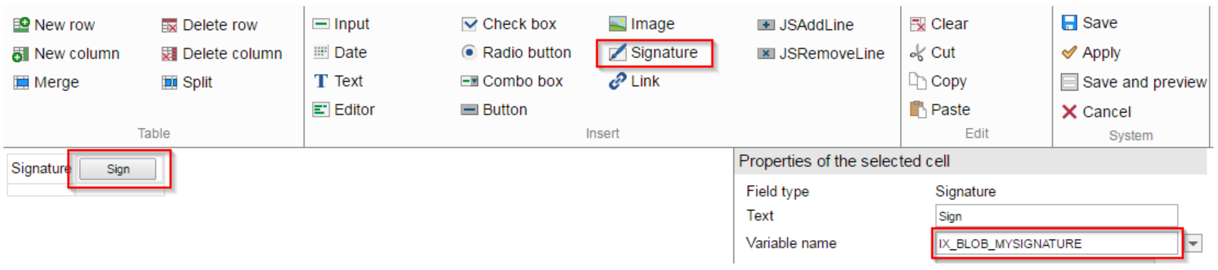


There are two kind of links:

- If the URL is a GUID, clicking the link opens the document in a new window/tab
- If the URL is a "normal" Internet address, the corresponding web page is displayed in a new window/tab

If the URL is left empty in the form designer, the form provides a link to the current document.

5 Signatures

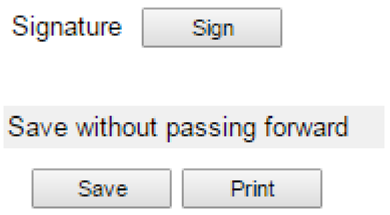


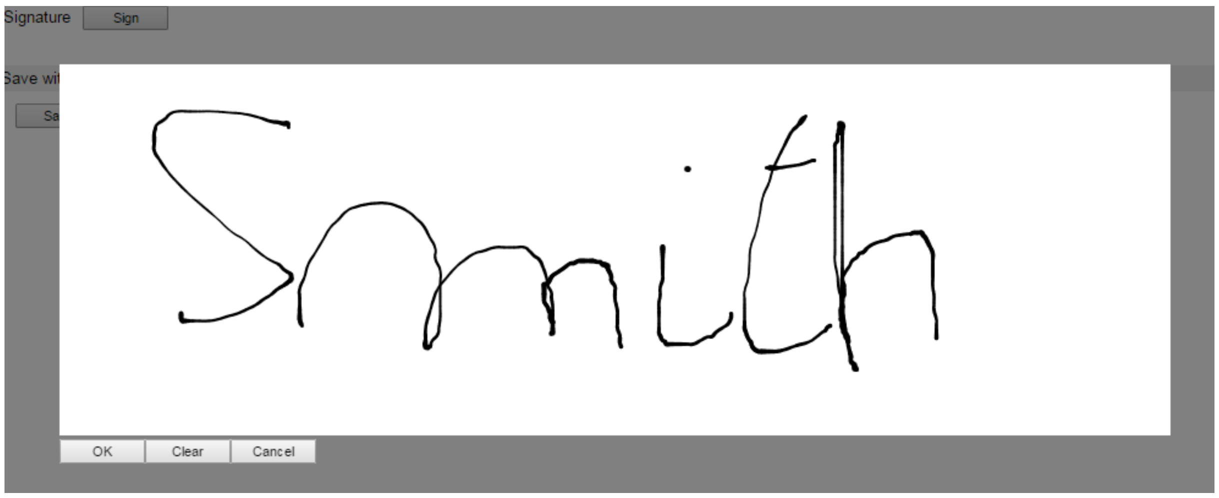
Properties of the selected cell

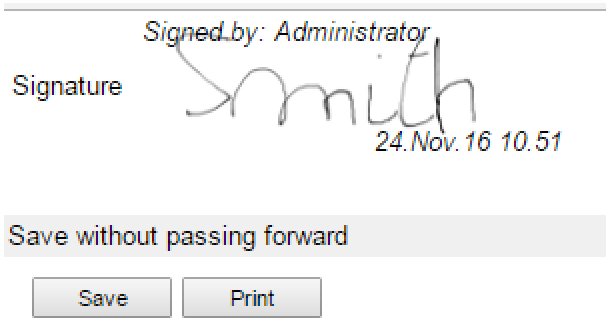
Field type	Signature
Text	Sign
Variable name	IX_BLOB_MYSIGNATURE

The signature widget is available in ELOwf 10.01 and later versions. The widget enables the user to sign a form.

Since the data to be stored by a signature is greater than the limit imposed by keywording and map fields, "BLOB" fields (Binary Large OBject) must be used. These are identified by variables starting with "IX_BLOB_" followed by an arbitrary name.

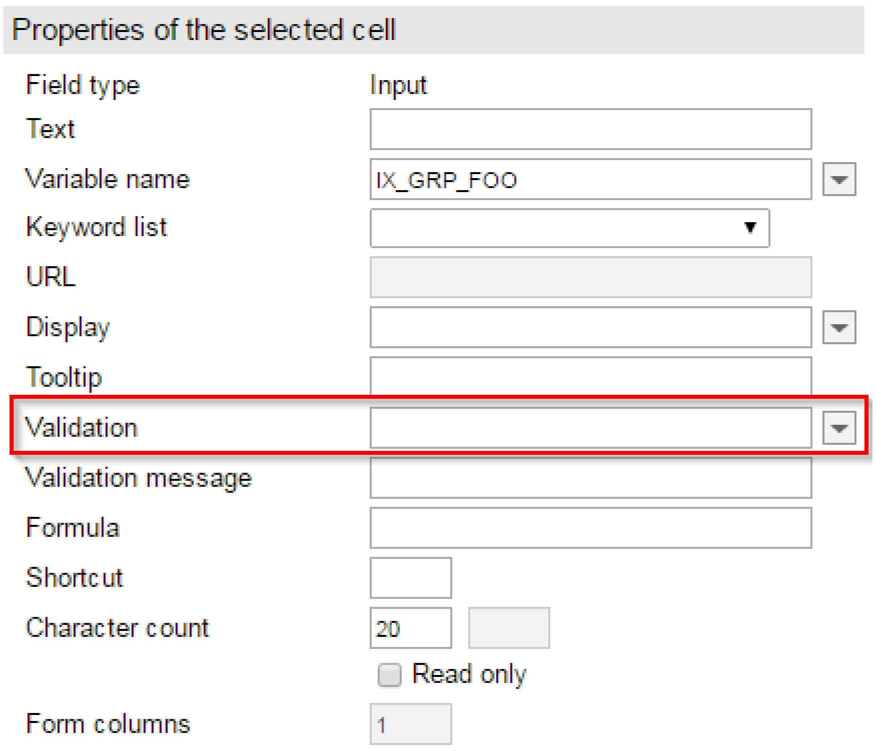






It is possible to use several signatures on a form, signed by the same or different users. The only limitation is that this widget is not intended to be used in combination with “JS_ADDLINE” dynamic lines.

6 Validation



Properties of the selected cell

Field type	Input
Text	
Variable name	IX_GRP_FOO
Keyword list	
URL	
Display	
Tooltip	
Validation	
Validation message	
Formula	
Shortcut	
Character count	20
	☐ Read only
Form columns	1

6.1 Introduction

The keywording form field constraints and the WF form field validation are basically two different sets of functionality.

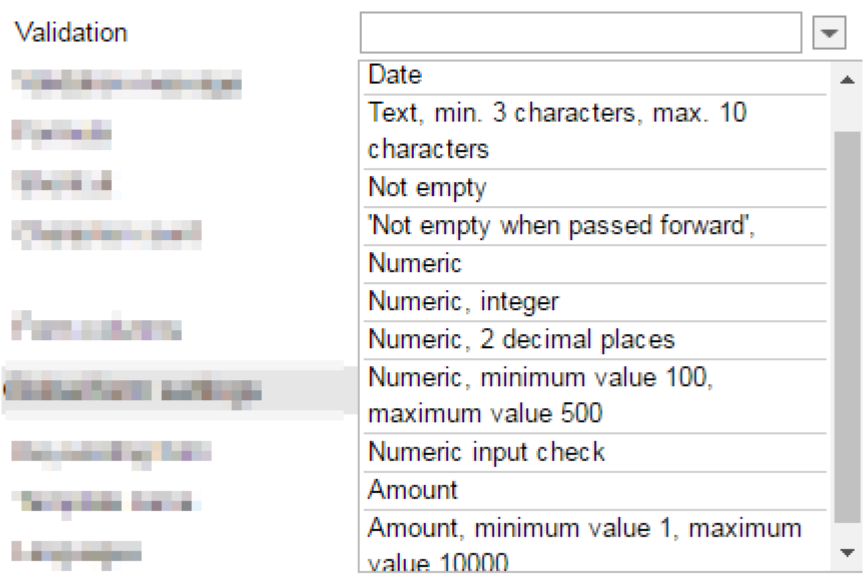
The restrictions specified in the keywording form are *ignored by ELOwf and the Indexserver (*)*. By default, you can save anything in any keywording field, even if it's declared as a number, a date or anything else.

The keywording forms are solely used by the clients (Java, Web, Windows) in order to validate/format the content, but *not* by ELOwf, which uses its own validation.

(*) …except for the min/max length, which *are* the only two keywording form properties enforced for a field.

6.2 Basic behavior

For each field, you can enter a series of predefined constraints:


Validation

- Date
- Text, min. 3 characters, max. 10 characters
- Not empty
- 'Not empty when passed forward',
- Numeric
- Numeric, integer
- Numeric, 2 decimal places
- Numeric, minimum value 100, maximum value 500
- Numeric input check
- Amount
- Amount, minimum value 1, maximum value 10000

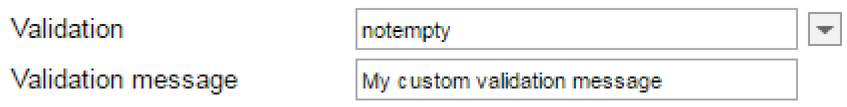
In order to be consistent, these should be set the same as the keywording form constraints, but it is not enforced.

A few more notes:

- For numeric values, the min/max in the keywording form defines the amount of digits, while in the form designer, it defines the min/max value
- The 'not empty when passed forward' validation was created so that the form can be saved with empty fields "temporarily", but must be filled when forwarding the form in a workflow

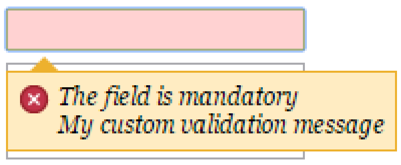
6.3 Custom validation messages

In addition to the default validation messages, it is possible to add a custom message.



Validation	notempty
Validation message	My custom validation message

Results in:


The field is mandatory
My custom validation message

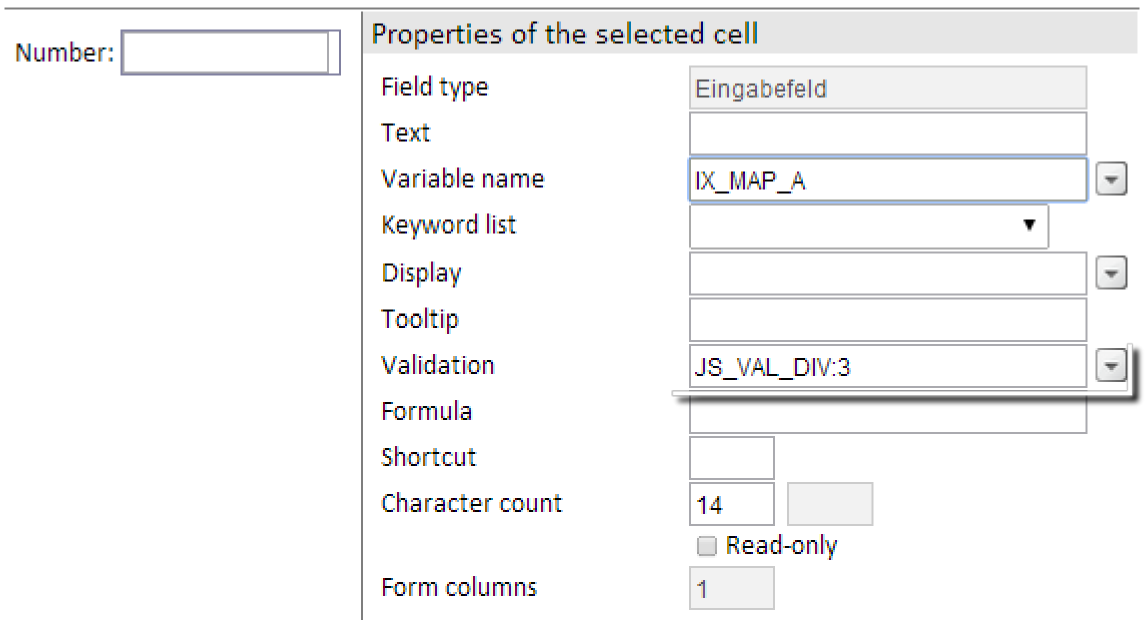
The message is shown additionally to the other error messages. It is also possible to use translation keys for the message.

6.4 Custom validation functions (JS_VAL_...)

Custom functions can be used to validate inputs:

```
JS_VAL_myValidation(fieldName, fieldValue, validationParam)
```

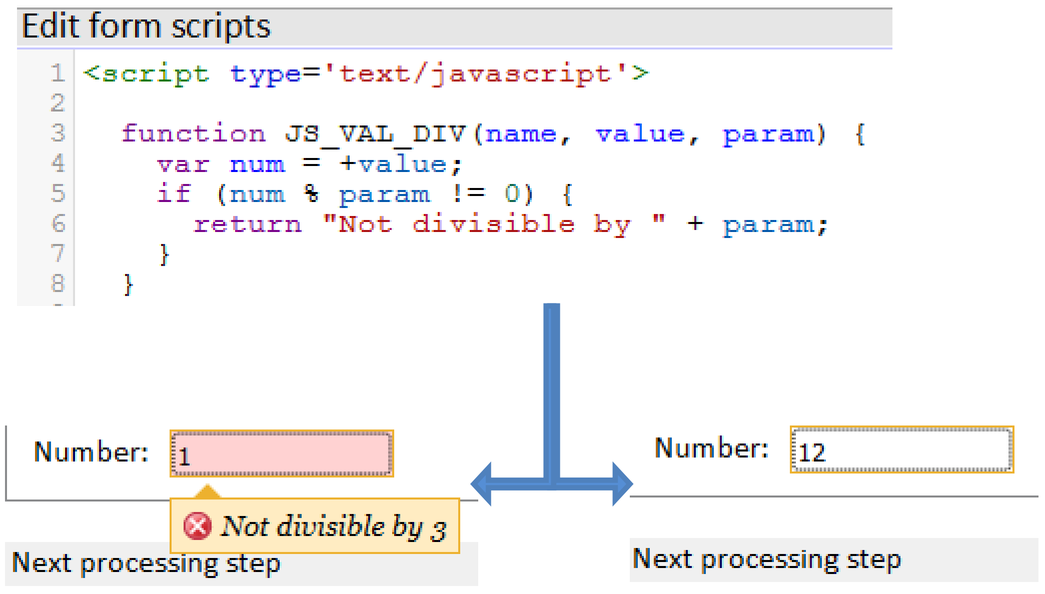
For an invalid field, the function must return the error to be displayed as a non-empty string.



Number:

Properties of the selected cell

Field type	Eingabefeld
Text	
Variable name	IX_MAP_A
Keyword list	
Display	
Tooltip	
Validation	JS_VAL_DIV:3
Formula	
Shortcut	
Character count	14
	Read-only
Form columns	1



Edit form scripts

```
<script type='text/javascript'>

  function JS_VAL_DIV(name, value, param) {
    var num = +value;
    if (num % param != 0) {
      return "Not divisible by " + param;
    }
  }
```

Number: 1

Not divisible by 3

Next processing step

Number: 12

Next processing step

This can be done in addition to other validation constraints. The only limitation of this functionality is that it has to be synchronous.

6.5 Custom filter functions (JS_FILTER_...)

When the user types something to a field, a filter can be applied to transform the value.

For example, the predefined function JS_FILTER_NUMBER allows the user to only input numbers.

Custom filters could be used to only allow digits, uppercase the input, ensure a certain format, or apply any other sort of text transformation.

To do this, add "JS_FILTER_MyFunction" or "JS_FILTER_MyFunction:parameter" to the "validation" property. The following function will be called when a user types or pastes something to the field:

```
JS_FILTER_MyFunc(front, inserted, tail, param)
```

Here is example an example to automatically uppercase the input:

```
function JS_FILTER_Uppercase(front, inserted, tail) {
    return inserted.toUpperCase();
}
```

In this case, the return value is a string and modifies what was inserted. However, it is also possible to modify what comes before and after the entry, by returning an array with the new values ([front, inserted, tail]).

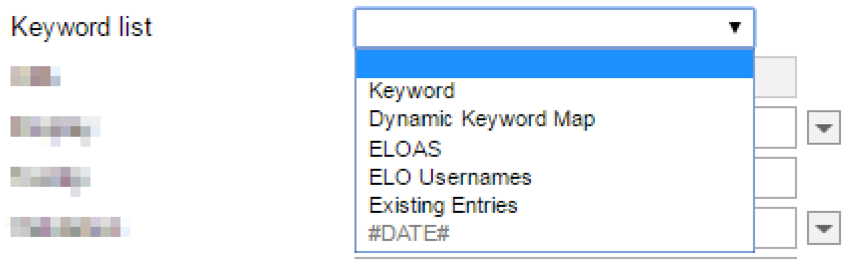
6.6 Can I turn validation on or off depending on the use case?

No. Validation always take place. It is also meaningful. Even if you have a workflow with a "Cancel" operation, the form still has to be saved. And in order to be saved, it has to contain valid data.

Rather than trying to disable validation, we recommend using "onNextClicked(…)" in order to simply fill dummy values where needed before the validation takes place.

7 Lists

7.1 Overview

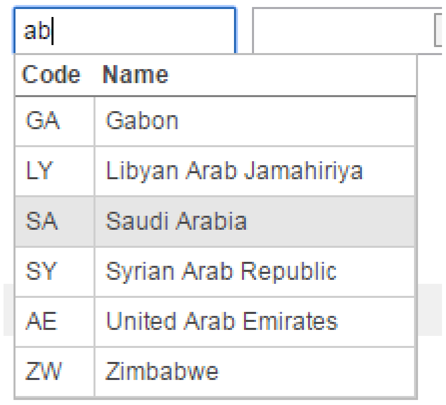


An explanation of these types of lists:

- Keyword: this might be a static keyword list or a dynamic one, depending on what is defined in the keywording form for that field.
- Dynamic Keyword Map: this list is typically used to get values for map fields.
- ELOAS: calls an AS ruleset and returns the results as a list.
- ELO Usernames: retrieves a list of users / groups.
- Existing Entries: when selecting this, the user will receive an autocomplete list of suggestions based on what has already been typed in this field in other documents.

7.2 Dynamic keyword

A dynamic keyword field is a field that can display several columns of content, typically dynamically filtered depending on one or more other field values. It looks like this:



ab

Code	Name
GA	Gabon
LY	Libyan Arab Jamahiriya
SA	Saudi Arabia
SY	Syrian Arab Republic
AE	United Arab Emirates
ZW	Zimbabwe

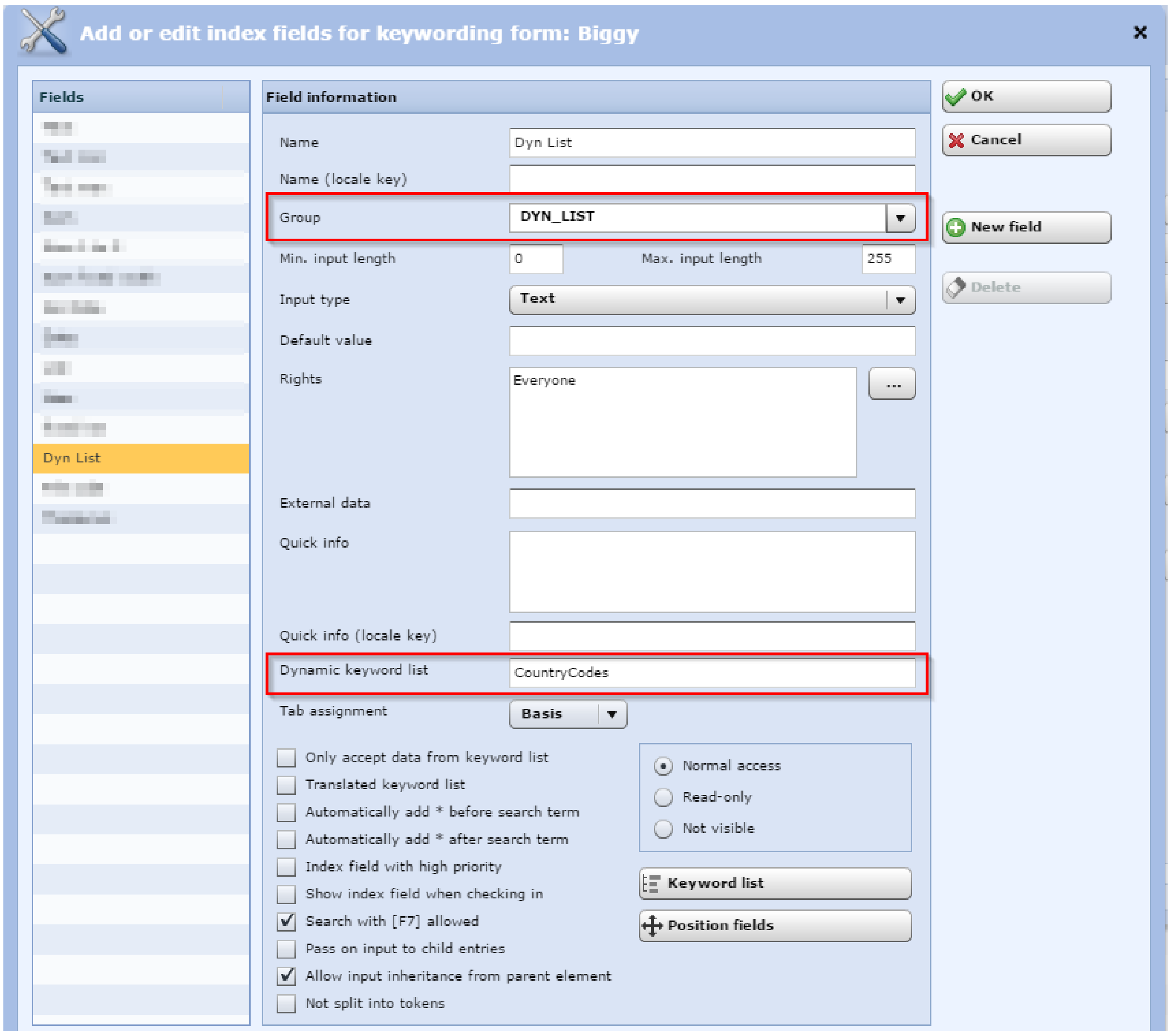
Dynamic keyword lists are a rather advanced feature of ELO that require scripting to work. They are available in the Java Client and the Web Client.

It can be used to:

- Display many columns of data

- Fetch content dynamically (for example: external databases, computed on the fly, elo scripting…)
- Filter based not only on the provided field, but also on other arbitrary fields (for example: what the user has typed and a category that the user previously selected)
- Fill many other fields upon selecting an item (including read-only fields)

The usage of a dynamic keywording list is defined directly in the keywording form:


Add or edit index fields for keywording form: Biggy

Fields
[redacted]
Dyn List
[redacted]

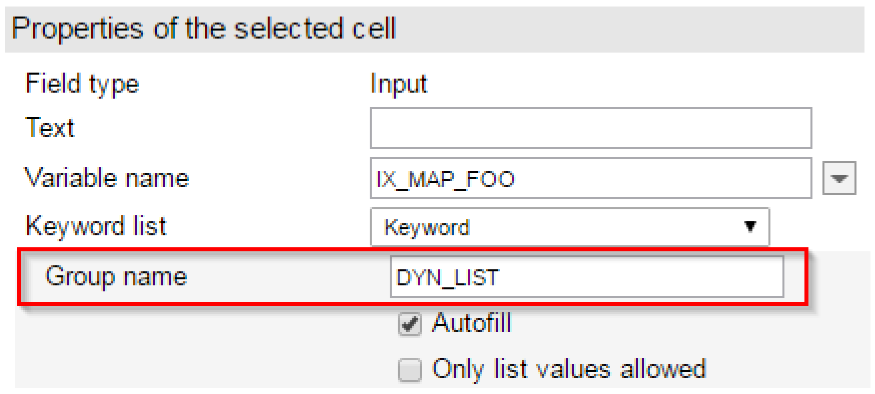
Field information

Name	Dyn List
Name (locale key)	
Group	DYN_LIST
Min. input length	0
Max. input length	255
Input type	Text
Default value	
Rights	Everyone
External data	
Quick info	
Quick info (locale key)	
Dynamic keyword list	CountryCodes
Tab assignment	Basis

- ☐ Only accept data from keyword list
- ☐ Translated keyword list
- ☐ Automatically add * before search term
- ☐ Automatically add * after search term
- ☐ Index field with high priority
- ☐ Show index field when checking in
- ☑ Search with [F7] allowed
- ☐ Pass on input to child entries
- ☑ Allow input inheritance from parent element
- ☐ Not split into tokens

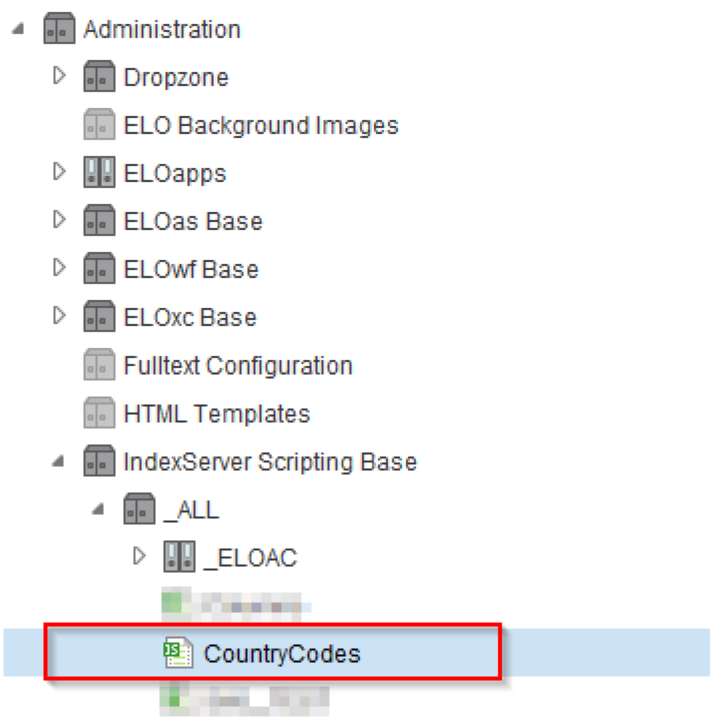
[✓ Normal access] Read-only Not visible

To be used in a formular, it has to be declared as such in the form editor:


Properties of the selected cell

Field type	Input
Text	
Variable name	IX_MAP_FOO
Keyword list	Keyword
Group name	DYN_LIST
	✓ Autofill
	☐ Only list values allowed

Of course, the corresponding "CountryCodes" script must exist and be filed to the `¶Administration¶IndexServer Scripting Base` folder.



Here is an example of a script that provides a static list of country codes and names:

```
importPackage(Packages.de.elo.ix.jscript);
importPackage(Packages.de.elo.ix.scripting);


function getDataIterator() {
  try {
    log.info("getDataIterator(SimpleDatabaseQuery");
    return new DynamicKeywordDataProvider(new CountryCodes());
  } finally {
    log.info(")getDataIterator");
  }
}


function CountryCodes() {
  var index = 0;
  var results = [];

  /* Utility function to filter a list of countries */
  this.filterCountries = function(filter) {
    log.info("filter: " + filter)
    filter = filter.toLowerCase()
    results = [];
    for (var i=0; i<isoCountries.length; i++){
      if (isoCountries[i].cname.toLowerCase().indexOf(filter) >= 0) {
            results.push([isoCountries[i].ccode, isoCountries[i].cname]);
      }
    }
    log.info("after filter: " + results.length);
  }

  /* Called when initializing a dynamic list (via declaration in the
keywording mask) */
  this.open = function(ec, sord, focus) {
    log.info("open");

    this.target = focus;
    // TODO: here, the first keywording line is picked, but ideally it
should be a better suited one
    var filter = sord.objKeys[0].data[0] || "";

    this.filterCountries(filter);
  }

  /* Called when initializing a dynamic list via ELOwf's "Dyn. keywording
map" field */
  this.openMap = function(ec, map, focus) {
    log.info("openMap");
    log.info(JSON.stringify(map));

    this.target = focus;
    var filter = map[focus] || "";

    this.filterCountries(filter);
  }

  this.close = function() {
    log.info("close()");
  }

  this.getNextRow = function() {
    var row = results[index];
    index++;
    log.info("getNextRow(" + row + ")");
    return row;
  }

  /* The name of the columns */
  this.getHeader = function() {
    log.info("getHeader()");
    return ["Code","Name"];
  }

  /* The selected item's target */
  this.getKeyNames = function() {
    log.info("getKeyNames()");
    return [this.focus, "SOME_OTHER_KEYWORDING_FIELD"];
  }

  this.hasMoreRows = function() {
    log.info("hasMoreRows()");
    return (index < results.length - 1);
  }

  /* Either return an error message or leave it empty on success */
  this.getMessage = function() {
    log.info("getMessage()");
    return "";
  }

  this.getTitle = function() {
    log.info("getTitle()");
    return "Country Codes";
  }
}
```

```
var isoCountries = [
    {'ccode' : 'AF', 'cname' : 'Afghanistan'},
    {'ccode' : 'AX', 'cname' : 'Aland Islands'},
    {'ccode' : 'AL', 'cname' : 'Albania'},
    /* ... */
    {'ccode' : 'ZW', 'cname' : 'Zimbabwe'}
]
```

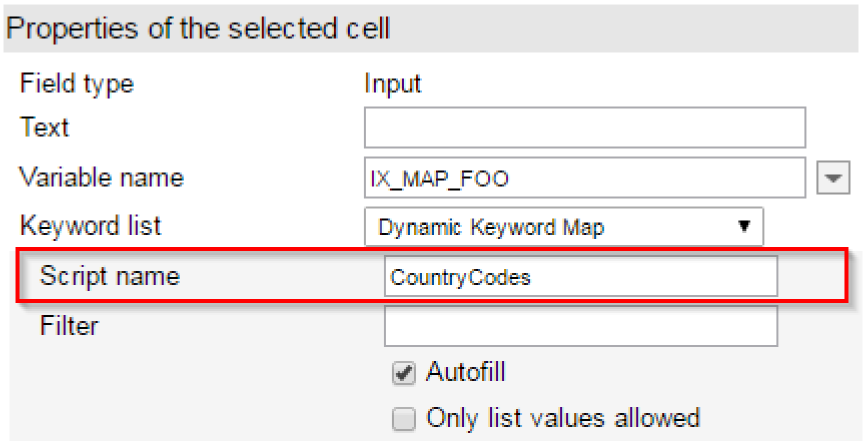
See also:

http://forum.elo.com/supportweb/eloimages/en-elo-dynamic-keyword-lists.19901.pdf

http://www.forum.elo.com/jforum/posts/list/3839.page#19317

7.3 Dynamic keyword map

A dynamic keywording map is similar, but is based on MAP fields. There, you should specify the target script directly, as in the screenshot provided below.


Properties of the selected cell

Field type	Input
Text	
Variable name	IX_MAP_FOO
Keyword list	Dynamic Keyword Map
Script name	CountryCodes
Filter	
	☑ Autofill
	☐ Only list values allowed

Notes

When an item is selected in a dynamic keywording list, the following event function is called in the header script:

```
onDynListItemSelected(item)
```

Please note: It is not possible to combine “Autofill” with “Only list values allowed” for dynamic keyword lists. The main reason why this is the case is that dynamic keyword lists may depend on several input fields and may also modify multiple input fields. It is not currently possible to support multi-field validation.

All lists can also be triggered within the scripts:

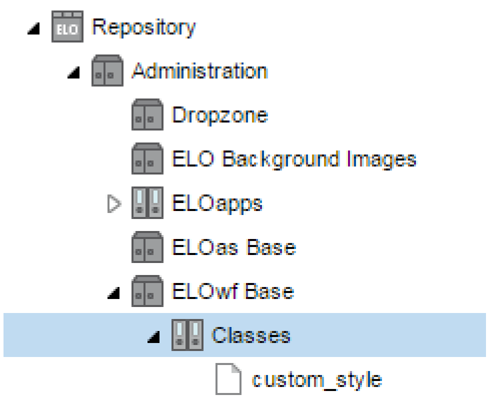
```
/**
 * Calls a specific rule from the AS.
 */
function $listAs(scriptName, param2, param3, onSuccess, onFailure) {

/**
 * $listKw("IX_GRP_DYN_FIELD", ...) will fetch the data from the keywording
 * list defined in the specified field
 */
function $listKw(swlid, onSuccess, onFailure) {


/**
 * $listDyn("MyScript", "foo", ["bar"], ...) will fetch the data from the
 * corresponding 'IndexServer Scripting Base' script.
 * The script will be invoked with "foo" as focus name and {"foo": ...,
 * "bar": ...} as map data, also replacing the wildcards "{i}" and "{*}"
 */
function $listDyn(scriptName, focusfield, filterfields, onSuccess,
onFailure) {
```

8 Adding custom files

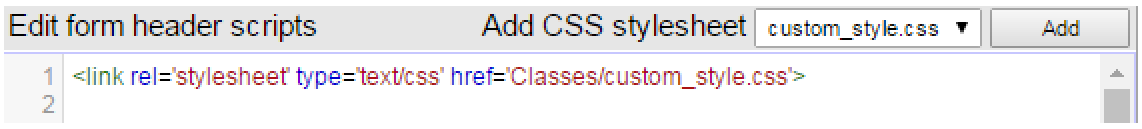
8.1 CSS files

It is possible to customize the design of forms by adding CSS inline in the form header.

If there is a need for more, the best practice would be to upload CSS files to "/Administration/ELOwf Base/Classes".



After adding files, you will have to refresh ELOwf in order for it to be aware of these new files. Afterwards, they will be available in the header of the form header editor:



```
<link rel='stylesheet' type='text/css' href='Classes/custom_style.css'>
```

You can select a stylesheet and click "Add" to add the corresponding line to the header.

9 Events

This chapter describes some events fired in ELOwf forms.

9.1 onInit()

This event is invoked once when the form is loaded.

9.2 inputChanged(elem)

This event is invoked each time the user changes the content of an input, for each key stroke. The input field is provided as argument. The event is not triggered when a script changes a value.

9.3 saveClicked()

The ‘saveClicked’ event is invoked when a form is saved. More precisely, it is invoked before the form is validated and saved.

This can be used to either perform additional validation or to set other form variables before saving.

Example:

```
function saveClicked() {
  if( $num(“IX_GRP_PRICE”) > 10000 && $val(“IX_GRP_TYPE”) == “CHEAP”) ) {
    eloAlert(“Sorry, that’s too expensive!”);
    return false; // cancel it, do not save
  }
  else {
    return true; // ok, proceed with validation and saving
  }
}
```

Starting with ELOwf 10.1, asynchronous processes can also be performed by returning promises:

```
function doSomething(resolve, reject) {
  // fetch or verify some data asynchronously
  // call ‘resolve()’ when successfully finished
  // or ‘reject()’ if the process should be aborted
}

function saveClicked() {
  return new Promise(doSomething);
}
```

The promise is a delayed result that is used to continue the process when it is resolved or rejected.

9.4 nextClicked(id)

This event works exactly the same way as "saveClicked" but is invoked when the form is passed forward to the next workflow node. Likewise, it can return "true" to proceed, "false" to cancel, or a promise to be resolved later.

The only difference between the way this event and the previous event functions is its additional parameter "id," which references the ID of the workflow node to which the workflow is being forwarded.

Instead of the ID, it is also possible to recover the internal node name:

```
var internalNodeName = ELO_PARAMS[“NEXT_” + id].split(‘\t’)[1]
```

10 Miscellaneous

10.1 Printing

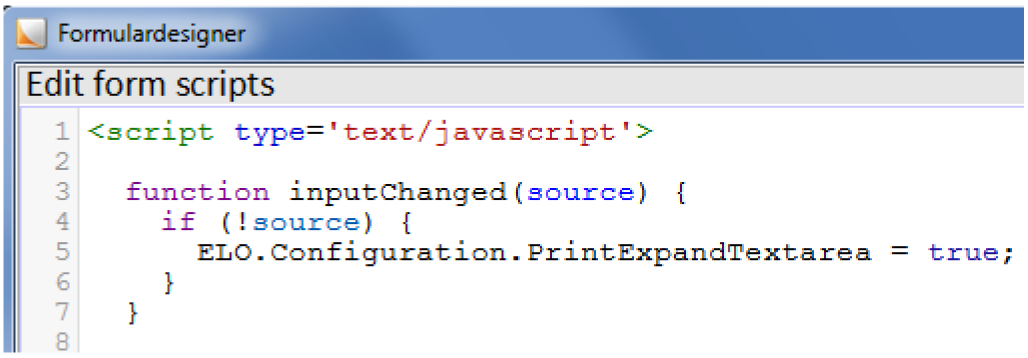
Please note: The following features are only applied when printing when the user chooses the *Print* button. They do not apply when the user presses *Ctrl+P* or prints via a browser menu.

10.1.1 Auto-expand text areas

When printing, in order to always see the full content of text areas, you can use the following flag:

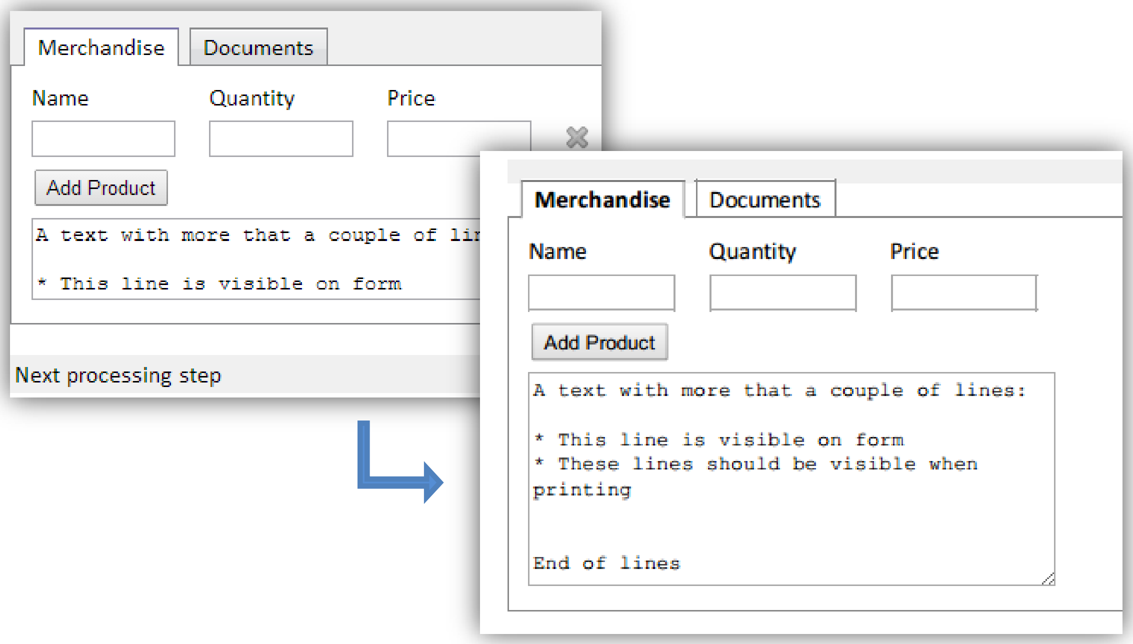
```
ELO.Configuration.PrintExpandTextarea = true;
```

Place this in the `inputChanged()` event function.



```
<script type='text/javascript'>

  function inputChanged(source) {
    if (!source) {
      ELO.Configuration.PrintExpandTextarea = true;
    }
  }

```

By doing so, all text areas will increase in height as needed:



```
A text with more that a couple of li

* This line is visible on form
```

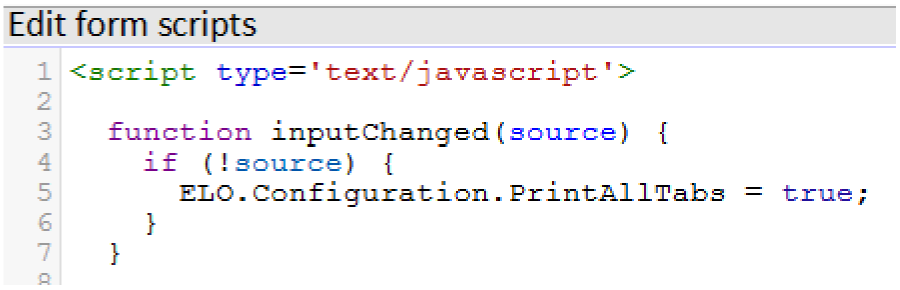
```
A text with more that a couple of lines:

* This line is visible on form
* These lines should be visible when
printing


End of lines
```

10.1.1.1 Print all tabs

When printing, if you want to print all tabs by default, you can set following flag:

```
ELO.Configuration.PrintAllTabs = true;
```

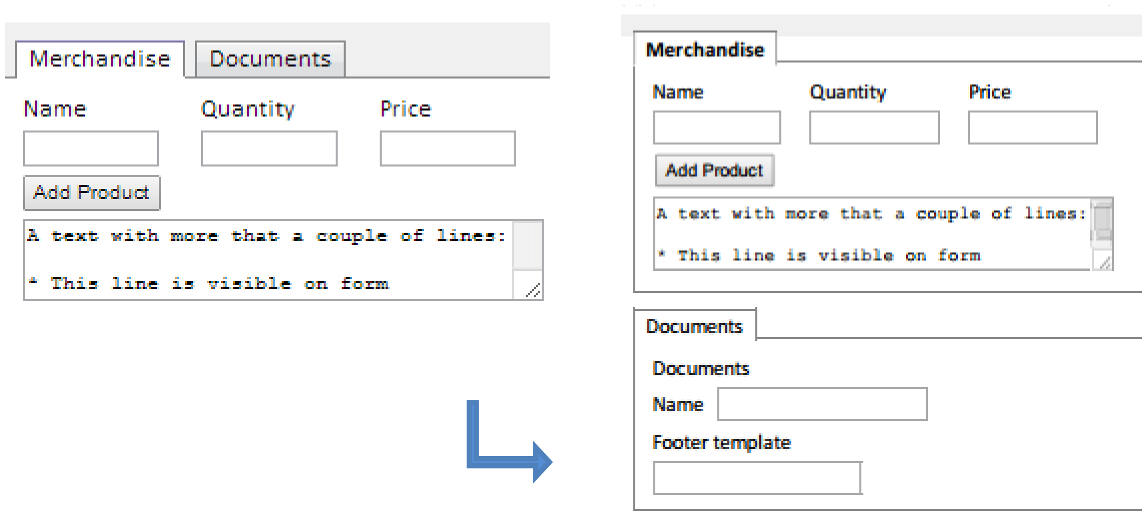
And place it in the `inputChanged()` event function.



Edit form scripts

```
<script type='text/javascript'>

  function inputChanged(source) {
    if (!source) {
      ELO.Configuration.PrintAllTabs = true;
    }
  }
```

Figure 1: JavaScript code: "inputChanged" function



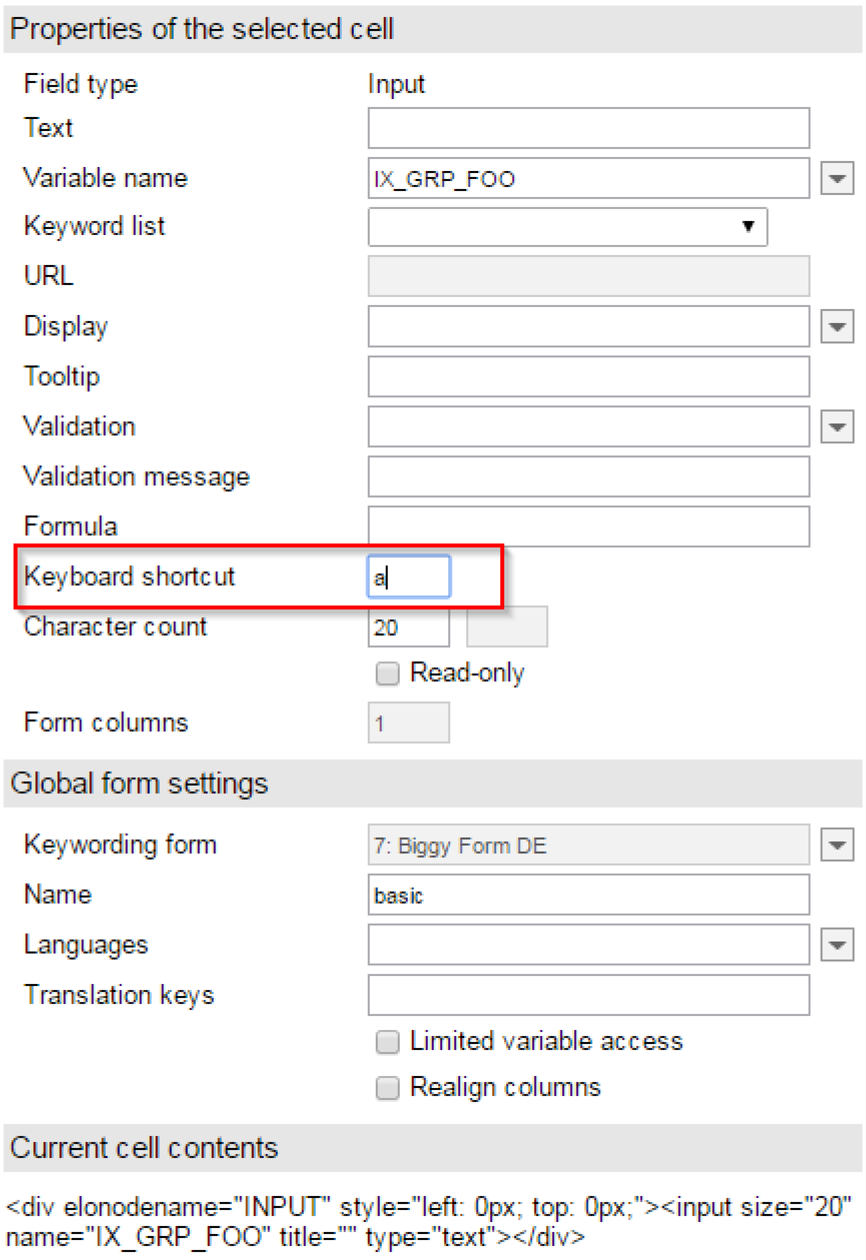
```
A text with more that a couple of lines:

* This line is visible on form
```

10.2 Keyboard shortcuts

Please note: These features have limited support depending on the client and browser used.

Keyboard shortcuts work when the form is opened directly or in the ELO Web Client in Chrome or Internet Explorer. In Firefox, the ALT key opens the browser menu instead, and the Java client has no support for this feature altogether. Support for other combinations of browsers/clients is unspecified.

It is possible to provide keyboard shortcuts to set the focus directly in a specific field:



Properties of the selected cell

Field type	Input
Text	
Variable name	IX_GRP_FOO
Keyword list	
URL	
Display	
Tooltip	
Validation	
Validation message	
Formula	
Keyboard shortcut	a
Character count	20
	☐ Read-only
Form columns	1

Global form settings

Keywording form	7: Biggy Form DE
Name	basic
Languages	
Translation keys	
	☐ Limited variable access
	☐ Realign columns

Current cell contents

```
<div elonodename="INPUT" style="left: 0px; top: 0px;"><input size="20"
name="IX_GRP_FOO" title="" type="text"></div>
```

When the user presses *ALT+a*, the focus will jump to the corresponding field.