ELOwf Entwicklerleitfaden

[Stand: 07.02.2020 | Programmversion: 12.00.000]

Inhalt

1 Übersetzung / Lokalisierung ... 3
1.1 Übersetzungstabelle verwenden ... 3
1.2 Properties-Dateien verwenden ... 4
1.3 Vergleich ... 6
1.4 Übersetzungen bearbeiten ... 7
1.5 Dynamische Sprachauswahl ... 7
2 Datumsfelder ... 9
2.1 Das Format ... 9
2.2 Internationalisierung & Zeitzonen ... 9
3 Numerische Felder und Betragsfelder ... 10
3.1 Der Unterschied ... 10
3.2 Verhalten ... 10
3.3 Format ... 11
3.4 Formeln ... 11
3.5 Spontane Formeln ... 12
3.6 Skripting ... 12
3.7 Runden ... 12
4 Linkfelder ... 13
5 Zeichnungen ... 14
6 Validierung ... 16
6.1 Einleitung ... 16
6.2 Grundverhalten ... 16
6.3 Eigene Validierungsnachrichten ... 17

6.4 Eigene Validierungsfunktionen (JS_VAL_...) ... 18
6.5 Eigene Filterfunktionen (JS_FILTER_...) ... 19
6.6 Kann ich je nach Anwendungsfall die Validierung ein- oder ausschalten? ... 20
7 Listen ... 21
7.1 Übersicht ... 21
7.2 Dynamische Stichwortliste ... 21
7.3 Dynamische Stichwort-Map ... 26
8 Dynamische Templates ... 29
8.1 JS_AddLine ... 29
9 Eigene Dateien hinzufügen ... 30
9.1 Bilder ... 30
9.2 JavaScript-Dateien ... 31
9.3 CSS-Dateien ... 32
10 Ereignisse ... 33
10.1 onInit() ... 33
10.2 inputChanged(elem) ... 33
10.3 saveClicked() ... 33
10.4 nextClicked(id) ... 34
11 Verschiedenes ... 35
11.1 Prüfung beim Speichern und Weiterleiten ... 35
11.2 Drucken ... 35
11.2.1 Textbereiche automatisch erweitern ... 35
11.3 Einstellbare Tastaturbefehle ... 37
11.4 Betrieb mehrerer ELOwf Instanzen ... 38

1 Übersetzung / Lokalisierung

Es stehen mehrere Möglichkeiten zur Verfügung, um Workflow-Formulare zu lokalisieren bzw. internationalisieren, damit Sprachtexte und Tooltipps automatisch in der Clientsprache übersetzt werden.

1.1 Übersetzungstabelle verwenden

Die „klassische" Vorgehensweise, um Texte zu internationalisieren, ist es, Sprachtexte und Tooltipps in dem Volltext zu speichern. Damit werden die Texte anhand der Übersetzungstabellen, die in den Clients verfügbar sind, automatisch ersetzt.

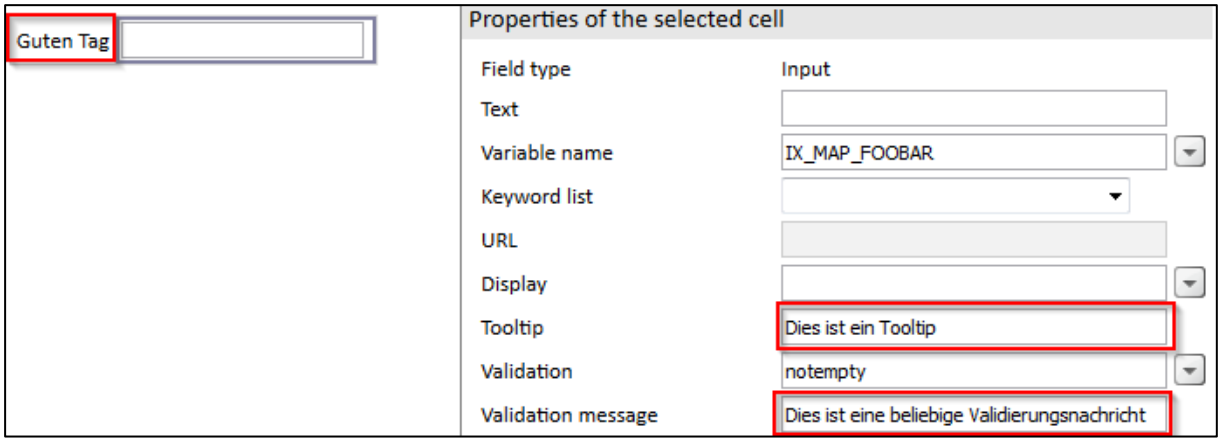


Abb. 1: Beispiel des Einsatzes der Übersetzungstabelle im Formulardesigner

Die Übersetzungstabelle würde in etwa so aussehen:



Abb. 2: Beispiel für eine Übersetzungstabelle

> **Information:** Die vorangestellte „z" in dem Screenshot oben wurde nur hinzugefügt, um die Texte für die Anzeige zu sortieren.

Das Ergebnis in einem englischsprachigen Client wäre also:

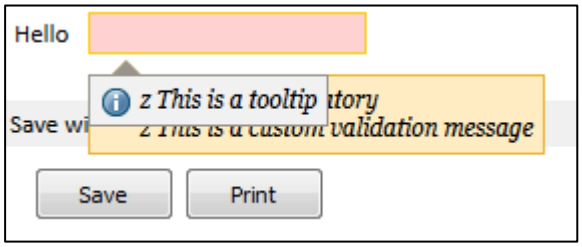


Abb. 3: Beispiel für einen Tooltip

Der Screenshot ist etwas überfüllt, aber es sollte Ihnen die zugrundeliegende Idee vermitteln. Falls keine Übersetzungen für einen bestimmten Text gefunden werden, wird das Original behalten.

Wie Sie in diesem Beispiel erkennen können, ist die Referenzsprache für ELOwf nicht unbedingt die Systemsprache. Die Standardsprache wird durch den Eintrag *mainlanguage* im Zusatztext der Verschlagwortung des ELOwf Ordners *Configuration* definiert.

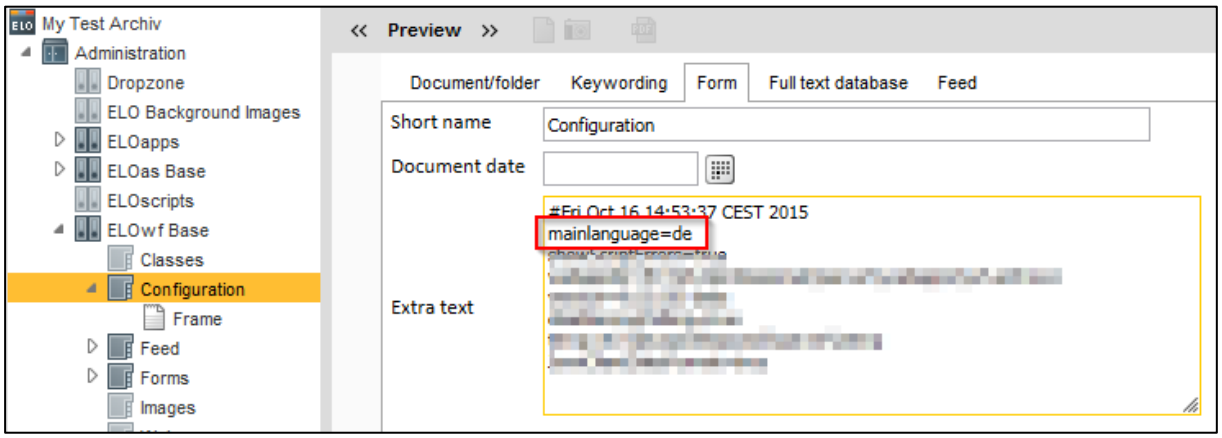


Abb. 4: ELOwf-Ordner 'Configuration'

Diese Sprache wird bei der Suche nach Übersetzungen als „Referenz“ verwendet.

1.2 Properties-Dateien verwenden

Die „neue“ Vorgehensweise, um Texte zu lokalisieren, ist es, Übersetzungsvariablen in dem Formular zu verwenden, und dann -Dateien zu verwenden, um die Übersetzungen auszuliefern. Damit können Sie auch maßgeschneidert übersetzte Hinweise per Skripting erstellen, was nicht auf andere Weise möglich wäre.

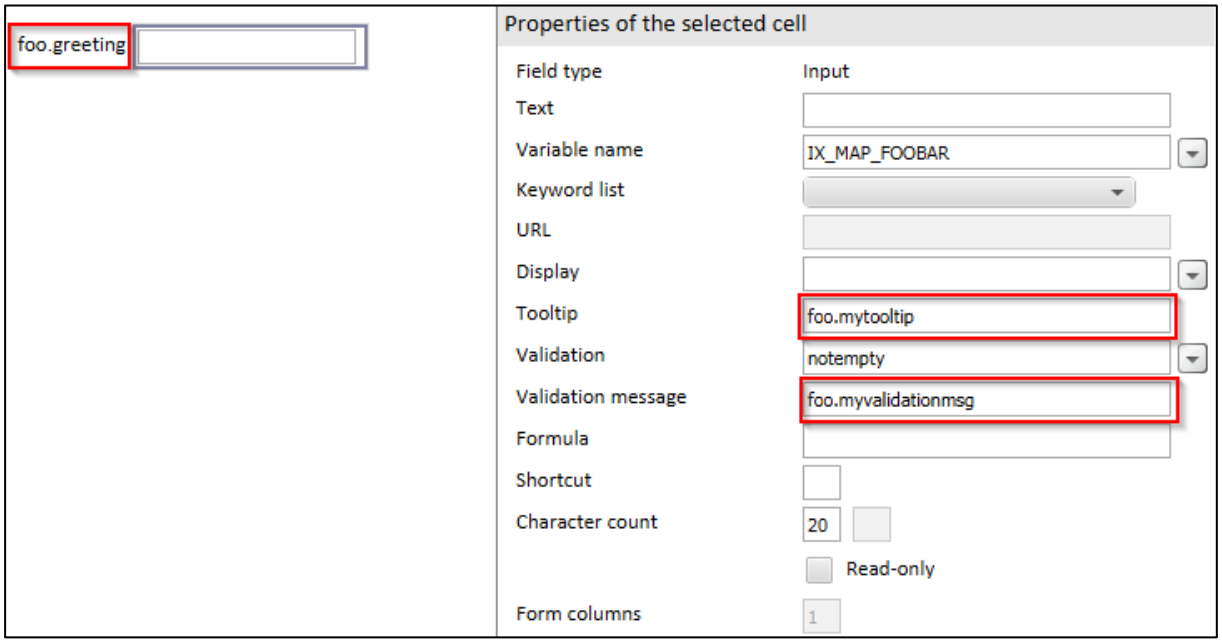


Abb. 5: Unterschiedliche Einsatzmöglichkeiten von Übersetzungsvariablen

Die Properties-Dateien (Dateiendung: *.properties*) können über den Pfad:

Administration //Localization // system abgelegt werden.

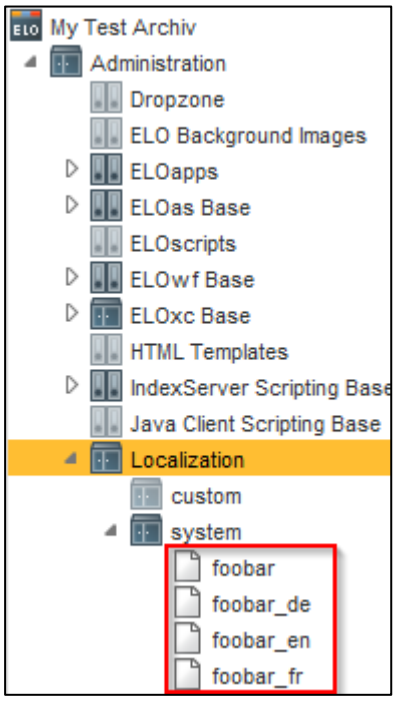


Abb. 6: Dokumentenpfad der 'property'-Dateien

Jede dieser vier Dateien ist eine Properties-Datei für die jeweilige Sprache. Die Datei *foobar* ohne ein nachgestelltes Sprachkürzel wird in diesem Fall per Default benutzt, falls eine unerwartete/unbekannte Sprache nachgefragt wird.

Information: Sie können jeden beliebigen Dokumentennamen (Sprachkürzel davon ausgenommen) für die Sprachdateien verwenden. Alle Variablen werden zur Laufzeit in den jeweiligen Speicher für die Sprache geladen.

Der Inhalt von *foobar_en.properties* könnte so aussehen:

foobar_en.properties

```
1  foo.greeting=Hello
2  foo.mytooltip=This is a tooltip
3  foo.myvalidationmsg=This is a validation message
4
5  bar.mynotification=That can be called inside a script!
```

Abb. 7: Inhalt einer Properties-Datei

Sie können auch eine Teilmenge von diesen Properties importieren, um sie für Skripte zur Verfügung zu stellen. Um das zu ermöglichen, geben Sie das Präfix für die Übersetzungsvariable in der unten angezeigten Eigenschaft ein. In diesem Beispiel werden alle Variablen, die mit „bar" beginnen, für das Formular verfügbar gemacht.

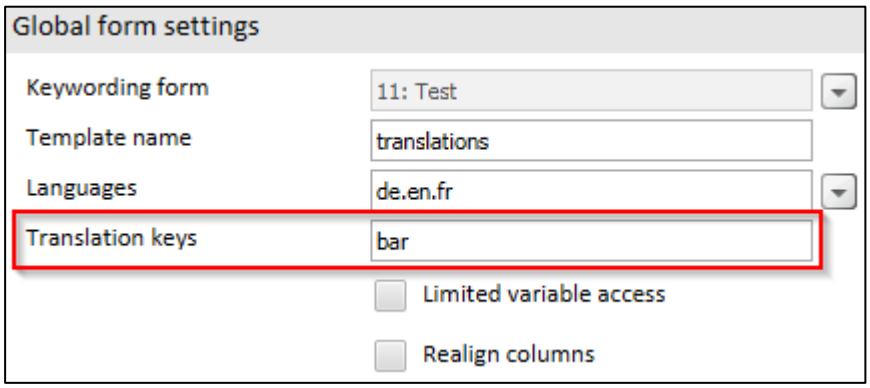


Abb. 8: Verwendung einer Übersetzungsvariable im Formulardesigner

In Skripten können die Texte dann über die Methoden von `api.helpers.Text` abgerufen werden. In diesem Fall können Sie eine MessageBox mit einer übersetzten Nachricht einfach so erstellen:

```
eloAlert(api.helpers.Text.getText('bar.mynotification'))
```

1.3 Vergleich

Als Faustregel gilt: Sie können Übersetzungstabellen verwenden, wenn Sie nur ein paar Formulare in einer Sprache übersetzen müssen. Wenn Sie mehrere größere Formulare haben, oder in mehreren Sprachen übersetzen müssen, wird die auf Properties basierte Lösung empfohlen.

Die Nutzung von Properties-Dateien hat die folgenden Vorteile:

- Skripte können auch auf übersetzte Texte zugreifen, zum Beispiel auf Hinweiskästen.
- Je nach Kontext muss manchmal ein Wort in einer Sprache in unterschiedliche Wörter einer anderen Sprache übersetzt werden. Dies ist nicht mit Übersetzungstabellen möglich.
- Properties-Dateien werden von professioneller Übersetzungssoftware unterstützt.
- Properties-Dateien werden in dem ELO Archiv versioniert.

Das mag in kleinen Archiven mit niedrigen Anforderungen für die Lokalisierung überflüssig sein, aber der Nutzen steigt zunehmend, wenn diese Anforderungen zunehmen.

1.4 Übersetzungen bearbeiten

Aus Performancegründen speichert ELOwf Übersetzungen in einem Cache. Nach der Bearbeitung der Übersetzungstabellen oder Properties-Dateien, laden Sie das Modul ELOwf neu, damit die Änderungen übernommen werden. Führen Sie nach der Änderung der Properties-Dateien außerdem einen Neustart des ELO Indexservers durch.

1.5 Dynamische Sprachauswahl

Bei der Eingabe von Sprachen in den Formulareinstellungen im Feld *Languages* (siehe Bild) gilt:

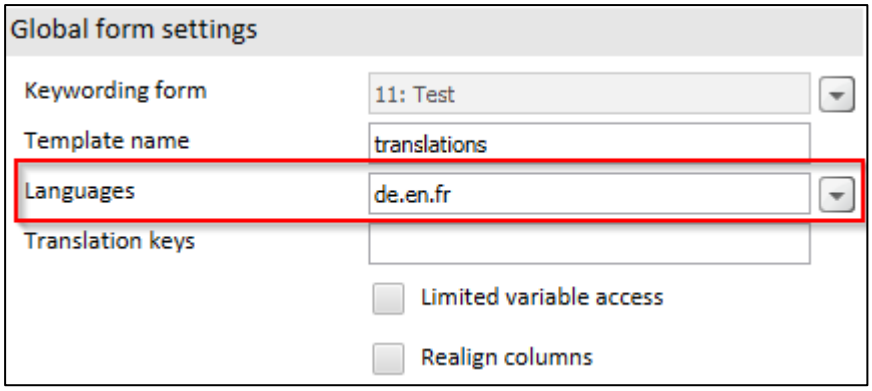


Abb. 9: Einstellungen für die Sprachen im Formulardesigner

Ein Kombinationsfeld wird dem Formular hinzufügt, mit dem man innerhalb des Formulars selbst die Sprache wechseln kann.

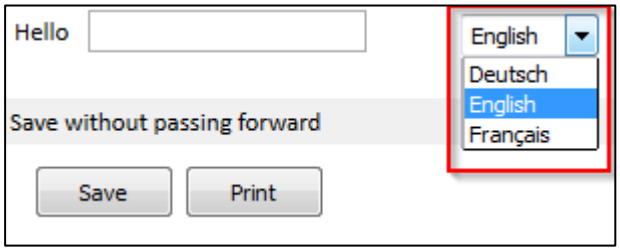


Abb. 10: Kombinationsfeld in einem Formular

2 Datumsfelder

Obwohl es nicht auf dem ersten Blick ersichtlich ist, kann jedes Datumsfeld zusätzliche Informationen zur Uhrzeit beinhalten. Sie können zum Beispiel „12. November 2015“ im Kalender auswählen, aber Sie können auch eine Uhrzeit eingeben, wie beispielsweise „12. November 2015 07:08 Uhr“. Sekunden werden nicht gespeichert.

2.1 Das Format

- Bei der Verwendung des ELO Java Clients haben die Formulare das gleiche Format. Dieses Format hat entweder ein bestimmtes Muster oder ist Locale-abhängig (in der Standardeinstellung).
- In anderen Fällen benutzt das Formular entweder das deutsche Format („dd.MM.yyyy HH:mm“) - es sei denn, die Sprache ist Ungarisch, in diesem Fall wird („yyyy.MM.dd HH:mm“) benutzt.

Falls Sie nicht den ELO Java Client verwenden, können Sie andere Datums-/Zeitformate nicht verwenden.

Es empfiehlt sich, für das Datumsfeld beim Schreiben, Einfügen oder Aktualisieren das gleiche Format zu verwenden. Es ist allerdings möglich, auf die folgenden Alternativen auszuweichen:

- Das Alternativformat, das im ELO Java Client angegeben wurde
- Das universelle Format ISO 8601 (z. B. „2014-09-08T08:02:17“)

2.2 Internationalisierung & Zeitzonen

Bei Anmeldung mit einem Client in einer anderen Zeitzone, werden *IX_DOCDATE* und *IX_CREATEDATE* für die örtliche Zeitzone angepasst.

Beispiel: Ein Dokument mit einem Erstellungsdatum vom „12. November 2015 07:08“ wird zu „12. November 2015 10:08“, falls ein Benutzer sich an einem Client in einer drei Stunden späteren Zeitzone anmeldet. Dies ist auch der Fall bei einem „puren“ Datum, hier wird die Zeit als 0 Uhr interpretiert. „12. November 2015“ wird zum Beispiel gleich wie „12. November 2015 um 00:00“ behandelt.

Alle anderen Datumswerte in dem Formular bleiben unverändert.

3 Numerische Felder und Betragsfelder

3.1 Der Unterschied

Der Hauptunterschied zwischen Betragsfeldern und numerischen Felder besteht darin, dass Betragsfelder Trennzeichen verwenden, währenddessen num-Felder diese nicht haben. Betragsfelder werden typischerweise für große Zahlen oder Währungen eingesetzt. Die Tausender- und Dezimaltrennzeichen sind auch sprachabhängig. Ein Betragsfeld wie „123.456,78“ auf Deutsch wird also als „123,456.78“ im Englischen dargestellt.

3.2 Verhalten

„amount nk:2“ verhält sich folgendermaßen (in der Client-Spracheinstellung Deutsch):

- „12346” -> „12.346,00” <= stellt ein Tausendertrennzeichen dar
- „12.346“ -> „12.346,00“ <= interpretiert den Punkt als Tausendertrennzeichen
- „12,346“ -> „12,35“

„num nk:2“ verhält sich folgendermaßen (in der Client-Spracheinstellung Deutsch):

- „12346“ -> „12346,00“ <= Kein Tausendertrennzeichen
- „12.346" -> „12,35“ <= interpretiert sowohl Komma als auch Punkt als Dezimaltrennzeichen
- „12,346“ -> „12,35“

Betragsfelder können Eingaben mit oder ohne Tausendertrennzeichen akzeptieren, werden aber die Daten immer mit Trennzeichen anzeigen.

Das folgende Flag kann im Header-Skript gesetzt werden:

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

Ist es gesetzt, dann verhalten sich amount-Felder ähnlich wie num-Felder, die das letzte Trennzeichen als ein Dezimaltrennzeichen interpretieren, egal, ob es sich um einen Punkt oder ein Komma handelt. Dies ist vor allem wichtig, falls die Daten die vom Server geliefert werden (z. B: Map-Felder) keine einheitlichen Trennsymbole verwenden, sondern dort durch verschiedene Importprozesse Zahlen sowohl mit Punkt als auch mit Komma in der Datenbank stehen.

3.3 Format

Was wird für die Tausender-/Dezimaltrennzeichen benutzt?

- Wenn der Benutzer den ELO Java Client einsetzt, wird es in den technischen Voreinstellungen der Konfiguration eingestellt. Dort haben Sie zwei Optionen:
  - **Standard:** (empfehlenswert, ist sprachabhängig).
  - **Benutzerdefiniert**: Alternativ dazu können Sie eigene Trennzeichen angeben (nicht empfehlenswert, weil das Trennzeichen immer gleich bleibt, auch wenn Sie die Sprache ändern).
- In anderen Fällen wird der Standard der Client-Spracheinstellung angewandt.

Falls Sie sich nicht sicher sind, welches Trennzeichen benutzt wird, können Sie die Einstellung in dem Formular-Header ermitteln. Diese sind in den folgenden Variablen gespeichert:

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

3.4 Formeln

Numerische Eingabefelder können über beliebige Formeln berechnet werden.

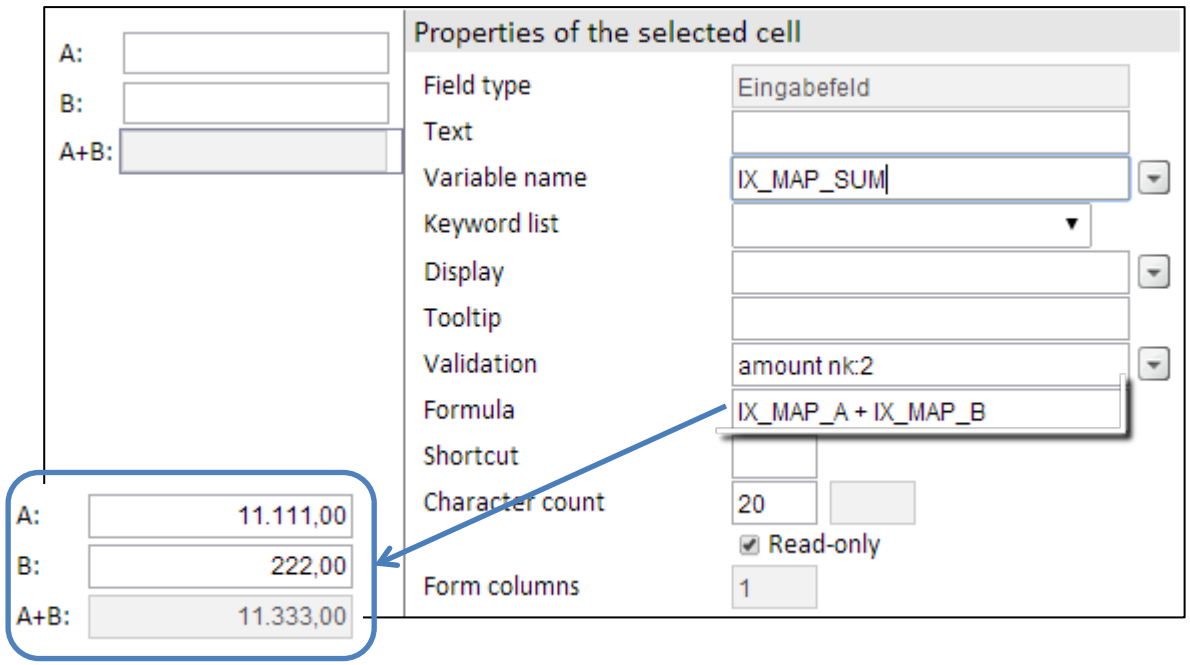


Abb. 11: Berechnung von numerischen Eingabefeldern

Die Sonderfunktion ESum(IX_MAP_N) als Formel kann benutzt werden, um alle Werte zu summieren. Es ist dasselbe wie IX_MAP_N1 + IX_MAP_N2 + ...

3.5 Spontane Formeln

Sie können in Formulare „=“ eingeben, gefolgt von einer mathematischen Formel wie „= 0.21 * 1234 + 57“, um etwas dynamisch zu berechnen. Drücken Sie danach einfach die Eingabetaste oder verlassen Sie das Feld; das Ergebnis wird automatisch berechnet.

Diese Inline-Formeln sind auf numerische Operationen beschränkt. Andere Variablen stehen nicht zur Verfügung.

3.6 Skripting



Information: Wenn Sie Werte aus Feldern über Skripte holen, verwenden Sie in jedem Fall die Funktion *$num(...)* und nicht die Funktion *$val(...).* Die erste Funktion gibt ein Float zurück, wobei die zweite Funktion einen „ELO-formatierten“ String zurückgibt: Das heißt, kein Tausendertrennzeichen und ein Komma als Dezimaltrennzeichen, egal welche Sprache angegeben wird!

Enthält die Eingabe den Wert „12.345,67“, gibt $val(„IX_...“) den String „12345,67“ zurück.

Numerische Werte und Betragswerte werden ebenfalls immer in *ELO_PARAMS* als ELO-formatierte Strings gespeichert.

3.7 Runden

Das Formular wird innerhalb von JavaScript ohne Kontextwissen ausgeführt. Somit kann es nicht wissen, welcher Sinn hinter einem Zahlenfeld steht. Wird ein Zahlenfeld mit mehr Nachkommastellen als definiert befüllt, kann es somit kein korrektes Verhalten in allen Fällen geben.

Wer hier kritische Prozesse abbildet sollte unbedingt auf die Definition einer Formel verzichten und eine geeignete Strategie zum Runden im Event *inputChanged* selbst vornehmen.

4 Linkfelder

Seit ELO 10 können Sie auch Links in Formularen einfügen.

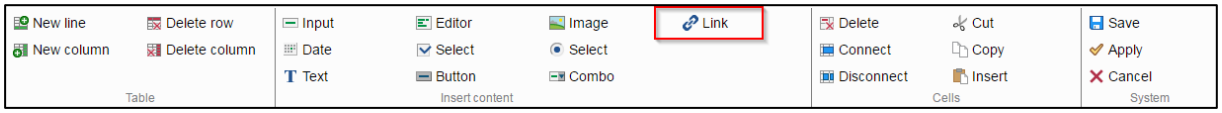


Abb. 12: Schaltfläche 'Link' im Formulardesigner

Es stehen zwei Arten von Links zur Verfügung:

- Ist die URL eine GUID, wird beim Klick das Dokument in einem neuen Fenster/Tab geöffnet.
- Ist die URL eine gewöhnliche Internetadresse, wird die zugehörige Webseite in einem neuen Fenster/Tab geladen.

Information: Wird die URL in dem Formulardesigner leer gelassen, zeigt das Formular einen Link auf das aktuelle Dokument an.

5 Zeichnungen

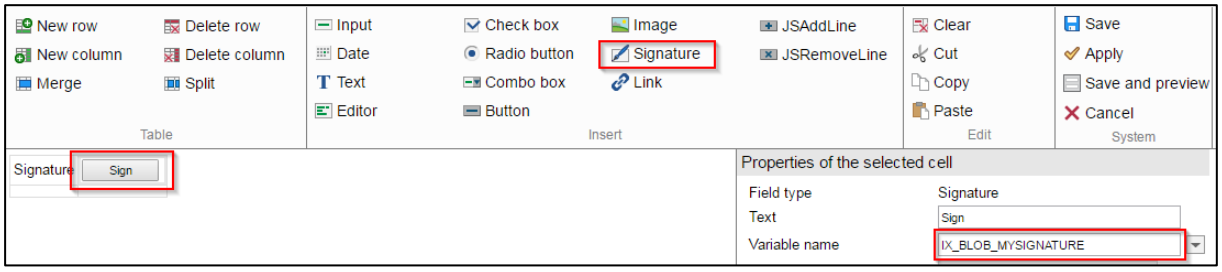


Abb. 13: Signatur-Widget im Formulardesigner

Das Signatur-Widget ist in ELOwf 10.01 und späteren Versionen verfügbar. Das Widget ermöglicht es dem Benutzer, ein Formular zu unterschreiben.

Da die Daten, die in einer Signatur gespeichert werden, größer sind als in den Indexfeldern und Map-Felder erlaubt, müssen „BLOB“ (Binary Large OBject)-Felder eingesetzt werden. Diese werden anhand von Variablen erkannt, die mit „IX_BLOB_“ beginnen.

> **Beachten Sie:** Für Signaturen müssen die „IX_BLOB_“-Variablennamen zusätzlich das Präfix „WO_“ tragen. Dieser Zusatz steht für „Write Once“ und ist notwendig, damit der ELOwf nach einmaligem Speichern der Unterschrift den Wert nicht überschreibt. Die Variablen aller Signatur-Felder sollten also mit „IX_BLOB_WO_“ beginnen.

In der Map wird dann als key "WO_<Name>" verwendet.

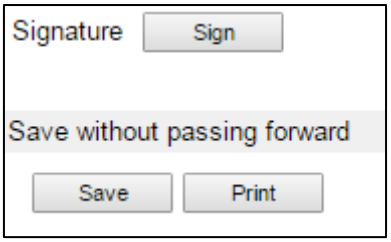


Abb. 14: Optionen im Signatur-Widget

Zur Weiterverarbeitung können die Daten aus der Map *formdata* ausgelesen werden. Das Format ist ein serialisiertes JSON-Objekt mit folgendem Inhalt:

- **timestamp**: Technischer Zeitstempel wann die Signatur erstellt wurde.
- **user**: Verbundener ELO Benutzer, der das Formular aufgerufen hat.
- **signature**: Bilddaten der Signatur als Data-URL.



Abb. 15: Eingabefeld im Signatur-Widget



Abb. 16: Beispiel einer Signatur im Widget

Es ist möglich, mehrere Signaturen auf einem Formular anzubringen, die vom gleichen Benutzer oder unterschiedlichen Benutzern unterzeichnet sind. Die einzige Beschränkung hierbei ist, dass dieses Widget nicht in Kombination mit dynamischen „JS_ADDLINE"-Zeilen verwendet werden kann.

6 Validierung

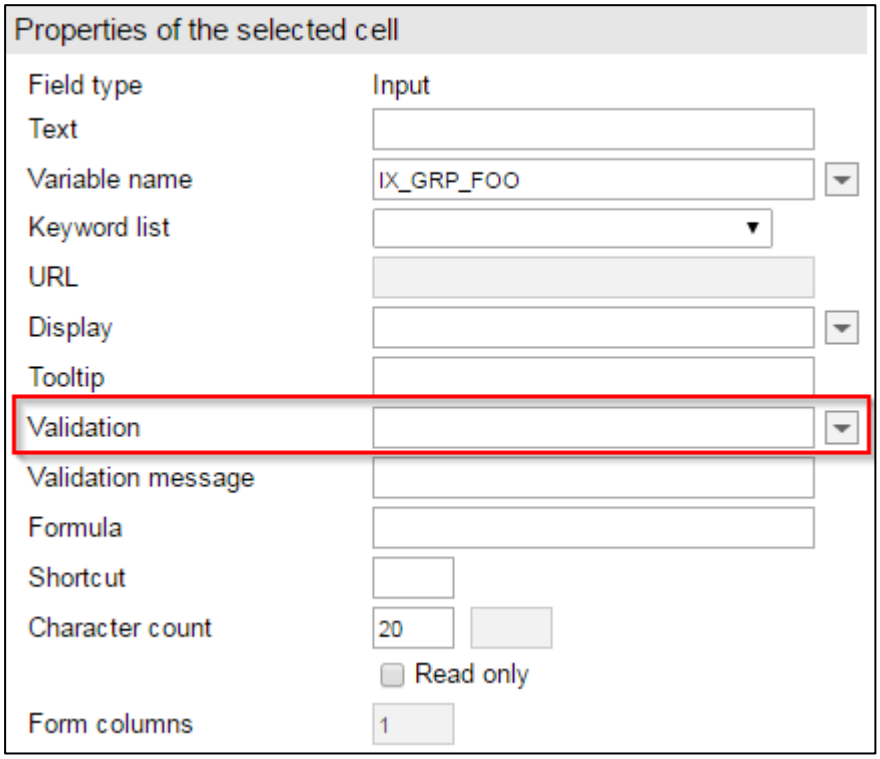


Abb. 17: Formularfeld 'Validation' im Formulardesigner

6.1 Einleitung

Die Einschränkungen der Indexfelder auf einer Verschlagwortungsmaske und die Validierung in den ELOwf Formularfeldern sind im Grunde zwei ganz verschiedene Funktionalitäten.

Die Einschränkungen, die in der Verschlagwortungsmaske angegeben sind, werden von ELOwf und ELO Indexserver ignoriert (*). Sie können per Default alles Mögliche in jedem Indexfeld speichern, auch wenn es als eine Nummer, Datum, oder alles andere definiert wurde.

Die Verschlagwortungsmasken werden ausschließlich in den Clients (Java, Web, Windows) verwendet, um die Inhalte zu validieren/formatieren, jedoch aber *nicht* von ELOwf, der eine andere Validierung einsetzt.

(*) **Ausnahme**: Die Minimal- und die Maximallänge sind die zwei einzigen Eigenschaften einer Verschlagwortungsmaske, die automatisch auch im Formular erzwungen werden.

6.2 Grundverhalten

Sie können pro Feld eine Reihe vordefinierter Einschränkungen angeben:

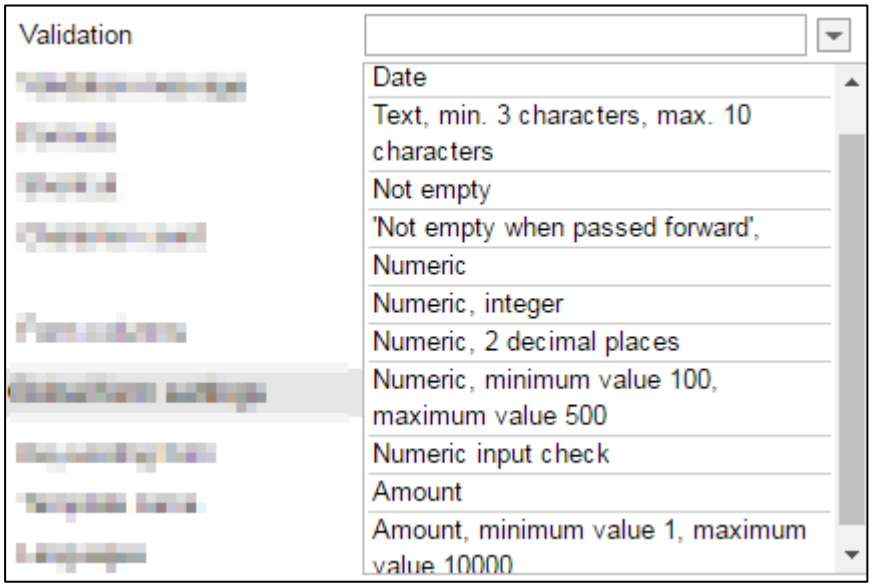


Abb. 18: Einschränkungen im Formularfeld 'Validation'

Um konsistent zu bleiben, sollen diese Einschränkungen gleich sein wie die in der Verschlagwortungsmaske – aber erforderlich ist das nicht.

Ein paar weitere Hinweise:

- Für numerische Werte bestimmt Min./Max. in der Verschlagwortungsmaske die Anzahl der Stellen, wobei diese Einstellung in dem Formulardesigner den Minimal-/Maximalwert bestimmt.
- Die Validierung *Nicht leer wenn weitergeleitet* wurde geschaffen, damit das Formular mit leeren Feldern "temporär" gespeichert werden kann, die zwingend gefüllt werden müssen, wenn das Formular innerhalb eines Workflows weitergeleitet wird.

6.3 Eigene Validierungsnachrichten

Zusätzlich zu den Standard-Validierungsnachrichten können Sie eigene Nachrichten schreiben.

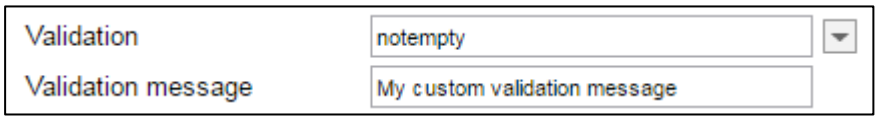


Abb. 19: Beispiel für eine eigene Nachricht im Feld 'Validation Message'

Wird zu:

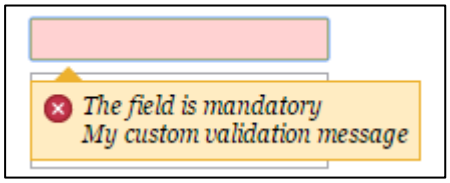


Abb. 20: Ausgabe der im Feld 'Validation message' definierten eigenen Nachricht

Die Nachricht wird zusätzlich zu den anderen Fehlermeldungen angezeigt. Es ist auch möglich, Übersetzungsvariablen für die Meldung zu verwenden.

6.4 Eigene Validierungsfunktionen (JS_VAL_...)

Angepasste Funktionen können verwendet werden, um Eingaben zu validieren.

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

Für ein ungültiges Feld sollte die Funktion den Fehler als nicht-leeren String ausgeben.

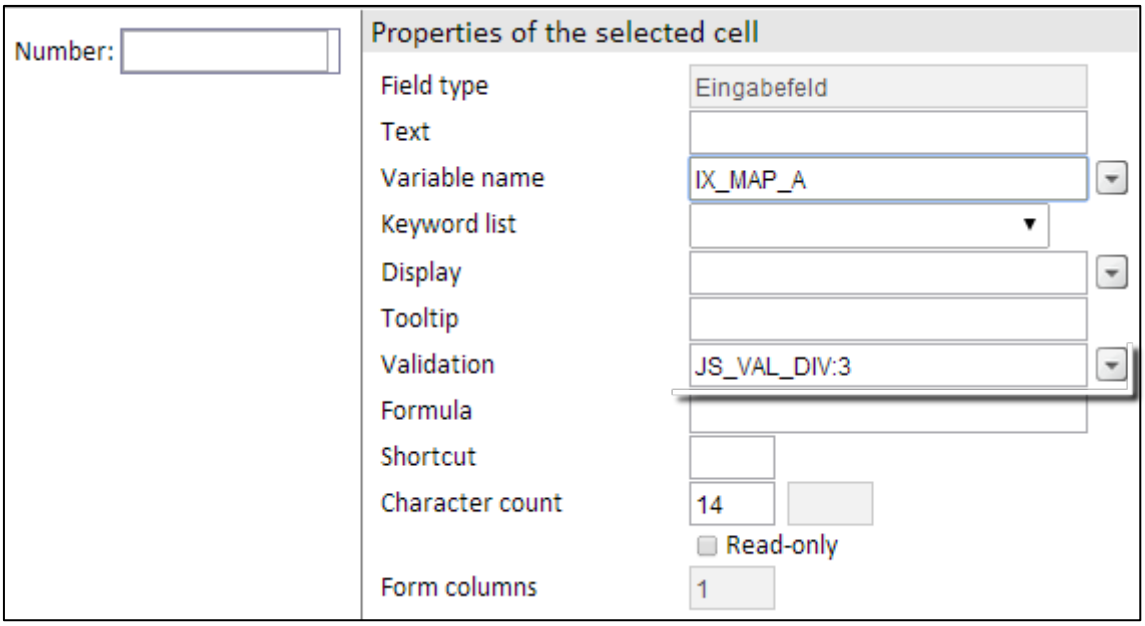


Abb. 21: Eigene Validierungsfunktionen im Feld 'Validation'

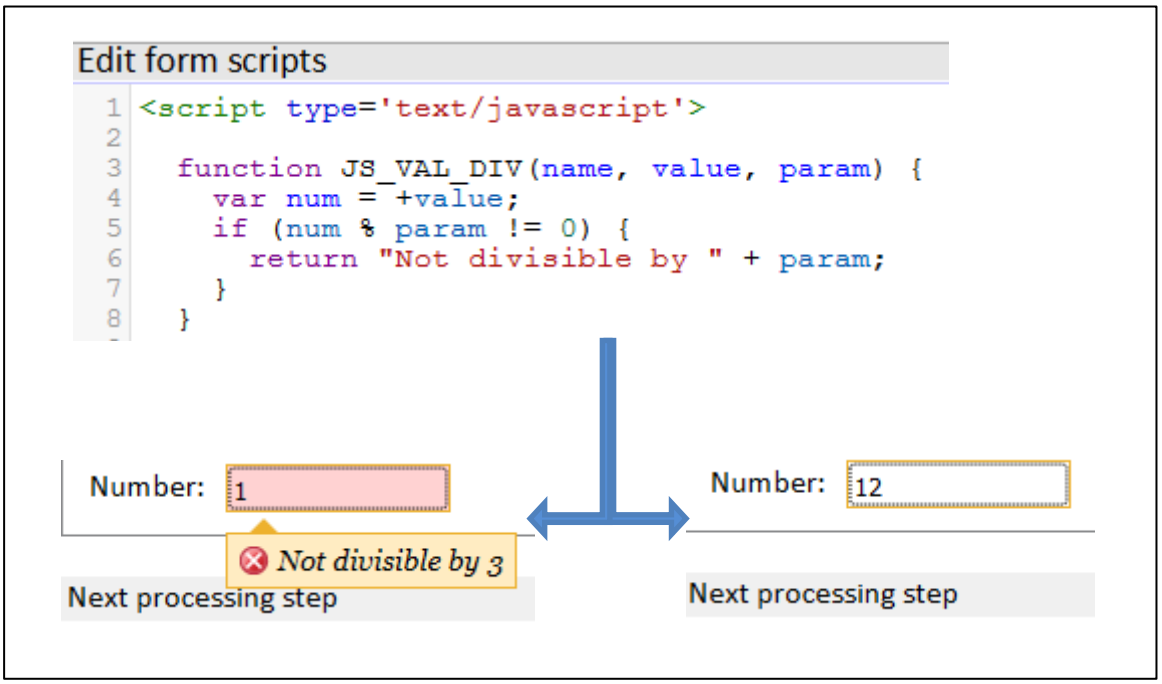


Abb. 22: Eigene Validierungsfunktionen im Skript-Editor

Das kann zusätzlich zu anderen Validierungseinschränkungen vorgenommen werden. Diese Funktionalität ist nur dadurch eingeschränkt, indem es synchron sein muss.

6.5 Eigene Filterfunktionen (JS_FILTER_...)

Gibt der Benutzer etwas in einem Feld ein, kann ein Filter eingesetzt werden, um den Wert umzuwandeln.

Die vordefinierte Funktion *JS_FILTER_NUMBER* beispielsweise erlaubt es dem Benutzer, nur Ziffern einzugeben.

Eigene Filter können eingesetzt werden, um nur Zahlen zu erlauben, die Eingaben in Großbuchstaben umzuwandeln, ein bestimmtes Format zu erzwingen oder jede beliebige andere Art von Texttransformation zu übernehmen.

Um das zu machen, geben Sie „`JS_FILTER_MeineFunktion`“ oder „`JS_FILTER_MeineFunktion:Parameter`“ im Feld *Validierung* ein. Die folgende Funktion wird aufgerufen, wenn ein Benutzer etwas in dem Feld eingibt oder einfügt.

```
JS_FILTER_MyFunc(vorne, eingegeben, danach, param)
```

Hier ist ein Beispiel, wie Sie die Eingaben in Großbuchstaben umwandeln können:

```
function JS_FILTER_Uppercase(vorne, eingegeben, danach) {
    return eingegeben.toUpperCase();
}
```

In diesem Fall wird der Rückgabewert ein String, der die Eingaben in modifizierter Form darstellt. Es ist allerdings auch möglich, die Angaben vor und nach der Eingabe zu modifizieren. Sie erreichen das, indem sie ein Array mit den neuen Werten zurückgeben ([vorne, eingegeben, danach]).

6.6 Kann ich je nach Anwendungsfall die Validierung ein- oder ausschalten?

Nein. Die Validierung findet immer statt. Es ist auch sinnvoll. Auch wenn Sie einen Workflow mit einer „Abbrechen"-Operation vorliegen haben, muss das Formular immerhin gespeichert werden. Und um speichern zu können, muss es gültige Daten beinhalten.

Anstatt die Validierung zu entfernen, empfehlen wir die Anwendung von „nextClicked(...)", um einfach Werte ggs. mit Dummy-Werten zu füllen, bevor die Validierung stattfindet.

7 Listen

7.1 Übersicht

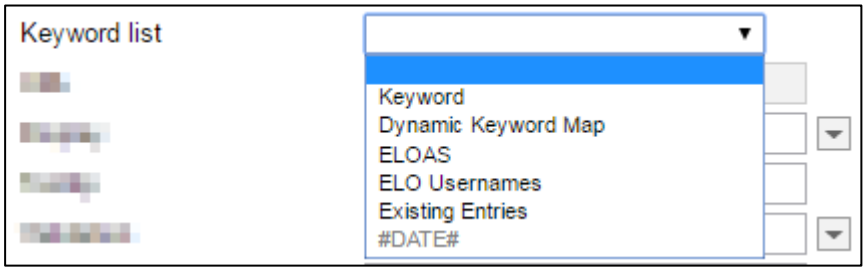


Abb. 23: Kombinationsfeld 'Keyword List'

Eine Erklärung für diese Listentypen:

- **Stichwort**: Das kann eine statische oder dynamische Stichwortliste sein, je nachdem, was für das Feld in der Verschlagwortungsmaske definiert wurde.
- **Dynamische Stichwortliste**: Diese Liste wird typischerweise benutzt, um Werte für Map-Felder zu holen.
- **ELOAS**: Ruft ein ELOas Ruleset auf und gibt die Ergebnisse als eine Liste zurück.
- **ELO Benutzernamen**: Ruft eine Liste von Benutzern und Gruppen ab.
- **Bestehende Einträge**: Wählen Sie diese Option aus, erhält der Benutzer eine Autovervollständigungs-Liste, die aus den vorher in dem Feld eingetragenen Werten in anderen Dokumenten basiert.

7.2 Dynamische Stichwortliste

Ein dynamisches Stichwortfeld ist ein Feld, das mehrere Spalten Inhalt darstellen kann, typischerweise dynamisch gefiltert nach einem oder mehreren anderen Feldwerten. Es sieht folgendermaßen aus:

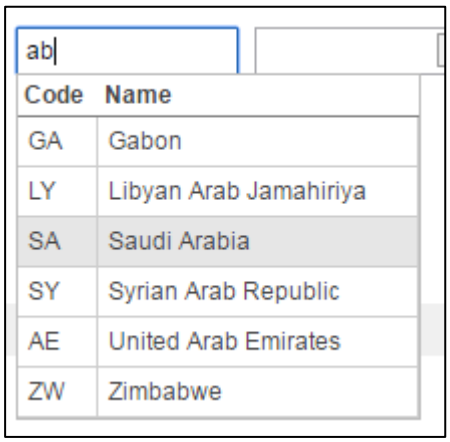


Abb. 24: Beispiel für ein dynamisches Stichwortfeld

Dynamische Stichwortlisten sind ein relativ komplexes Feature in ELO, das für die Funktionalität Skripting-Kenntnisse voraussetzt. Sie sind im ELO Java Client und dem ELO Web Client verfügbar.

Sie können Folgendes damit tun:

- Mehrere Spalten von Daten anzeigen
- Inhalte dynamisch abrufen (zum Beispiel: externe Datenbanken, dynamisch berechnete Daten, ELO Skripte...)
- Daten können gefiltert werden, nicht nur in einem Feld, sondern auch abhängig von beliebig anderen Feldern (zum Beispiel: was der Benutzer eingegeben hat und eine Kategorie, die der Benutzer zuvor ausgewählt hat)
- Füllen weiterer anderer Felder mit Eintragsauswahl (inklusive der Nur-Lesen-Felder)

Die Verwendung einer dynamischen Stichwortliste wird direkt in der Verschlagwortungsmaske definiert:

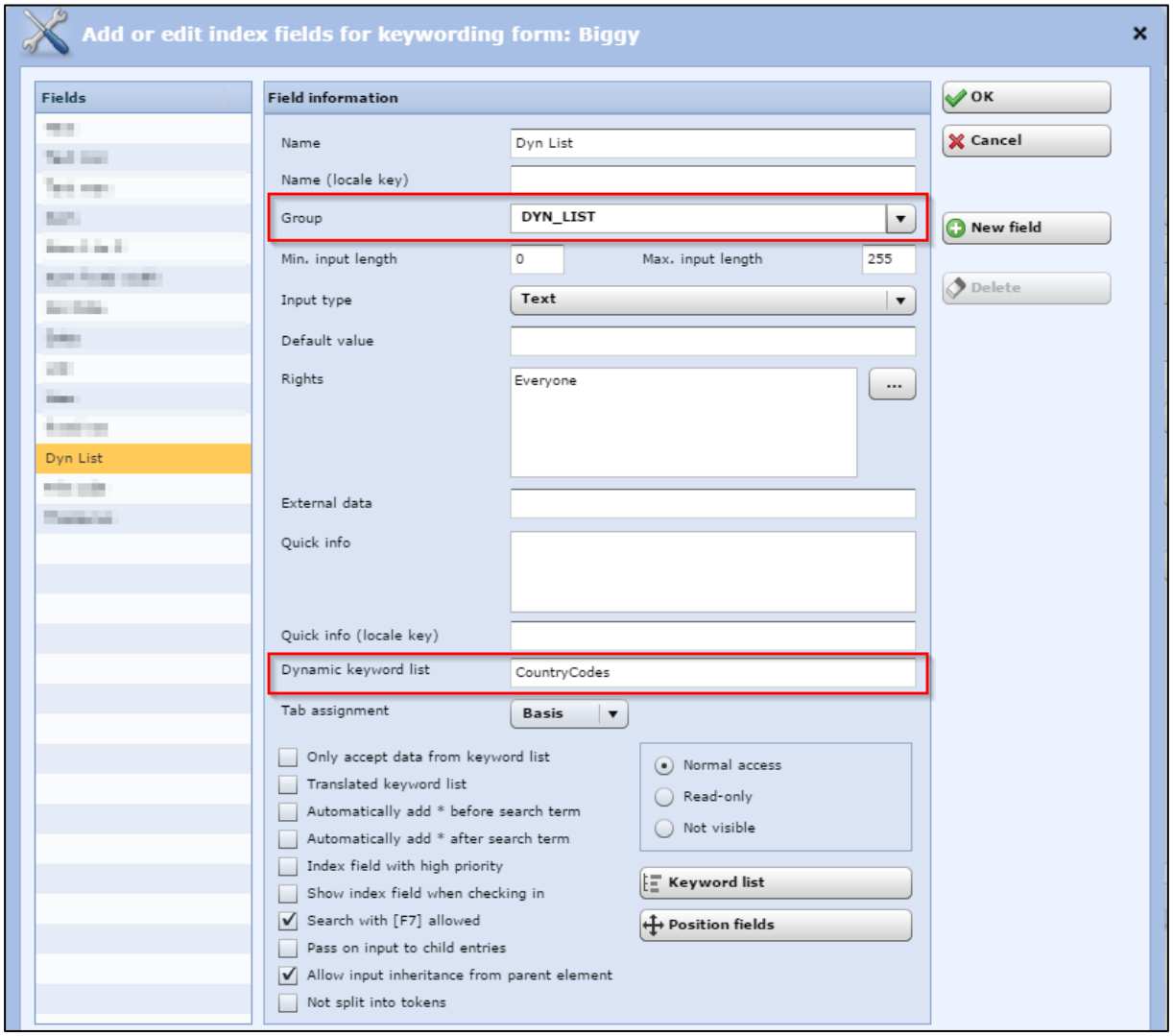


Abb. 25: Verschlagwortungsmasken-Editor, Felder 'Group' und 'Dynamic keyword list'

Um diese Listen in einem Formular zu verwenden, müssen sie in dem Formulardesigner folgendermaßen deklariert werden:

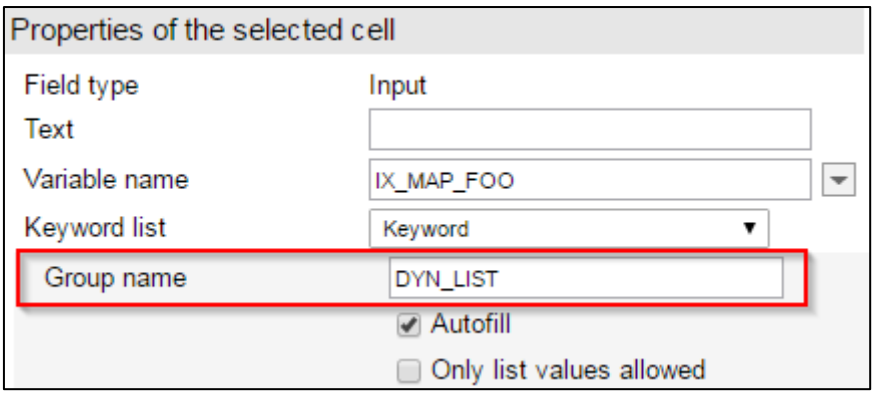


Abb. 26: Gruppenname 'DYN_LIST' im Formulardesigner

Das entsprechende Skript *CountryCodes* muss vorhanden sein und in folgendem Ordner liegen:

Administration // IndexServer Scripting Base.

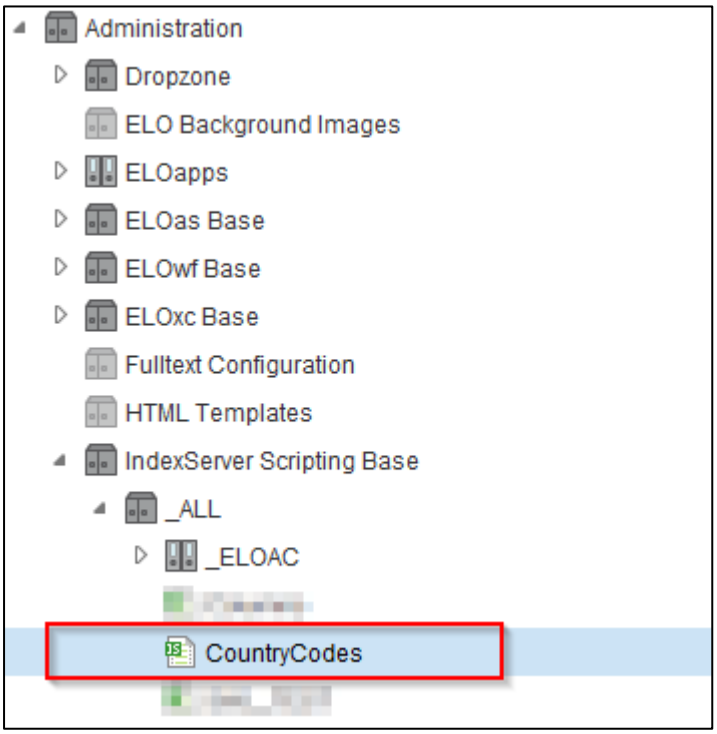


Abb. 27: Dokumentenpfad für das Skript 'CountryCodes'

Hier ist ein Beispiel für ein Skript, der eine statische Liste von Länderkürzel und -namen bereitstellt:

```
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 = [];

  /* Hilfefunktion, die eine Länderliste filtert */
  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("Nach Filter: " + results.length);
  }

  /* Wird bei der Initialisierung einer dynamischen Liste aufgerufen (durch
Deklarierung in der Verschlagwortungsmaske) */
  this.open = function(ec, sord, focus) {
    log.info("öffnen");

    this.target = focus;
    /*In diesem Fall wird das erste Indexfeld ausgewählt, idealerweise
sollte aber ein anderes ausgewählt werden */
    var filter = sord.objKeys[0].data[0] || "";

    this.filterCountries(filter);
  }

  /* Wird aufgerufen, wenn eine dynamische Liste durch das ELOwf-Feld "Dyn.
keywording map" initialisiert wird */
  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("schließen()");
  }

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

  /* Die Benennung der Spalten */
  this.getHeader = function() {
    log.info("getHeader()");
    return ["Code","Name"];
```

```
  }

  /* Das Ziel des ausgewählten Eintrags */
  this.getKeyNames = function() {
    log.info("getKeyNames()");
    return [this.focus, "EIN_ANDERES_INDEXFELD"];
  }

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

  /* Entweder wird eine Fehlermeldung zurückgegeben, oder bei Erfolg leer
gelassen */
  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'}
]
```

Lesen Sie dazu auch:

- Das Handbuch zum ELO Indexserver
- Folgenden Forumsbeitrag:

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

7.3 Dynamische Stichwort-Map

Eine dynamische Stichwort-Map ist ähnlich wie im vorherigen Beispiel, basiert aber auf Map-Feldern. Hierbei sollen Sie das Zielskript direkt angeben, wie im Screenshot unten abgebildet.

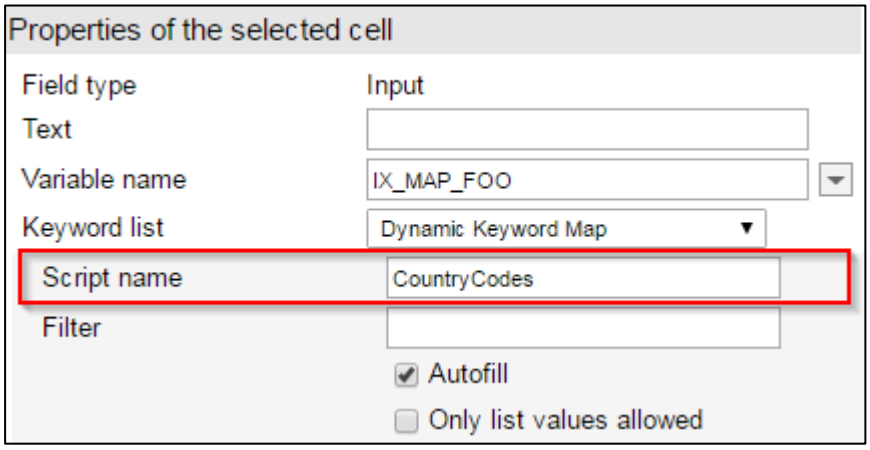


Abb. 28: Dynamische Stichwort-Map, Einstellungen im Formulardesigner

Hinweise

Wenn ein Eintrag in einer dynamischen Stichwortliste ausgewählt wird, wird die folgende Ereignisfunktion im Header Script aufgerufen:

```
onDynListItemSelected(eintrag)
```

> **Beachten Sie**: Es ist in dynamischen Stichwortlisten nicht möglich, die Option *Autovervollständigen* mit der Option *Nur Listeneinträge erlaubt* zu kombinieren. Der Hauptgrund hierfür ist, dass dynamische Stichwortlisten von mehreren Eingabefeldern abhängen können, sowie mehrere Eingabefelder modifizieren können. Es ist aktuell nicht möglich, eine Validierung in mehreren Feldern gleichzeitig zu unterstützen.

Alle Listen können auch innerhalb der Skripte angestoßen werden.

```
/**
 * Ruft eine bestimmte Regel in ELOas auf.
 */
function $listAs(skriptName, param2, param3, beiErfolg, beiFehler) {

/**
 * $listKw("IX_GRP_DYN_FIELD", ...) holt die Daten aus der Stichwort-
 * Liste, die in dem angegebenen Feld definiert wurde
 */
function $listKw(swlid, beiErfolg, beiFehler) {


/**
 * $listDyn("MeinSkript", "foo", ["bar"], ...) holt die Daten vom
 * entsprechenden Skript in 'IndexServer Scripting Base'.
 * Das Skript wird mit "foo" als Fokusname aufgerufen und {"foo": ...,
```

```
 * "bar": ...} als Map-Daten, auch mit den Wildcards "{i}" und "{*}"
ersetzt
 */
function $listDyn(skriptName, fokusfeld, filterfelder, beiErfolg,
beiFehler) {
```

8 Dynamische Templates

8.1 JS_AddLine

Durch das Hinzufügen einer JS_AddLine-Schaltfläche auf dem Formular kann die über der Schaltfläche liegende Zeile kopiert werden.

Zu beachten sind folgende Punkt:

- Alle Felder der zu kopierenden Seite sollten auf Map-Felder („IX_MAP_“, „WF_MAP_“) verweisen und müssen mit dem gleichen Index versehen werden („IX_MAP_<Name der Variablen>1“).
- Der zu kopierende Bereich kann über *lines* im Validierungsfeld des Formulardesigners erweitert werden, so dass nicht nur die über dem Button liegende Zeile (lines:1) sondern die x über dem Button liegenden Zeilen kopiert werden (lines:x).
- Durch das Attribut *max* im Validierungsfeld des Formulardesigners wird angegeben, wie viele Zeilen maximal hinzugefügt werden können, danach wird die Schaltfläche deaktiviert. Sollten beim Laden des Formulars mehr Daten in der Datenbank vorhanden sein, werden dennoch alle Daten in das Formular geladen, auch wenn dadurch der max-Parameter überschritten wird.
- Bei mehreren JS_AddLine-Feldern auf einem Formular sollten diese durch eine *addlineid* im Validierungsfeld des Formulardesigners erweitert werden. Diese Id wird, in Scripting-Funktionen verwendet, um zwischen den JS_AddLine-Feldern unterscheiden zu können.
- In den ELO Solutions sollten nicht mehr als 100 Zeilen auf einem Formular hinzugefügt werden.
- Es gibt eine feste Beschränkung von 2000 Zeilen.

9 Eigene Dateien hinzufügen

9.1 Bilder

Um eigene Bilder im Formular zu benutzen, sollten diese in über folgenden Pfad hinterlegt werden:

Administration // ELOwf Base // Images.

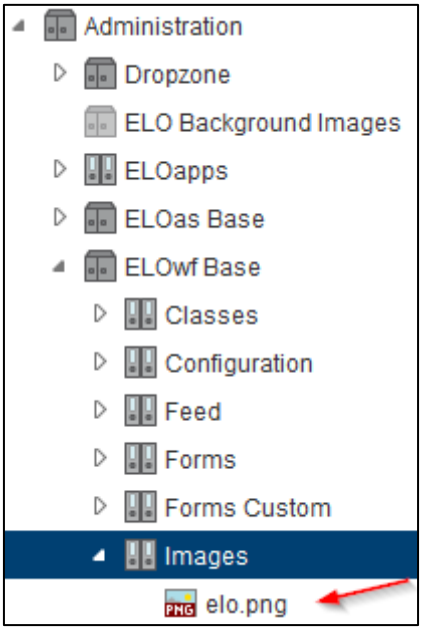


Abb. 29: Bild mit Dateiendung in der Kurzbezeichnung

Beachten Sie: Beim Einchecken einer Datei wird die Endung im Namen („.png“ oder „.jpg“ z. B.) standardmäßig entfernt. Wir empfehlen die Endung in der Kurzbezeichnung in diesem Fall hinzuzufügen.



Abb. 30: Funktion 'Aktualisieren' im Formulardesigner

Beim Aktualisieren vom ELOwf wird dieser Ordner auf den Server dann synchronisiert.

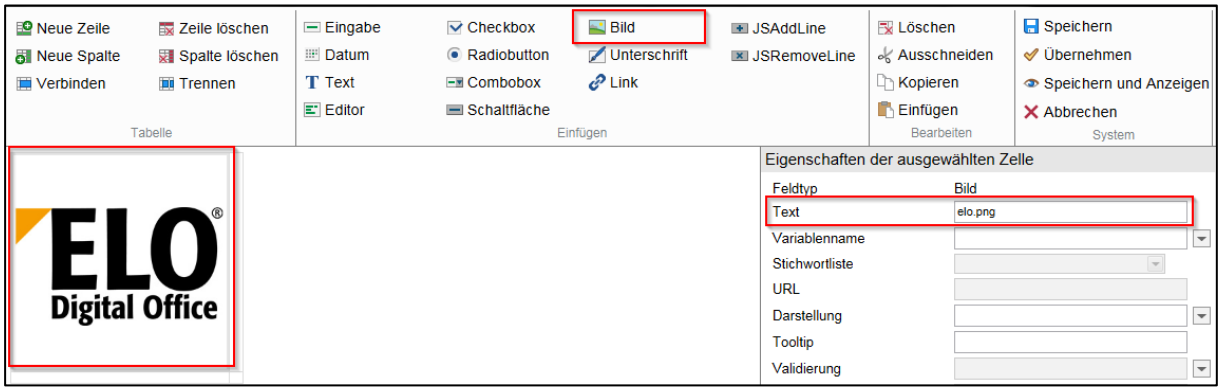


Abb. 31: Eingebettetes Bild

Das Bild kann dann einfach im Formular eingebettet werden

9.2 JavaScript-Dateien

Falls Sie JavaScript-Bibliotheken im Formular benutzen wollen, ob es eigene oder Third-party-libraries sind, sollten Sie diese unter folgendem Pfad ablegen:

Administration // ELOwf Base // Webapp

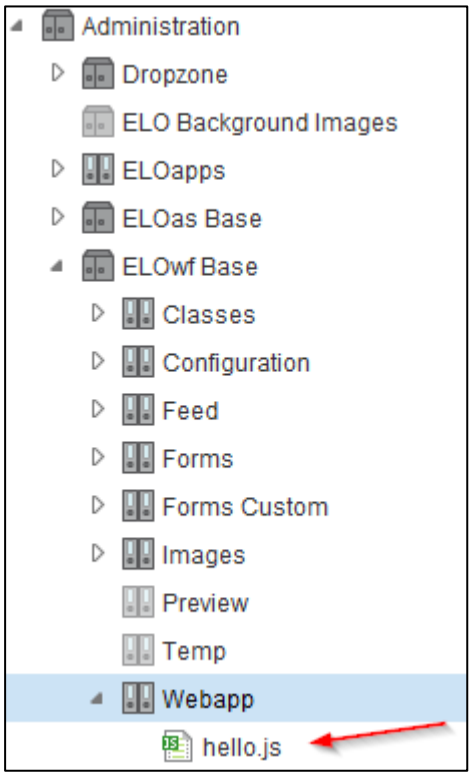


Abb. 32: JavaScript-Datei mit Dateiendung

Wie bei Bildern, empfehlen wir der Kurzbezeichnung die Dateiendung „.js“ hinzuzufügen.

Danach den ELOwf aktualisieren, und das Skript im Baustein *Benutzerskripte* des Formulars importieren:

```
<script type="text/javascript" src="hello.js" ></script>
```

9.3 CSS-Dateien

Es ist möglich, Formulare umzugestalten, indem Sie Inline-CSS im Formular-Header hinzufügen.

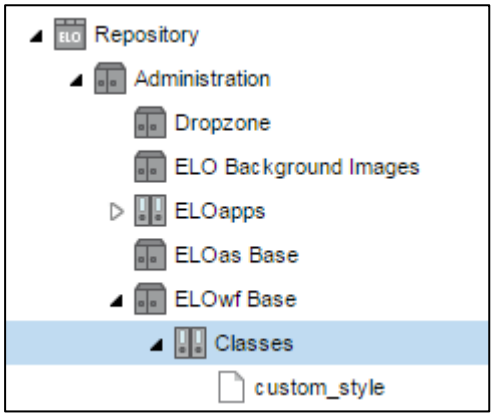


Abb. 33: Dokumentenpfad für CSS-Dateien

Falls Sie umfangreichere Gestaltungsmöglichkeiten benötigen, legen Sie entsprechende CSS-Dateien unter folgendem Pfad ab:

Administration // ELOwf Base // Classes

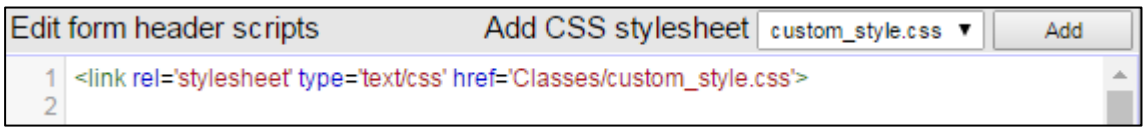


Abb. 34: Beispiel für das Deklarieren eines CSS-Styles im Header des Formulardesigners

Nachdem Sie die Dateien dort abgelegt haben, müssen Sie ELOwf aktualisieren, damit diese neuen Dateien erkannt werden. Danach sind sie über den Header des Formulardesigners verfügbar.

Wählen Sie ein Stylesheet und klicken Sie auf *Hinzufügen*, um die entsprechende Zeile im Header hinzuzufügen.

10 Ereignisse

Dieses Kapitel beschreibt einige Ereignisse, die in ELOwf-Formulare abgefeuert werden.

10.1 onInit()

Dieses Ereignis wird einmal aufgerufen, wenn das Formular geladen wird.

10.2 inputChanged(elem)

Dieses Ereignis wird immer aufgerufen, wenn der Benutzer die Inhalte des Formulars ändert (nach jedem Tastendruck). Das Eingabefeld wird als Argument mitgeliefert. Das Ereignis wird nicht aufgerufen, wenn ein Skript einen Wert ändert.

10.3 saveClicked()

Das Ereignis *saveClicked* wird aufgerufen, wenn ein Formular gespeichert wird. Genauer gesagt, es wird aufgerufen, bevor das Formular validiert und gespeichert wird.

Dieses Ereignis eignet sich, um zusätzliche Validierungen durchzuführen oder andere Variablen in dem Formular zu setzen, bevor es gespeichert wird.

Beispiel:

```
function saveClicked() {
  if( $num(“IX_GRP_PRICE”) > 10000 && $val(“IX_GRP_TYPE”) == “GUT&GÜNSTIG”)
) {
    eloAlert(“Sorry, das ist zu teuer!”);
    return false; // Nicht speichern und abbrechen
  }
  else {
    return true; // ok, weiter mit Validierung und Speichern
  }
}
```

Ab ELOwf 10.1 können auch asynchrone Prozesse ausgeführt werden, indem sie Promises zurückgeben:

```
function doSomething(resolve, reject) {
  // Daten asynchron holen oder verifizieren
  // ‘resolve()’ aufrufen, falls erfolgreich abgeschlossen
  // oder ‘reject()’ falls Prozess abgebrochen werden soll
}

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

Das Promise ist ein verzögertes Ergebnis, das verwendet wird, um den Prozess fortzuführen, falls es beschlossen oder abgelehnt wird.

10.4 nextClicked(id)

Dieses Ereignis funktioniert identisch zu *saveClicked*, aber es wird aufgerufen, falls das Formular weiter zum nächsten Workflowknoten weitergeleitet wird. Es kann ebenfalls „true“ zurückgeben, um weiter zu machen, „false“, um abzubrechen, oder ein Promise für eine verzögerte Auslösung.

Der einzige funktionelle Unterschied zwischen diesem und dem vorherigen Ereignis ist ein zusätzlicher Parameter „id“, der die ID des Workflowknotens referenziert, zu dem der Workflow weitergeleitet wird.

Statt der ID ist es auch möglich, den internen Knotennamen wiederherzustellen:

```
var nextNodeName;
for (var i=1; i<20; i+=1) {
     if (ELO_PARAMS["NEXT_" + i] && ELO_PARAMS["NEXT_" +
i].indexOf(String(id) + "\t") === 0 ){
          nextNodeName = ELO_PARAMS["NEXT_" + i].split("\t")[1];
     }
 }
```

Dazu wird über alle potenziellen Nachfolgeknoten iteriert (maximal 20) und diese mit der „id“ abgeglichen.

Der dort extrahierte Knotenname ist der technische oder übersetzte Name des Knotens. Falls hier mit einem Übersetzungsschlüssel gearbeitet wurde, steht in „ELO_PARAMS.["KEY_NEXT_" + i]“ der Übersetzungsschlüssel, dieser eignet sich besser für einen Vergleich, da er in allen Sprachen gleich ist.

Es wird auch zwischen dem Namen des Nachfolgeknotens und dem Label unterschieden, der im aktuellen Knoten angezeigt werden soll. Somit können Skriptknoten für den Benutzer übergangen werden. Falls diese Felder im Workflow verwendet werden, steht im Formular unter „ELO_PARAMS.["LABEL_NEXT_" + i]“ der übersetzte Anzeigename für diesen Knoten und unter „ELO_PARAMS.["LABEL_KEY_NEXT_" + i]“ der Übersetzungsschlüssel. All diese zusätzlichen Einträge müssen Sie auf Existenz prüfen, da sie optional sind.

11 Verschiedenes

11.1 Prüfung beim Speichern und Weiterleiten

Ab der Version 10.01 gibt es eine Prüfung beim Schreiben der Daten. Hier wird kontrolliert, ob seit dem Laden des Formulars (und somit dem Auslesen der Daten) Änderungen im Server vorgenommen wurden.

Dies könnte z. B. geschehen, wenn ein Benutzer A die Verschlagwortung ändert, während Benutzer B am Formular arbeitet. Solange B hier das Formular vor den Änderungen geladen hat, sieht er diese Änderungen nicht und bisher wurden diese einfach überschrieben.

Wird jetzt festgestellt, dass sich Daten verändert haben, wird über einen Dialog darauf hingewiesen. Der Benutzer kann dann bestätigen, dass er die Daten überschreiben möchte oder den Speichern/Weiterleiten Vorgang abbrechen.

Über die Einstellung im Script:

```
ELO.Configuration.ForceSave = true;
```

Wird das alte Verhalten angenommen und alle Änderungen ohne Rückfrage direkt überschrieben.

11.2 Drucken



Beachten Sie: Die folgenden Funktionen werden nur ausgeführt, wenn der Benutzer über die Schaltfläche *Drucken* etwas ausdruckt. Die Features werden nicht funktionieren, falls der Benutzer über *Strg+P* ausdruckt, oder über das Browsermenü geht.

11.2.1 Textbereiche automatisch erweitern

Beim Druck kann man das folgende Flag setzen, um die vollständigen Inhalte der Textbereiche zu sehen:

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

Fügen Sie dieses Flag in der Ereignisfunktion *inputChanged()* ein.

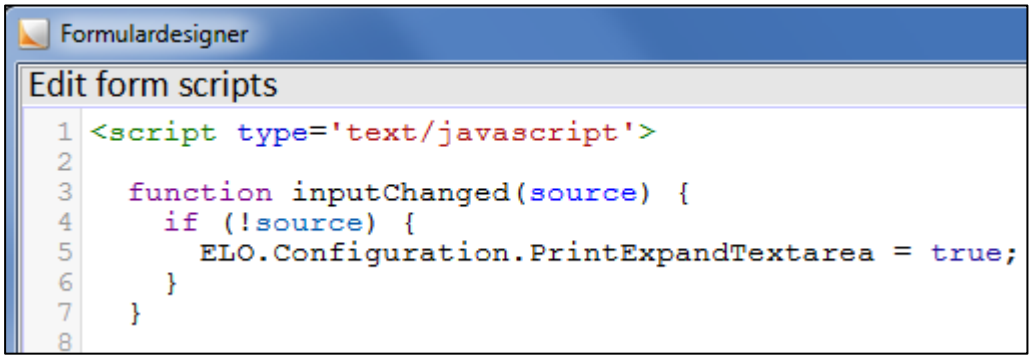


Abb. 35: JavaScript-Code: Ereignisfunktion 'inputChanged', Flag zur Erweiterung des Textbereichs

Nach Ausführung werden alle Textbereiche ausreichend größer gemacht.

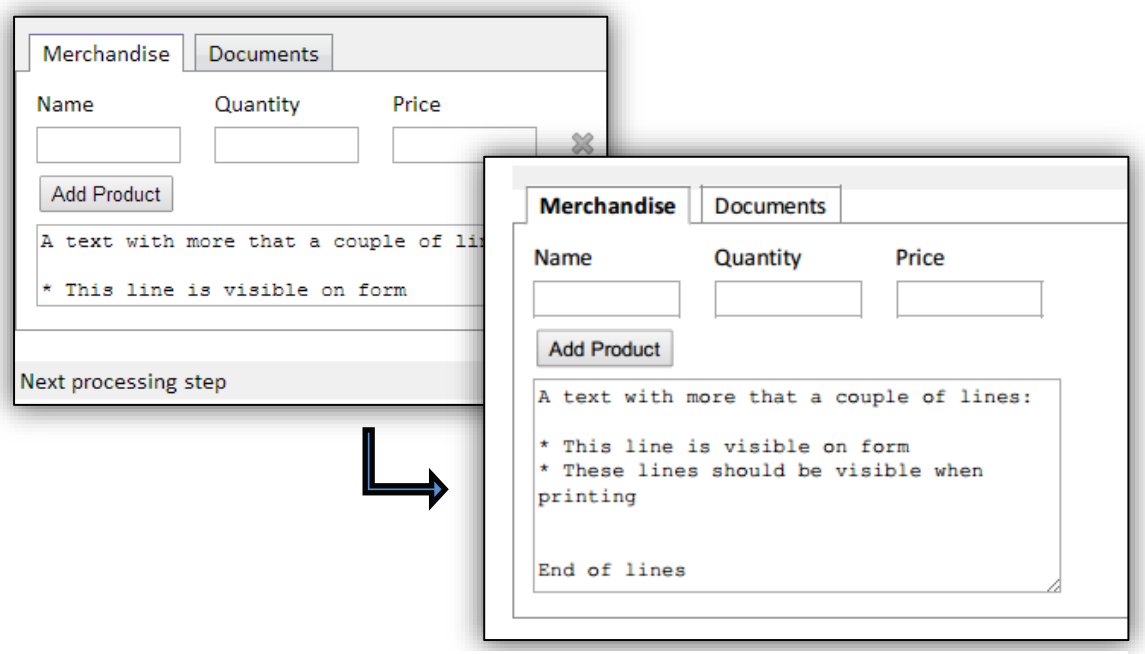


Abb. 36: Automatisch erweiterte Textbereiche im Formulardesigner

11.2.1.1 Alle Reiter ausdrucken

Falls Sie alle Reiter standardmäßig ausdrucken wollen, können Sie das folgende Flag setzen:

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

Fügen Sie dieses Flag in der Ereignisfunktion *inputChanged()* ein.

Edit form scripts

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

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

Abb. 37: JavaScript-Code: Ereignisfunktion 'inputChanged', Flag zum standardmäßigen Druck



Abb. 38: Formulardesigner, alle Reiter ausdrucken

11.3 Einstellbare Tastaturbefehle

Beachten Sie: Diese Funktionen werden eingeschränkt unterstützt, je nach angewendetem Client und Browser.

Tastaturbefehle funktionieren nur, wenn das Formular direkt geöffnet wird, über den ELO Web Client in Google Chrome oder dem Microsoft Internet Explorer. In Mozilla Firefox wird stattdessen das Browsermenü über die ALT-Taste geöffnet.

Der ELO Java Client hat überhaupt keine Unterstützung für diese Funktionalität. Andere Kombinationen von Browsern/Clients werden nicht unterstützt.

Es ist möglich, Tastaturkürzel einzustellen, um den Fokus direkt auf ein bestimmtes Feld zu setzen:

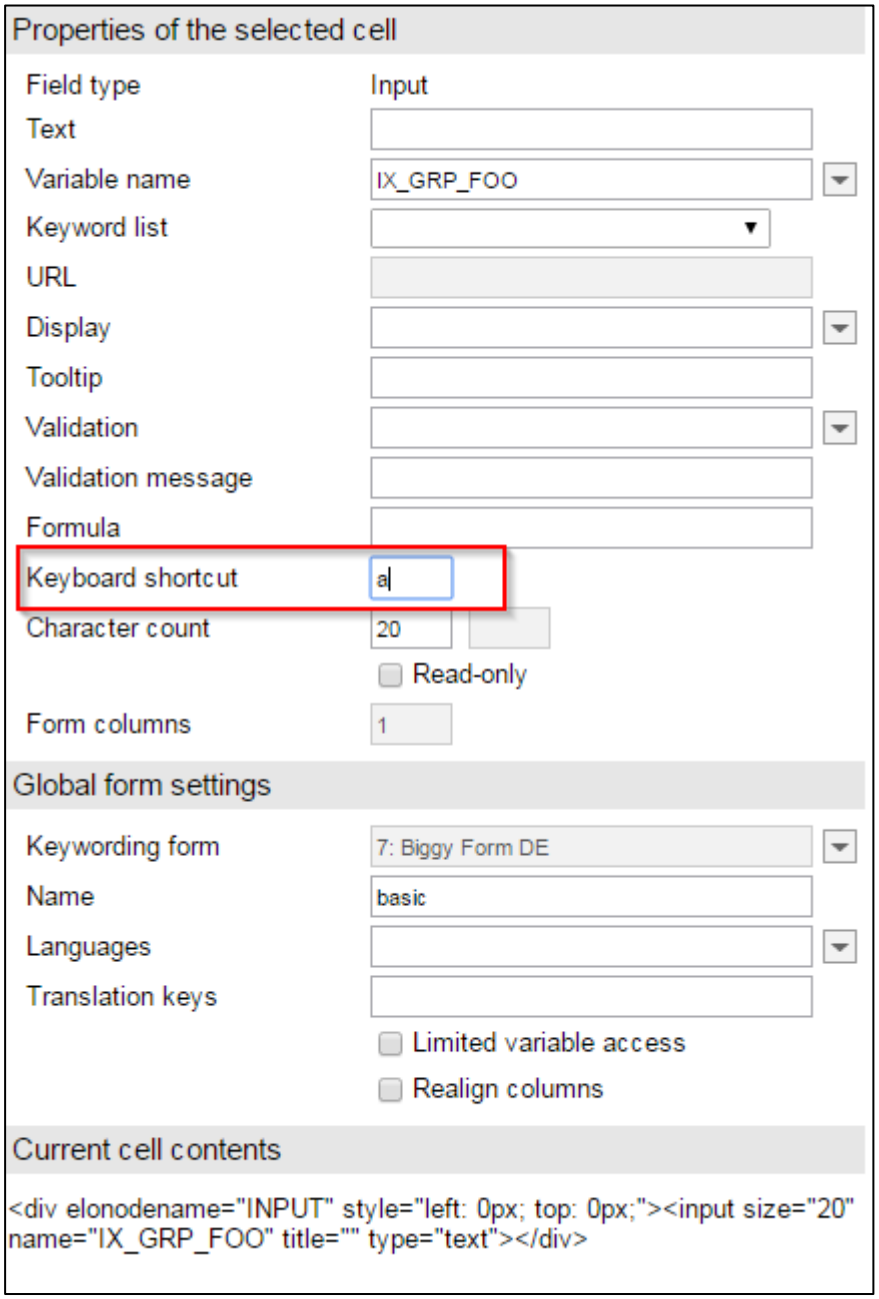


Abb. 39: Formulardesigner, Formularfeld 'Keyboard shortcut'

Drückt der Benutzer `ALT+A`, springt der Fokus zu dem entsprechenden Feld.

11.4 Betrieb mehrerer ELOwf Instanzen

Ab ELO12 unterstützt der ELOwf den Betrieb mehrerer Instanzen. Dazu müssen jeder ELOwf Instanz die URLs der jeweiligen anderen Instanzen übergeben werden. Dies geschieht in der *config.xml* des ELOwf über den Eintrag

```
wfInstances
```

mit einer kommaseparierten Liste von URLs. Zum Beispiel wird für ein Szenario mit drei Instanzen (serverA, serverB, serverC) in der *config.xml* des ELOwf auf serverA folgender Eintrag gemacht:

```
<entry key="wfInstances">http://serverB:9090/wf-Archive,
http://serverC:9090/wf-Archive</entry>
```

Auf serverB entsprechend:

```
<entry key="wfInstances">http://serverA:9090/wf-Archive,
http://serverC:9090/wf-Archive</entry>
```

Auf serverC entsprechend:

```
<entry key="wfInstances">http://serverA:9090/wf-Archive,
http://serverB:9090/wf-Archive</entry>
```



Beachten Sie: Ab ELO 12 wird die URL des ELOwf im Proxy-Plugin des Indexservers eingetragen. Jede ELOwf Instanz sollte somit zusammen mit jeweils einer ELOix Instanz eine Einheit bilden. Die Verteilung der Sessions findet dann vorgeschaltet statt und bezieht sich immer auf eine solche Einheit.