**EPO Mobile Apps**

Basierend auf dem EPO Connector

**Technical-dokumentation**

Status: Internal

Version: 2.0

Aktualisierung: Juli 2014

**EPO Mobile Apps**

**Inhaltsverzeichnis – Technische Dokumentation**

[1 Konzept / Architektur 4](#_Toc421523271)

[1.1 Beschreibung der Epo Mobile Apps 4](#_Toc421523272)

[1.2 technische Voraussetzungen 4](#_Toc421523273)

[2 EMA Framework 6](#_Toc421523274)

[2.1 Aufbau von EMACust 6](#_Toc421523275)

[2.1.1 Kurzbeschreibung 6](#_Toc421523276)

[2.1.2 Elemente von EMA Cust 6](#_Toc421523277)

[2.1.3 config.properties 6](#_Toc421523278)

[2.1.4 ema.html 9](#_Toc421523279)

[2.1.5 menu.json 9](#_Toc421523280)

[2.2 Aufbau von EMACore 10](#_Toc421523281)

[2.2.1 Kurzbeschreibung 10](#_Toc421523282)

[2.2.2 Elemente von EMA Core 10](#_Toc421523283)

[2.2.3 keyvalues.json 12](#_Toc421523284)

[2.2.4 modules/ema.login.js 12](#_Toc421523285)

[2.2.5 modules/ema.menu.js 13](#_Toc421523286)

[2.2.6 modules/ema.sync.js 13](#_Toc421523287)

[2.2.7 ressources/css/icons.css 14](#_Toc421523288)

[2.2.8 ressources/css/jquery-ui-1.10.4.custom.css 14](#_Toc421523289)

[2.2.9 ressources/css/res\_1000.css 14](#_Toc421523290)

[2.2.10 ressources/css/res\_699.css 14](#_Toc421523291)

[2.2.11 ressources/css/res\_700-999.css 14](#_Toc421523292)

[2.2.12 ressources/css/res\_720\_1280\_p.css 14](#_Toc421523293)

[2.2.13 ressources/css/zema.css 14](#_Toc421523294)

[2.2.14 ressources/css/clearsans\_... 14](#_Toc421523295)

[2.2.15 ressources/css/icons/ 14](#_Toc421523296)

[2.2.16 ressources/languages/ 14](#_Toc421523297)

[2.2.17 ressources/scripts/ema.datamanager.js 15](#_Toc421523298)

[2.2.18 ressources/scripts/ema.formgenerator.js 17](#_Toc421523299)

[2.2.19 ressources/scripts/ema.js 20](#_Toc421523300)

[2.2.20 ressources/scripts/ema.listgenerator.js 20](#_Toc421523301)

[2.2.21 ressources/scripts/ema.model.js 25](#_Toc421523302)

[2.2.22 ressources/scripts/ema.selectiongenerator.js 27](#_Toc421523303)

[2.2.23 ressources/scripts/ema.shell.js 28](#_Toc421523304)

[2.2.24 ressources/scripts/minutetimer.js 32](#_Toc421523305)

[2.3 Externe Bibliotheken 32](#_Toc421523306)

[2.3.1 IndexedDBShim.min.js 32](#_Toc421523307)

[2.3.2 jquery-2.1.1.js bzw. jquery-2.1.1.min.js 32](#_Toc421523308)

[2.3.3 jquery-ui-1.10.4.custom.min.js 32](#_Toc421523309)

[2.3.4 modernizr.js 32](#_Toc421523310)

[2.3.5 sjl.min.js 32](#_Toc421523311)

[2.3.6 signature\_pad.min.js 32](#_Toc421523312)

[3 EMA Modulentwicklung 33](#_Toc421523313)

[3.1 Erstellen eines neuen Moduls 33](#_Toc421523314)

[3.1.1 Festlegen des Namespace 33](#_Toc421523315)

[3.1.2 Festlegen der Modulstruktur und der einzelnen Formulare 34](#_Toc421523316)

[3.1.3 ema.appcache anpassen 35](#_Toc421523317)

[3.1.4 Webservices und Synchronisation definieren 36](#_Toc421523318)

[3.1.5 Aufbau eines JavaScript Files 37](#_Toc421523319)

[3.1.5.1 Kopfbereich 37](#_Toc421523320)

[3.1.5.2 Modulfunction 37](#_Toc421523321)

[3.1.5.3 Variablenbereich 38](#_Toc421523322)

[3.1.5.4 Functions und Return: 39](#_Toc421523323)

[3.1.5.5 Erforderliche und optionale Standardfunktionen 39](#_Toc421523324)

[3.1.5.6 Aufruf eines Webservices 40](#_Toc421523325)

[3.2 Entwickeln der Auswahl für die Belegliste 41](#_Toc421523326)

[3.3 Erstellen einer Auswahl für Inventurbelege 46](#_Toc421523327)

[3.4 Erstellen der Belegliste 50](#_Toc421523328)

[3.5 Erstellen des Formulars Inventurbeleg 56](#_Toc421523329)

[4 Ressourcen 59](#_Toc421523330)

[4.1 Eclipse 59](#_Toc421523331)

[4.2 JQuery 59](#_Toc421523332)

[4.3 Icons 59](#_Toc421523333)

[4.4 Modernizr 62](#_Toc421523334)

[4.5 Can i use 62](#_Toc421523335)

[4.6 HTML 5 62](#_Toc421523336)

[4.7 Single Page Applications 62](#_Toc421523337)

[4.8 JSON 62](#_Toc421523338)

[4.9 Use Strict 63](#_Toc421523339)

[4.10 JSLint 63](#_Toc421523340)

[5 Anhang 66](#_Toc421523341)

[5.1 SourceCode von Inventurbeleg 66](#_Toc421523342)

# Konzept / Architektur

## Beschreibung der Epo Mobile Apps

Epo Mobile Apps ist eine modulare, dynamisch erweiterbare Single Page Application. EMA Core ist hierbei das Kernstück der Applikation, es stellt viele Funktionen und einige zentrale Module zur Verfügung, die teilweise auch von anderen Modulen verwendet werden können.

Im Paket EMA Cust befinden sich alle kundenspezifischen Dateien wie z.b. die Startseite, die Config Datei und das Json für das Menü. Dadurch ist es möglich mehrere Instanzen von EMA auf einem System laufen zu lassen, die denselben Kern verwenden, aber ein unterschiedliches Menü haben.

Der Kern umfasst die Authentifizierung, die Verwaltung des Menüs, den Aufbau eines Dashboards mit bestimmten Informationen, die Erstellung einer Breadcrumb Navigation, die Synchronisation von Daten und das Speichern von Daten im Localstorage sowie in der IndexedDb des Browsers.

Über das Menü, das dynamisch auf Basis eines Json-Files oder über einen Webserviceaufruf erstellt wird wird gesteuert, welche Module die Benutzer sehen und verwenden können. Beim Aufruf eines bestimmten Menüpunkts das entsprechende Modul geladen und ist ab dann in der Applikation verfügbar. Dadurch können einzelne Module an unterschiedlichen Speicherplätzen liegen und müssen der Kernapplikation beim Starten nicht unbedingt bekannt sein.

Für die Offlinefähigkeit der Applikation ist es jedoch notwendig, dass sämtliche benötigten Dateien und Ressourcen in einem Manifestfile erfasst werden, damit diese durch den Browser offline gespeichert werden können.

Durch den Kern müssen bei der Entwicklung von Modulen viele Dinge nicht mehr neu überlegt und entwickelt werden. Will man Daten von einem Webservice abfragen so kann man eine zentrale Funktion dafür verwenden. Ist der Aufruf in der Config Datei entsprechend konfiguriert so werden die Daten automatisch in der IndexedDB gespeichert und können später auch im Offlinemodus verwendet werden. Will man ein Formular anzeigen, so wird durch die entsprechende Funktion automatisch die Breadcrumb Navigation angepasst, das Dashboard aktualisiert und die Vorwärts/Rückwärts Navigation im Browser gesetzt.

Dadurch wird einerseits die Entwicklung von Modulen vereinfacht und beschleunigt, andererseits wird es leichter Änderungen vorzunehmen, da diese an weniger Stellen durchgeführt werden müssen.

## technische Voraussetzungen

Für die Verwendung der Epo Mobile Apps wird ein aktueller Browser benötigt, der HTML 5 unterstützt.

Im Detail muss der Browser folgende Features unterstützen:

* IndexedDB
* LocalStorage
* Offlinefähigkeit über ein Manifestfile
* Canvas
* Neue Feldtypen wie Date und Number

Folgende Browser unterstützen HTML5:

* Internet Explorer ab V10
* Firefox ab V29
* Chrome ab V34
* Safari ab V7
* Opera ab V22
* iOS Safari ab V5.0
* Android Browser ab V3.0

# EMA Framework

## Aufbau von EMACust

### Kurzbeschreibung

Im Paket EMA Cust befinden sich alle kundenspezifischen Dateien. Das Paket kann in einem beliebigen Ordner auf dem Server gelegt werden, wodurch es möglich ist mehrere unterschiedliche Instanzen gleichzeitig zu betreiben, die den Kern und andere Module gemeinsam verwenden, aber ein unterschiedliches Menü, unterschiedliche Logos usw. haben.

### Elemente von EMA Cust

EMA Cust besteht aus folgenden Elementen, die in den folgenden Kapiteln im Detail erklärt werden – diese Auflistung entspricht den aktuellen Dateipfaden im Projekt und nicht der Objektstruktur in Java Script

config.properties

ema.appcache

ema.html

menu.json

### config.properties

In diesem File werden alle möglichen zentralen Konfigurationen der Applikation als Json String gespeichert. Hierbei gibt es folgende Abschnitte:

* Config
  + Version: die Version der Applikation
  + Hostprotocol: Protokoll für Ajax-Requests, z.B. http oder https
  + Hosturl: URL für Ajax-Requests
  + Hostport: Port für Ajax-Requests, z.B. 8010
  + Hostpath: Pfad für Ajax-Requests – Requests werden also gegen die Adresse Hostprotocol + „://“ + Hosturl + „:“ + Hostport + Hostpath durchgeführt
  + Loginpath: Pfad für die Anmeldung – der Pfad weicht von anderen Requests ab, weil hier ein anonymer Zugriff erforderlich ist
  + Searchpath: Pfad für die die Eposuche, die in Autocompletefeldern verwendet wird
  + Statefulpath: Pfad für Requests, die „stateful“ sein müssen, z.b. um ein Dokument zu sperren
  + Hosttimeout: Timeout im Millisekunden
  + RequestType: JSONP oder JSON (wird an die URL angehängt)
  + Client: Mandant in SAP
  + Json\_strip: gibt an, ob leere Strukturen bei Requests ausgeblendet werden sollen
  + Json\_format: Definiert das Format des übergebenen Jsonstrings bei Requests
  + Ajax\_Datatype: Datatype der Requests, derzeit werden json und jsonp unterstützt
  + Ajax\_Type: GET oder POST
  + Ajax\_Auth: wie erfolgt die Authentifizierung:
    - url: Benutzername und Passwort werden in der URL als Parameter übergeben
    - basic
    - saphttpfields: ist ähnlich wie basic, es werden aber SAP spezifische Header Parameter übergeben
  + Search\_Datatype: Datatype bei Searchrequests, derzeit werden json und jsonp unterstützt
  + Debug: URL Parameter bei Requests
  + I\_Menu: Baustein bei dem das Menü abgefragt werden kann
  + Logoutminutes: Nach wievielen Minuten erfolgt der automatische Logout
  + Isloginrequired: Ist ein Login erforderlich oder wird gleich das Menü geladen
  + Contentcontainer: Name des Div-Elements in dem der Content der Applikation angezeigt wird (sollte sich nicht ändern)
  + Menuislocal: Menü ist lokal im File menu.json und muss nicht abgefragt werden
  + Maxlistrows: maximale Anzahl von Listeneinträgen, die von SAP zurückgeliefert werden
  + LGNUM: Lagernummer für Inventur
  + Signatureminwidth: minimale Breite von Strichen in der Unterschrift
  + Signaturemaxwidth: maximale Breite von Strichen in der Unterschrift
  + Signaturecolor: Farbe der Unterschrift
  + Werk: Werknummer in SAP
  + Developmentmode:
    - true: ermöglicht debuggen im Browser, Module werden in diesem Modus aber nicht zuverlässig geladen (sjl.js wird zum nachladen verwendet)
    - false: debuggen ist nicht möglich, das Laden von Modulen ist dafür aber zuverlässig (jquery wird zum nachladen verwendet)
  + Footerlogoleft: Pfad des Logos links in der Fußzeile
  + Footerlogoright: Pfad des Logos rechts in der Fußzeile
  + Corepath: Pfad unter dem EMA Core zu finden ist
* Override:

In diesem Abschnitt kann man Properties aus dem Abschnitt Config für bestimmte Hosts übersteuern. Dadurch kann man in einem Config File die Einstellungen für mehrere Hosts speichern.

Beispiel:

"OVERRIDE":

{

"epoconsulting.com":

{

"HOSTURL":"epoconsulting.com",

"HOSTPORT":"8083"

}

},

* Modules:

In diesem Abschnitt kann man Files von Modulen angeben, die automatisch nachgeladen werden sollen. Dazu gehören Sprachfiles, Configfiles und Javascriptdateien mit allgemeinen Funktionen für ein Modul.

* + Configs: Hier werden Configs als Array angegeben. Dadurch ist es möglich die Config dynamisch mit Modulspezifischen Werten zu erweitern, das Überschreiben eines Wertes ist jedoch nicht möglich.
  + Scripts: Hier werden Javascript Files als Array angegeben. An dieser Stelle werden sollen nur allgemeine JS Files der Module angegeben werden und nicht die Files die der einzelnen Transaktionen
  + Languages: Hier werden Ordner angegeben, in denen Sprachfiles zu finden sind. Es wird dann versucht die Files mit den in der allgemeinen Config eingetragenen Sprachen zu laden. Steht in der allgemeinen Config z.B. „de“ und „en“ und unter Languages „../zematest/languages“ so wird versucht „../zematest/languages/de.json“ und „../zematest/languages/en.json“ zu laden.
* IndexedDB: In diesem Abschnitt werden die Objectstores der IndexedDB, der Sync sowie die verfügbaren Transaktionen in SAP konfiguriert
  + DBName: Name der IndexedDB
  + Objectstores: Config der einzelnen Datenbank“tabellen“
    - Internalname: interne Bezeichnung zur eindeutigen Identifizierung in der Applikation
    - Operation. Funktionsbaustein in SAP
    - Storename: „Tabellenbezeichnung“ für die IndexedDB
    - Jsonformat: generell 0, bei großen Datenmengen 3
    - Userspecific: wird hier true gesetzt so wird dieser Objectstore für jeden Benutzer erstellt (z.B. requeststore\_gundacker)
    - Messagehierarchy: an welcher Stelle im Jsonobjekt werden Fehlermeldungen übergeben
    - Objecthierarchy. an welcher Stelle im Jsonobjekt werden die eigentlichen Daten übergeben
    - Keyhierarchy: an welcher Stelle in einem Datenobjekt befindet sich der eindeutige Schlüssel
    - Indizes: Wie bei Keyhierarchy können noch beliebig viele weitere Schlüssel definiert werden für die jeweils ein Index erstellt wird
    - Downloadconfig: Wird dieser Abschnitt konfiguriert so wird die Abfrage nach dem Login durchgeführt – auf diese Weise können größere Datenmengen im Hintergrund auf das Gerät synchronisiert werden.
      * Syncdesc: Bezeichnung im Schirm „Synchronisation“
      * Synctype: complete oder delta – bei Complete werden vor der Synchronisation alle Daten des Objectstores gelöscht
      * Requestobject: Abfrage mit der die Daten ermittelt werden
    - Uploadconfig: Wird dieser Abschnitt konfiguriert so werden Daten dieses Abfragetyps, die im Offlinemodus gespeichert werden sollen im Requeststore zwischengespeichert und später, wenn wieder eine Verbindung zu SAP möglich ist ins System zurückgespielt
      * Syncdesc: Bezeichnung im Schirm „Synchronisation“
* Languages: Hier werden die verfügbaren Sprachen angegeben, jeder Wert besteht aus einer Kurzbezeichnung und einem Anzeigenamen z.B. „de“:“Deutsch“. Für jede hier angegebene Sprache muss ein Sprachfile mit den Übersetzungen erstellt werden.
* Keyvalues: Hier können verfügbare Schlüsselwerte eingestellt werden. Beim Starten der Applikation werden diese in den Speicher geladen.
  + Type: SAP oder File
  + Location: bei SAP der Internalname des Objectstores, bei File der Filename
  + Entries: Bezeichnung der Schlüsselwerte

### ema.html

Diese Datei ist die Startseite der Applikation. In diesem File sind die “statischen” Files des Projektes, wie z.B. die CSS-Dateien, die externen JS-Files und die eigenen JS-Files, die nicht zu Modulen gehören eingebunden. Wichtig hierbei ist, dass die eigenen JS-Files erst am Ende eingebunden werden um das Risiko zu verhindern, dass Variablen durch fremde Funktionen überschrieben werden.

In dieser Datei wird die Funktion „ema.initModule“ des Files ema.js aufgerufen.

### menu.json

Wenn in config.properties MENUISLOCAL den Wert “true” hat, dann wird das Menü in diesem File gesucht. Der Aufbau der Datei ist so, wie er auch von einem Webservice erwartet würde.

Die Kopfdaten werden hier nicht weiter verarbeitet, für das Menü interessant ist der rot marktierte Teil.

{

"SEQUENCE":"",

"EXPORT":

{

"E\_CLIENT":"200",

"E\_MENU":"ZDEMO",

"E\_LANGUAGE":"D",

"ET\_MENU":

[

[

1,

0,

"EMA Men&uuml; f&uuml;r Demo",

"",

"",

"app",

"",

],

[

2,

1,

"Inventur",

"",

"",

"app",

"icon-truck",

],

[

3,

2,

"Inventur LVS",

"",

"",

"app",

"icon-docs",

Die eckigen Klammern umfassen immer einen Menüpunkt, der durch 7 Einzelwerte definiert wird. Diese Werte sind:

1. Laufnummer
2. Laufnummer des übergeordneten Menüpunktes
3. Bezeichnung der Kachel im Menü
4. ???
5. Pfad des Quelltextes bzw. URL
6. Typ
   1. app: interne App oder ein Modul
   2. url: Url, die in einem neuen Fenster aufgerufen werden soll
   3. embeddedurl: Url, die in einem Iframe in der Applikation aufgemacht werden soll
7. Icon, das auf der Kachel angezeigt werden soll

Mit diesem Json wird das Menü erstellt, dabei wird für jeden Menüpunkt, bei dem Wert 5 leer ist ein neuer Menücontainer erstellt, der mit den Menüpunkten befüllt wird, die an diesem Elternelement hängen. Initial wird nur der Hautpmenücontainer angezeigt, alle anderen sind ausgeblendet.

## Aufbau von EMACore

### Kurzbeschreibung

Durch die Architektur der Applikation als Single Page Application (SPA) ist es möglich zu steuern, welche Funktionen eines JS-Files öffentlich zur Verfügung stehen und welche nicht. Auch bei Variablen kann definiert werden, ob sie abgefragt oder geändert werden können.

Beim Aufruf der Applikation wird ein zentrales Objekt „ema“ erstellt. Alle weiteren JS-Files erstellen in diesem Objekt ein Unterobjekt wie z.B. „ema.shell“ oder „ema.inventur.lvs.inventurbeleg“. Über diese Objektstruktur ist ein Zugriff auf Funktionen eines Moduls einfach möglich.

Html wird in den meisten Fällen dynamisch in JS-Funktionen erstellt, die Startseite ema.html bildet hier die einzige Ausnahme.

### Elemente von EMA Core

EMA Core besteht aus folgenden Elementen, die in den folgenden Kapiteln im Detail erklärt werden – diese Auflistung entspricht den aktuellen Dateipfaden im Projekt und nicht der Objektstruktur in Java Script

keyvalues.json

modules/ema.login.js

modules/ema.menu.js

modules/ema.sync.js

ressources/css/icons.css

ressources/css/jquery-ui-1.10.4.custom.css

ressources/css/res\_1000.css

ressources/css/res\_699.css

ressources/css/res\_700-999.css

ressources/css/res\_720\_1280\_p.css

ressources/css/zema.css

ressources/css/clearsans\_bold/ClearSans-Bold-webfont.eot

ressources/css/clearsans\_bold/ClearSans-Bold-webfont.svg

ressources/css/clearsans\_bold/ClearSans-Bold-webfont.ttf

ressources/css/clearsans\_bold/ClearSans-Bold-webfont.woff

ressources/css/clearsans\_light/ClearSans-Light-webfont.eot

ressources/css/clearsans\_light/ClearSans-Light-webfont.svg

ressources/css/clearsans\_light/ClearSans-Light-webfont.ttf

ressources/css/clearsans\_light/ClearSans-Light-webfont.woff

ressources/css/clearsans\_regular/ClearSans-Regular-webfont.eot

ressources/css/clearsans\_regular/ClearSans-Regular-webfont.svg

ressources/css/clearsans\_regular/ClearSans-Regular-webfont.ttf

ressources/css/clearsans\_regular/ClearSans-Regular-webfont.woff

ressources/css/icons/epo.eot

ressources/css/icons/epo.svg

ressources/css/icons/epo.ttf

ressources/css/icons/epo.woff

ressources/images/csscheckbox\_40.png

ressources/images/csscheckbox\_plus.png

ressources/images/csscheckbox\_raute.png

ressources/images/csscheckbox.png

ressources/images/icon\_pdf.png

ressources/languages/de.json

ressources/languages/en.json

ressources/scripts/chart.min.js

ressources/scripts/ema.datamanager.js

ressources/scripts/ema.formgenerator.js

ressources/scripts/ema.js

ressources/scripts/ema.listgenerator.js

ressources/scripts/ema.model.js

ressources/scripts/ema.selectiongenerator.js

ressources/scripts/ema.shell.js

ressources/scripts/IndexedDBShim.min.js

ressources/scripts/jquery-2.1.1.min.js

ressources/scripts/jquery-ui-1.10.4.custom.min.js

ressources/scripts/minutetimer.js

ressources/scripts/modernizr.js

ressources/scripts/pdfmake.min.js

ressources/scripts/pdfmake.min.map

ressources/scripts/signature\_pad.min.js

ressources/scripts/sjl.min.js

ressources/scripts/vfs\_fonts.js

### keyvalues.json

In diesem File werden Schlüsselwerte gespeichert, die lokal vorliegen und nicht aus SAP oder einem anderen Sysetm abgefragt werden. Der Aufbau der Datei ist

{

"LAGERTYP":

{

"---":"---Ausw&auml;hlen---",

"001":"Hochregallager",

"002":"Regallager",

"003":"Freilager",

"004":"Blocklager"

},

...

}

### modules/ema.login.js

Dieses File zeigt die Loginmaske an und führt den Login entweder gegen ein entferntes System oder gegen den Localstorage aus.

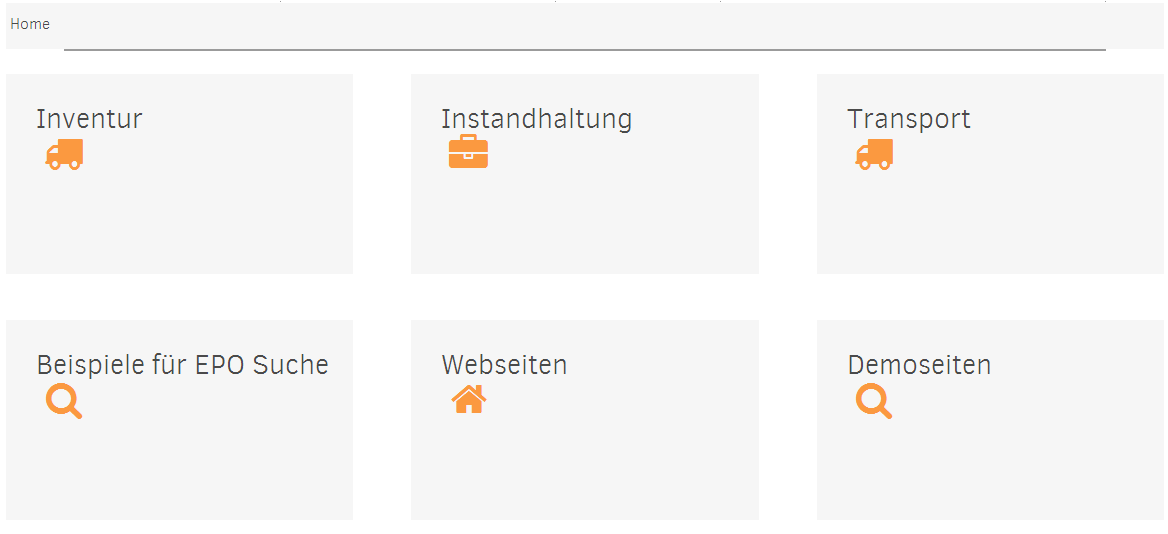


Öffentliche Funktionen:

* initModule(path, container)
  + wird beim ersten Laden des Moduls ausgeführt
  + path: modules/ema.login
  + container: Name des Containerelements
* onModuleLoad(container)
  + wird bei jeder Anzeige des Formulars ausgeführt (also nicht nur beim ersten Mal)
  + container: Name des Containerelements
* processLogin
  + führt den Login durch
* switchLanguage(elem)
  + stellt die Applikation auf die gewählte Sprache um
  + elem: das aktuelle Element – in diesem Fall das Dropdownfeld zur Sprachauswahl

### modules/ema.menu.js

Dieses Modul erstellt das Menü. Es wird entweder sofort beim Start oder nach einem erfolgreichen Login aufgerufen.



Öffentliche Funktionen:

* initModule(path, container)
  + wird beim ersten Laden des Moduls ausgeführt
  + path: modules/ema.login
  + container: Name des Containerelements

### modules/ema.sync.js

Erstellt die Seite “Synchronisation”, vor dem Aufbau der Seite wird geprüft, ob eine Verbindung zum Server besteht und ob zwischengespeicherte Daten übertragen werden können.



Öffentliche Funktionen:

* initModule(path, container)
  + wird beim ersten Laden des Moduls ausgeführt
  + path: modules/ema.login
  + container: Name des Containerelements
* onModuleLoad(container)
  + wird bei jeder Anzeige des Formulars ausgeführt (also nicht nur beim ersten Mal)
  + container: Name des Containerelements

### ressources/css/icons.css

Hier finden sich die Stile, die notwendig sind um die Icons aus dem Iconfont unter “ressources/css/icons” anzeigen zu können.

### ressources/css/jquery-ui-1.10.4.custom.css

Dieses File steuert die Stile für autocomplete Felder.

### ressources/css/res\_1000.css

In diesem File finden sich Stile für Auflösungen mehr als 1000 Pixel Breite.

### ressources/css/res\_699.css

In diesem File finden sich Stile für Auflösungen weniger als 700 Pixel Breite.

### ressources/css/res\_700-999.css

In diesem File finden sich Stile für Auflösungen zwischen 700 und 999 Pixel Breite.

### ressources/css/res\_720\_1280\_p.css

In diesem File finden sich Stile für die Auflösung 720 \* 1280 Pixel im Portrait-Modus.

### ressources/css/zema.css

In diesem Stylesheet finden alle relevanten Stile, die in der Applikation verwendet werden. Für viele optische Anpassungen sollte es ausreichen diese Datei entsprechend anzupassen.

### ressources/css/clearsans\_...

In diesen Ordnern befinden sich die Webfonts, die in der Applikation verwendet werden.

### ressources/css/icons/

Dieser Ordner beinhaltet den Iconfont der Applikation.

### ressources/languages/

In diesem Ordner finden sich die Sprachdateien der Applikation. Im Prinzip kann können Sprachdateien für beliebig viele Sprachen erstellt werden, am besten kopiert man für eine neue Sprache eine bestehende Datei und ändert darin die Textbausteine.

### ressources/scripts/ema.datamanager.js

Der Datamanager führt die Abfrage von Daten und deren Speicherung in der IndexedDB aus. Es wird empfohlen Abfragen ausschliesslich über den Datamanager durchzuführen, da man sich dadurch keine weiteren Gedanken darüber machen muss, wie die Daten lokal gespeichert werden, oder wann man Daten aus der Datenbank abfragen muss und wann vom Server.

Öffentliche Funktionen:

* doRequest(operation, request, callbackFunction, reqKey)
  + diese Funktion führt eine Abfrage gegen den Server oder gegen die IndexedDb aus.Für jede operation für die ein objectstore definiert wurde wird ein Eintrag im requeststore erstellt um die Abfrage später im Offlinemodus wiederholen zu können
  + operation: der Internalname der Abfrage
  + request: der gewünschte Request als Javascript Object
  + callbackFunction: die Funktion, die nach der Abfrage aufgerufen werden soll – hierbei ist es wichtig, dass nicht der Funktionsname als String, sondern die Funktion übergeben wird.
  + reqKey: ein Key um die Abfrage eindeutig zu identifizieren, z.B. die Belegnummer, wird hier null übergeben so wird die aktuelle Zeit in Millisekunden genommen
* doMultipleRequests(requestObj)
  + Mit dieser Funktion können mehrere Requests mit einem Auftruf abgesetzt werden. Dabei kann es sich z.B. um die Genehmigung von mehreren voneinander unabhängigen Anträgen handeln. Die Funktion führt die Requests nacheinander aus und gibt dann gesammelt eine Rückmeldung aus.
  + requestObj: in diesem Objekt werden die Requests übergeben
    - requests: Liste der tatsächlichen Requests
      * operation: der Internalname der Abfrage
      * request: der gewünschte Request als Javascript Object
      * callbackFunction: die Funktion, die nach der Abfrage aufgerufen werden soll – hierbei ist es wichtig, dass nicht der Funktionsname als String, sondern die Funktion übergeben wird.
      * reqKey: ein Key um die Abfrage eindeutig zu identifizieren, z.B. die Belegnummer, wird hier null übergeben so wird die aktuelle Zeit in Millisekunden genommen
* requestRecordsByKey(operation, keys, request, callbackFunction)
  + diese Funktion kann aufgerufen werden, wenn eine Abfrage nur einen Datensatz liefern soll. Im Gegensatz zu doRequest sucht diese Funktion im Offlinemodus nicht nach einem passenden, bereits ausgeführten Request sondern sucht gleich in der IndexedDb anhand des Schlüssels der übergeben wurde.
  + operation: der Internalname der Abfrage
  + keys: Array mit dem Schlüssel des gewünschten Datensatzes
  + request: der gewünschte Request als Javascript Object
  + callbackFunction: die Funktion, die nach der Abfrage aufgerufen werden soll – hierbei ist es wichtig, dass nicht der Funktionsname als String, sondern die Funktion übergeben wird.
* initDatabase(sourcePath, navText)
  + initialisiert die IndexedDB beim Starten der Applikation. Es wird geprüft, ob alle notwendigen Objectstores vorhanden sind, wenn nicht wird die Datenbank geschlossen und mit einer höheren Versionsnummer neu geöffnet, damit die Obejctstores erstellt werden können. Ist die Datenbank korrekt geöffnet wird die Folgeseite aufgerufen
  + sourcePath: Pfad der Folgeseite
  + navText: Bezeichnung der Folgeseite
* startSynchronisation(syncType)
  + Führt die Synchronisation durch, entweder in beide Richtungen oder nur up/down
  + syncType: kann die Werte both, up oder down haben
* startSingleDownloadSync(taskName)
  + führt die Synchronisation für einen einzelnen Downloadtask aus
  + taskName: Internalname des Downloadtasks
* startUploadByKey(keys)
  + startet den Uploadsync für einen oder mehrere local zwischengespeicherte Datensätze
  + keys: ein Array mit den Keys der Datensätze
* updateSyncStatus
  + berechnet den aktuellen Status der Synchronisation, aktualisiert das Dashboard und erstellt den Bildschirm “Synchronisation” neu, wenn er sichtbar ist. Der aktuelle Status ist der höchste(grün<gelb<rot) Status mit der aktuellen Synczeit
* getInboundSyncTasks
  + liefert das JS Object stateMap.inboundSyncTasks
* getOutboundSyncOperations
  + liefert das JS Object stateMap.outboundSyncOperations
* getEMADatabase
  + gibt das EMA Datenbank Object zurück
* readOfflineData(internalName, keys, callbackFunction)
  + diese Funktion liest Daten aus der IndexedDB und liefert diese an die übergebene Callbackfunction
  + internalName: der Internalname der Abfrage
  + keys: Array mit dem Schlüssel des gewünschten Datensatzes
  + callbackFunction: die Funktion, die nach der Abfrage aufgerufen werden soll – hierbei ist es wichtig, dass nicht der Funktionsname als String, sondern die Funktion übergeben wird.
* deleteOutboundSyncTask(taskKeys)
  + Löscht eine oder mehrere Outbound Datensätze aus der IndexedDb
  + taskKeys: die Schlüssel der Abfragen die gelöscht werden sollen
* getStoreName(internalName)
  + Diese Funktion erwartet den Internalname einer Abfrage und liefert den zugehörigen Storename zurück
  + internalName: der Internalname der Abfrage
* deleteEntryFromObjectStore(storeName, key)
  + Löscht einen Datensatz aus einem Objectstore in der IndexedDb
  + storeName: Name des Objectstores
  + key: Schlüssel des Eintrags der gelöscht werden soll

### ressources/scripts/ema.formgenerator.js

Der Formgenerator ermöglicht die einfache Erstellung von Formularen mit einem standardisierten Design. Über öffentliche Funktionen können Header, Footer und Zeilen des Formulars jeweils mit einem Befehl erstellt werden. Generell ist der Aufbau von Formularen dreispaltig, in der ersten Spalte steht rechtsbündig das Label, in der mittleren Spalte linksbündig das Feld oder der sonstige Inhalt und in der letzten Spalte ebenfalls linksbündig eventuelle Fehlermeldungen.

Öffentliche Funktionen:

* generateHeader(headerText, headerLabel, firstColValue, firstColClass, lastColValue, lastColClass)
  + Diese Funktion erstellt eine Headerzeile für das Formular
  + headerText: Text, der in der mittleren Spalte angezeigt wird. Dieser Text ist fix und wird nicht wie das Label übersetzt
  + headerLabel: Name des Labels in den Sprachdateien, beim Erstellen des Formulars wird der passende Textbaustein entsprechend der gewählten Sprache ermittelt, weiters könnte auch eine CSS Klasse mitübergeben werden um das Aussehen zu verändern
  + firstColValue: analog zu headerText kann ein Fixtext für die erste Spalte übergeben werden
  + firstColLabel: analog zu headerLabel kann auch hier ein Label übergeben werden
  + lastColValue: auch für die 3. Spalte kann eine Value übergeben werden, dabei kann es sich auch um HTML handeln, um z.b. einen Button anzuzeigen, der die Zeilen darunter ein/ausblendet
  + lastColLabel: analog zu headerLabel
* generateFooter(showCancel, showReset, functionReset, showOkButton, labelOkButton, functionOkButton)
  + Mit dieser Funktion wird eine Fusszeile erstellt, in der max. 3 Buttons zur Verfügung stehen. Das Label der ersten beiden Buttons ist immer gleich, beim ersten Button kann auch ie Funktion nicht geändert werden. Nur beim 3. Button kann Label und Funktion gesetzt werden.
  + showCancel: soll der Cancel Button geszeigt werden true/false
  + showReset: soll der Reset Button gezeigt werden true/false
  + function Reset: welche Funktion soll bei Reset aufgerufen werden z.B. ema.inventur.lvs.inventurbeleg.resetFields()
  + showOkButton: Soll der OK Button gezeigt werden true/false
  + labelOkButton: Name des Labels für den OK Button
  + functionOkButton: welche Funktion soll durch den OK Button aufgerufen werden z.B. ema.inventur.lvs.inventurbeleg.saveCount()
* generateRow(fieldType, fieldId, fieldValue, jsEvent, disabled, labelValue, labelClass, errorValue, errorLabelClass, showErrorDiv, maxLength)
  + Mit dieser Funktion kann eine Zeile in einem Formular erstellt werden
  + fieldType: steuert, welche Art von Zeile erstellt wird. Erlaubte Werte sind:
    - empty: Zeile ohne Feld nur mit Text
    - emptylow: Zeile ohne Feld nur mit Text mit Höhe 20
    - text: Zeile mit Textfeld
    - textlow: Zeile mit Textfeld und Höhe 20
    - textsmall: Zeile mit kleinerem Textfeld
    - password: Zeile mit Passwortfeld
    - date: Zeile mit Datumsfeld
    - datelow: Zeile mit Datumsfeld und Höhe 20
    - number: Zeile mit Zahlenfeld
  + fieldId: HTML Id des Feldes – die Id eines Feldes muss in der gesamten Applikation eindeutig sein!
  + fieldValue: Wert des Feldes oder der Zeile
  + jsEvent: Event der im Feld ausgeführt werden soll – der Flexibilität halber wird hier z.B. onchange=”ema.inventur.lvs.inventurbeleg.doonchangestuff()” übergeben
  + disabled: ist das Feld aktiv (true/false)
  + labelValue: Value der Spalte Label
  + labelClass: Name des Labels
  + errorValue: Value der Error Spalte
  + errorLabelClass: Name des Errorlabels
  + showErrorDiv: soll das ErrorDiv angezeigt werden? Im Normalfall wird das Div erst im Fehlerfall eingeblendet.
  + maxLength: maximal erlaubte Zeichen
* generateSelectRow(fieldId, fieldValue, selection, jsEvent, disabled, labelValue, labelClass, errorValue, errorLabelClass, showErrorDiv)
  + Diese Funktion erstellt eine Zeile mit einem Select/Dropdownfeld
  + fieldId: HTML Id des Feldes – die Id eines Feldes muss in der gesamten Applikation eindeutig sein!
  + fieldValue: Wert des Feldes oder der Zeile
  + selection: JS Object mit den möglichen Werten des Feldes
  + jsEvent: Event der im Feld ausgeführt werden soll – der Flexibilität halber wird hier z.B. onchange=”ema.inventur.lvs.inventurbeleg.doonchangestuff()” übergeben
  + disabled: ist das Feld aktiv (true/false)
  + labelValue: Value der Spalte Label
  + labelClass: Name des Labels
  + errorValue: Value der Error Spalte
  + errorLabelClass: Name des Errorlabels
  + showErrorDiv: soll das ErrorDiv angezeigt werden? Im Normalfall wird das Div erst im Fehlerfall eingeblendet.
* generateCheckboxRow(fieldId, fieldValues, selection, jsEvent, disabled, labelValue, labelClass, errorValue, errorLabelClass, showErrorDiv)
  + Diese Funktion erstellt eine oder mehrere Zeilen mit Checkboxfeldern
  + fieldId: HTML Id des Feldes – die Id eines Feldes muss in der gesamten Applikation eindeutig sein!
  + fieldValues: selektierte Werte
  + selection: JS Object mit den möglichen Werten des Feldes
  + jsEvent: Event der im Feld ausgeführt werden soll – der Flexibilität halber wird hier z.B. onchange=”ema.inventur.lvs.inventurbeleg.doonchangestuff()” übergeben
  + disabled: ist das Feld aktiv (true/false)
  + labelValue: Value der Spalte Label
  + labelClass: Name des Labels
  + errorValue: Value der Error Spalte
  + errorLabelClass: Name des Errorlabels
  + showErrorDiv: soll das ErrorDiv angezeigt werden? Im Normalfall wird das Div erst im Fehlerfall eingeblendet.
* generateDateTimeRow(fieldId, dateValue, timeValue, jsEvent, disabled, labelValue, labelClass, errorValue, errorLabelClass, showErrorDiv)
  + Diese Funktion erstellt eine Zeile mit einem Datumsfeld und einem Zeitfeld
  + fieldId: HTML Id des Feldes – die Id eines Feldes muss in der gesamten Applikation eindeutig sein!
  + dateValue: Wert des Datumsfeldes
  + timeValue; Wert des Zeitfeldes
  + jsEvent: Event der im Feld ausgeführt werden soll – der Flexibilität halber wird hier z.B. onchange=”ema.inventur.lvs.inventurbeleg.doonchangestuff()” übergeben
  + disabled: ist das Feld aktiv (true/false)
  + labelValue: Value der Spalte Label
  + labelClass: Name des Labels
  + errorValue: Value der Error Spalte
  + errorLabelClass: Name des Errorlabels
  + showErrorDiv: soll das ErrorDiv angezeigt werden? Im Normalfall wird das Div erst im Fehlerfall eingeblendet.
* generateButtonRow(fieldId, fieldValue, jsEvent, disabled, labelValue, labelClass, errorValue, errorLabelClass, showErrorDiv, buttonText, buttonFunction)
  + Diese Funktion erstellt eine Zeile mit einem Textfeld und einem Button
  + fieldId: HTML Id des Feldes – die Id eines Feldes muss in der gesamten Applikation eindeutig sein!
  + fieldValue: Wert des Textfeldes
  + jsEvent: Event der im Feld ausgeführt werden soll – der Flexibilität halber wird hier z.B. onchange=”ema.inventur.lvs.inventurbeleg.doonchangestuff()” übergeben
  + disabled: ist das Feld aktiv (true/false)
  + labelValue: Value der Spalte Label
  + labelClass: Name des Labels
  + errorValue: Value der Error Spalte
  + errorLabelClass: Name des Errorlabels
  + showErrorDiv: soll das ErrorDiv angezeigt werden? Im Normalfall wird das Div erst im Fehlerfall eingeblendet.
  + buttonText: Beschriftung des Buttons
  + buttonFunction: Funktion, die beim Click auf den Button ausgeführt werden soll
* generateDialog(dialogId, dialogContent)
  + Diese Funktion erstellt einen modalen Dialog, der mit der Funktion “ema.shell.showDialog(dialogId)” ein/ausgeblendet werden kann. Clickt man in den Dialog wird dieser wieder ausgeblendet.
  + dialogId: ID des zu erstellenden Dialogs
  + dialogContent: Inhalt des Dialogs
* generateActionDialog(dialogId, dialogContent, label, cssClass)
  + mit dieser Funktion kann ein Dialog erstellt werden, der weitere Benutzeraktionen erfordert. Im Gegensatz zu generateDialog muss hier ein Schliessen-Button betätigt werden um den Dialog wieder zu schliessen.
  + dialogId: ID des zu erstellenden Dialogs
  + dialogContent: Inhalt des Dialogs
  + label: Name des Labels des Dialogs
  + cssClass: CSS Klasse des ContentDivs. Kann z.b. dazu benutzt werden die Breite des Dialogs zu steuern
* generateNumberFieldRow(fieldId, fieldValue, kommastellen, jsEvent, disabled, labelValue, labelClass, errorValue, errorLabelClass, showErrorDiv)
  + Mit dieser Funktion kann ein Nummernfeld erstellt werden, das nur eine bestimmte Anzahl an Kommastellen erlaubt.
  + fieldId: HTML Id des Feldes – die Id eines Feldes muss in der gesamten Applikation eindeutig sein!
  + fieldValue: Wert des Feldes oder der Zeile
  + kommastellen: Anzahl an Kommastellen, die im Feld erlaubt sind
  + jsEvent: Event der im Feld ausgeführt werden soll – der Flexibilität halber wird hier z.B. onchange=”ema.inventur.lvs.inventurbeleg.doonchangestuff()” übergeben
  + disabled: ist das Feld aktiv (true/false)
  + labelValue: Value der Spalte Label
  + labelClass: Name des Labels
  + errorValue: Value der Error Spalte
  + errorLabelClass: Name des Errorlabels
  + showErrorDiv: soll das ErrorDiv angezeigt werden? Im Normalfall wird das Div erst im Fehlerfall eingeblendet.

### ressources/scripts/ema.js

Diese Bibliothek enthält die initModule Methode, die beim Start von ema.html aufgerufen wird. Durch initModule wiederum werden die Funktionen ema.shell.configModule und ema.shell.initModule aufgerufen, die die weitere Initialisierung der Applikation durchführen.

### ressources/scripts/ema.listgenerator.js

Mit Hilfe dieses JS Files wird die Erstellung von Listen und Reports vereinfacht. Ähnlich wie im Formgenerator gibt es auch hier die Methoden generateHeader und generateFooter und für die Erstellung einer ListenZeile generateRow, zusätzlich gibt es aber auch die Methode initializeSortedList mit der eine ganze Liste erstellt werden kann.

Der Aufbau einer Listenzeile ist:

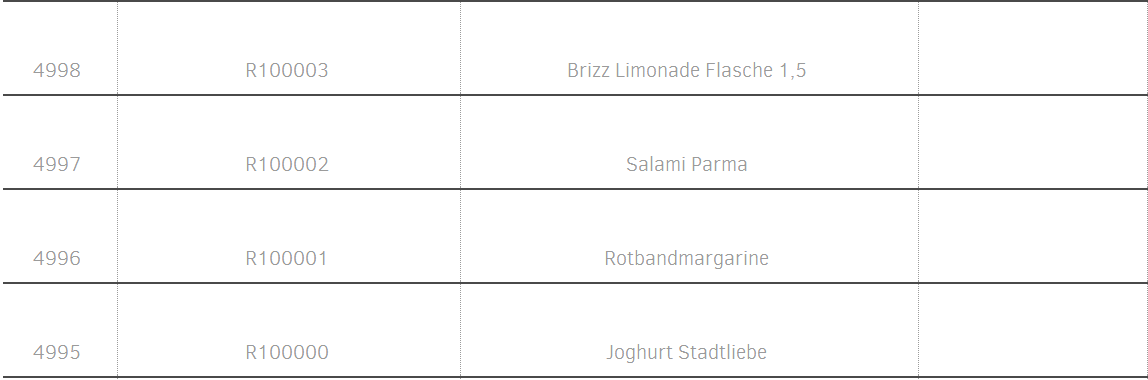
1. Fixe sichtbare Zeile
2. Optionale versteckte 2. Zeile
3. Optionale versteckte “Unterobjekte”
4. Optionaler zusätzlicher Inhalt (z.b. für Felder in der Liste)

Beispiele für Listen:

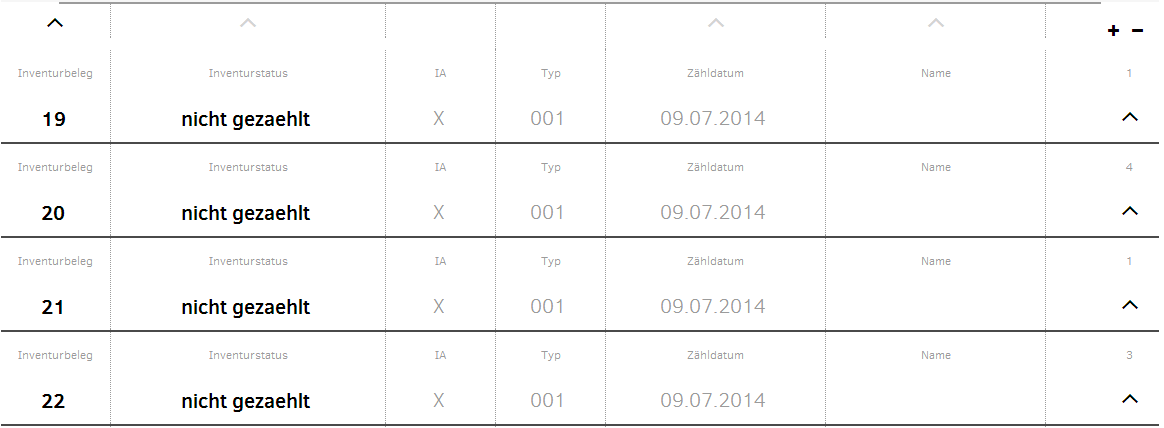
1. Liste in der die beiden ersten Spalten einen zusätzlichen Stil haben



1. Liste ohne versteckte Zeilen:



1. Sortierbare Liste mit versteckten Zeilen:



1. Aufgeklappte sortierbare Liste mit versteckten Zeilen:



Öffentliche Methoden:

* generateRow(rowId, columnWidth, columnLabel, columnValues, columnStyles, rowOnClick, hasHiddenSection, subCount, hiddenColumnWidth, hiddenColumnLabel, hiddenColumnValues, customHiddenContent, subColumns, drawLabels, alternateRowClass, rowNumber)
  + Mit dieser Methode kann eine Zeile in einer Liste erstellt werden. Eine Zeile besteht aus beliebig vielen Spalten, die zusammen max. 100% Breite haben dürfen, wird das überschritten wird eine Fehlermeldung beim Laden der Liste ausgegeben.
  + rowId: eindeutige Id um die Row ansprechen zu können
  + columnWidth: Array mit den Spaltenbreiten in %
  + columnLabel: Array mit den Labels für die Spalten
  + columnValues: Array mit den Werten für die Spalten
  + columnStyles: Array mit css Klassen, die den Content der Spalte, nicht aber den Label betreffen
  + rowOnClick: javascript, das im Div der Zeile eingefügt wird z.B. onclick=”ema.inventur.lvs.belegliste.selectRow(rowId)”
  + hasHiddenSection: Gibt es für diese Zeile einen versteckten Bereich (true/false)
  + subCount: Wieviele Subdatensätze gibt es – diese Zahl wird über dem Button angezeigt, mit dem der versteckte Bereich ein/ausgeblendet werden kann
  + hiddenColumnWidth: Array mit den Spaltenbreiten in %
  + hiddenColumnLabel: Array mit den Labels für die Spalten
  + hiddenColumnValues: Array mit den Werten für die Spalten
  + customHiddenContent: zusätzlicher Inhalt – dieser Inhalt muss schon in einer passenden Form angeliefert werden, er wird einfach in ein Div-Element mit 100% Breite eingefügt.
  + subColumns: Array mit den Subeinträgen. Jeder Arrayeintrag muss folgende Werte haben:
    - width: Array mit Breiten in %
    - labels: Array mit den Labels für die Spalten
    - values: Array mit den Werten für die Spalten
  + drawLabels: true/false – gibt an, ob Überschriften in den Spalten angezeigt werden
  + alternateRowClass: alternativer Stil für jede 2. Zeile
  + rowNumber: aktuelle Zeilennummer, wird für alternateRowClass benötigt
* generateSumRow(columnWidth, columnValues, hasHiddenSection)
  + Diese Funktion erstellt eine Summenzeile die nicht als Teil der Liste betrachtet wird, also nicht von Filter oder Sortierfunktionen verändert wird. Wird die Liste eingeschränkt so kann die angezeigte Summe falsch sein, wenn sie nicht im Modul entsprechend angepasst wird.
  + columnWidth: Array mit Breiten einer Zeile in %
  + columnValues: Array mit Werten für die Summenzeile
  + hasHiddenSection: Haben Zeilen in der Liste versteckte Bereiche (true/false)
* generateHeader(headerText, headerLabel, onChangeFunctino, divId)
  + Mit dieser Funktion wird ein Div erstellt, in dem die Liste eingefügt werden kann. Wird ein Headertext oder Label übergeben so wird eine Überschrift ausgegeben. Wird eine onChangeFunction übergeben, so wird ein Button neben der Überschrift angezeigt mit dem die Liste aktualisiert werden kann.
  + headerText: Text für die Überschrift
  + headerLabel: cssLabel für die Überschrift
  + onChangeFunktion: Funktion für die Aktualisierung der Liste
  + divId: Id des Divs in dem die Liste erstellt wird
* initializeSortedList(listData)
  + Diese Methode initialisiert eine sortierbare und/oder filterbare Liste. Da die dafür notwendigen Operationen (Wertevergleiche und Sortierungen) in Javascript erfolgen müssen ist es nicht ratsam diesen Listentyp bei sehr großen Datenmengen(mehrere 1000 Datensätze) zu verwenden, da sich dadurch die Performance der Applikation deutlich verschlechtern kann.
  + container: HTML Id des Container-Divs in dem die Liste angezeigt werden soll
  + listData: komplexes Objekt mit allem Informationen zur Liste:
    - container: Id des ContainerDivs in dem die Liste erstellt wird
    - preventDeletion: true/false – verhindert, dass die Liste beim Seitenwechsel gelöscht wird
    - sortType: Wie soll die Liste sortiert werden – mögliche Werte: asc, desc, ascNum, descNum, ascDate, descDate
    - sortColumn: Index der Spalte nach der sortiert werden soll
    - hasHiddenSection: gibt es versteckte Zeilen (true/false)
    - columnWidth: Array mit der Breite der Spalten in %
    - columnLabel: Array mit den Labels der Spalten
    - columnSearch: Soll die Liste filterbar sein (true/false)
    - columnSearchValues: Bereits eingegebene Werte in den Suchfeldern (Werden zur Laufzeit benötigt und sind initial voraussichtlich leer)
    - columnSearchIds: Ids der Suchfelder
    - sortFunction: JS Function für die Suche
    - searchFunction: JS Function für die Sortierung
    - hiddenColumnWidth: Array mit Breiten der Spalten in %
    - hiddenColumnLabels. Array mit Labels der Spalten
    - rows: hierbei handelt es sich wieder um Objekte die in einem Array übergeben werden – eine Row hat die Werte
      * rowId: Id der Zeile
      * columnValues: Array mit den Werten der Zeile
      * columnStyles: Array mit zusätzlichen Stilen
      * rowOnClick: javascript, das im Div der Zeile eingefügt wird z.B. onclick=”ema.inventur.lvs.belegliste.selectRow(rowId)”
      * subCount: Anzahl der Subdatensätze
      * hiddenColumnValues: Werte für die versteckte Zeile
      * customHiddenContent: HTML des Custom Content
      * subColumns: Array mit den Subeinträgen. Jeder Arrayeintrag muss folgende Werte haben:
        + width: Array mit Breiten in %
        + labels: Array mit den Labels für die Spalten
        + values: Array mit den Werten für die Spalten
* sortList(path, columnNo, drawRowsOnly)
  + Diese Funktion sortiert eine bestehende Liste und zeichnet sie neu
  + Path: Schlüssel unter dem die Listdaten in der listDataCollection zu finden sind
  + columnNo: Spaltennummer der Spalte nach der sortiert werden soll
  + drawRowsOnly: true/false – gibt an, ob nur die Zeilen oder auch die Suchfelder erstellt werden sollen – muss nur beim 1. Aufruf true sein
* searchList(path, elemId, elemValue)
  + Diese Methode schränkt eine Liste aufgrund von Werten in den Suchfeldern ein. Diese Methode muss nur selten direkt aufgerufen werden, da sie bei Eingabe eines Wertes in eines der Suchfelder automatisch aufgerufen wird.
  + Path: Schlüssel unter dem die Listdaten in der listDataCollection zu finden sind
  + elemId: ID des Suchfeldes
  + elemValue: Wert des Suchfeldes
* addRowToSortedList(row, path)
  + Mit dieser Funktion ist es möglich eine einzelne Zeile zu einer bestehenden Liste hinzuzufügen, die Liste wird anschliessend neu generiert und ist somit korrekt sortiert.
  + Row: das Zeilenobjekt der neuen Zeile
  + Path: Schlüssel unter dem die Listdaten in der listDataCollection zu finden sind
* editRowInSortedList(path, rowId, columnValues, hiddenColumnValues)
  + Mit dieser Funktion kann ein einzelner(oder auch mehrere) Wert€ in einer bestehenden Zeile einer Liste verändert werden. Die Liste wird anschliessend neu aufgebaut
  + Path: Schlüssel unter dem die Listdaten in der listDataCollection zu finden sind
  + rowId: id der Zeile
  + columnValues: neue Spaltenwerte der Zeile
  + hiddenColumnValues: neue versteckte Spaltenwerte der Zeile
* removeRowFromSortedList(rowId, Path)
  + Mit dieser Funktion kann eine Zeile einer Liste gelöscht werden
  + rowId: Id der Zeile die gelöscht werden soll
  + Path: Schlüssel unter dem die Listdaten in der listDataCollection zu finden sind
* deleteListData
  + Diese Funktion löscht sämtliche listDataCollection Einträge. Die Funktion dient dazu zu verhindern, dass der Hauptspeicher durch viele große Liste belastet wird. Durch die Property „preventDeletion“ kann das Löschen der Liste verhindert werden.

### ressources/scripts/ema.model.js

Model umfasst eine Sammlung von Hilfsfunktionen, die nicht direkt mit der Oberfläche interagieren.

Öffentliche Funktionen:

* saveToLocalStorage(key, value)
  + Speichert einen Wert im Localstorage – sowohl key als auch value werden dazu base64 encoded
  + key: der Schlüssel unter dem der Wert zu finden ist
  + value: der Wert der gespeichert werden soll
* loadFromLocalStorage(key)
  + liest einen Wert aus dem Lokalstorage und decoded ihn bevor er übergeben wird
  + key: Schlüssel des gewünschten Wertes
* startTimerTask()
  + startet einen Webworkertask, der automatisch jede Minute überprüft, ob die automatische Logoutzeit erreicht ist, wenn ja wird ein Logout ausgeführt und man muss dich erneut anmelden.
* terminateTimerTask()
  + beendet den Webworkertask
* checkLogoutTime()
  + prüft, ob die im Localstorage hinterlegte Logoutzeit bereits erreicht ist.
* setLogoutTime()
  + setzt die Logoutzeit im Localstorage auf die aktuelle Zeit – dies wird automatisch bei jedem Seitenwechsel durchgeführt
* loadLanguagePattern(newLanguage):
  + diese Funktion lädt ein Sprachfile und setzt ändert die Labels in der gesamten Applikation auf die neue Sprache
  + newLanguage: Kürzel der neuen Sprache z.B. de
* generateRequestUrl(internalName)
  + Erstellt die Url für einen Webserviceaufruf
  + internalName: interner Name des Funktionsbausteins laut Config
* generateSearchURL(table, searchVal)
  + Erstellt die Url für einen Webserviceaufruf der Eposuche
  + Table: Name der Table für die Eposuche
  + searchVal: Wert nach dem gesucht werden soll
* formatDateForSap(strDate)
  + Konvertiert ein Datum ins Format YYYY-MM-DD
  + strDate: String mit dem Datumswert
* formatTimeForSap(strTime)
  + Ergänzt die Sekunden bei einer Uhrzeit die nur 5 Stellen hat (hh:mm) und gibt hh:mm:ss zurück
  + strTime: String mit der Uhrzeit
* formatSapDateForDisplay(strDate)
  + konvertiert ein Datum vom Format YYYY-MM-DD oder YYYYMMDD ins Format DD.MM.YYYY
  + strDate: String mit dem Datumswert
* formatDateTimeForDisplay(newDate)
  + erstellt aus einem Datum einen String im Format DD.MM.YYYY HH:mm:SS
  + newDate: JS Datumsobjekt
* formatDateForDisplay(newDate)
  + formatiert ein Datumsobjekt und gibt einen String im Format DD.MM.YYYY zurück
  + newDate: JS Datumsobjekt
* formatTimeForDisplay(newDate)
  + Gibt die Uhrzeitkomponente eines Datums im Format hh:mm:ss zurück
  + newDate: JS Datumsobjekt
* generateCurrentDateTime(generateTime)
  + erstellt einen String mit dem aktuellen Datum und (optional) der aktuellen Zeit im Format DD.MM.YYYY hh:mm:ss
  + generateTime: true/false – soll auch die Uhrzeit erstellt werden?
* generateJSDate (dateString)
  + erwartet einen String im Format DD.MM.YYYY und erstellt daraus ein JS Dateobjekt
  + dateString: String mit dem Datumswert
* generateLeadingZeros(origNumber, digits)
  + erstellt aus einer Zahl einen String mit führenden Nullen
  + origNumber: die übergebene Zahl
  + digits: Anzahl der gewünschten Stellen
* getUriParameter(uri, paramName)
  + ermittelt aus einer URI einen bestimmten Parameter, dabei wird erst hinter dem Zeichen „#“ mit der Suche begonnen, Aufbau der Uri muss daher <http://www.epoconsulting.com#param1=test1&param2=test2> sein. Es wird nach base64 encodeden und nicht encodeden Parametern gesucht.
  + uri: die Uri in der nach dem Parameter gesucht werden soll
  + paramName: Name des Parameters
* getUri(uri)
  + liefert die Uri ohne Parameter zurück – als Trennzeichen wird „#“ erwartet.
* getDecodedUrl(uri)
  + liefert die Uri mit Parametern zurück, wobei die Parameter decoded werden, wenn sie base64 encoded waren
* getHostname(uri)
  + liefert den Hostname aus der übergebenen Uri zurück
* formatNumberForDisplay(numberValue)
  + Formatiert eine Zahl für die Ausgabe – es wird das Komma „.“ durch „,“ ersetzt
* formatCurrencyForDisplay(numberValue)
  + Formatiert eine Zahl als Währungsbetrag mit 2 Nachkommestellen und Tausenderpunkten
* formatNumberForSAP(numberValue)
  + stellt sicher, dass numberValue eine Zahl ist, ist das nicht der Fall wird 0 übergeben
* validateNumericInput(fieldId, kommastellen)
  + prüft, ob eine in einem Inputfeld eingegebene Zahl nicht zu viele Nachkommastellen hat und blendet im Fehlerfall einen Error neben dem Feld ein.
* formatSAPDateForInputField(strDate)
  + konvertiert einen Datumstring ins Format YYYY-MM-DD
* replaceSpecialCharacters(textStr)
  + ersetzt die Zeichen „!“, „&“ und „#“ in einem Textstring
* isNumber(n)
  + Prüft ob ein String eine Zahl ist oder nicht
* generateAuthHeader(xhr, addLogonInformation)
  + generiert die Headerinformationen für SAP Authentifizierung oder für Basic Authentifizierung für Webservice Aufrufe
  + xhr: Request Object
  + addLogonInformation: true/false – sollen Headerinformationen hinzugefügt werden oder nicht
* generateRequestObj(request)
  + wandelt ein Requestobject in einen json-String um

### ressources/scripts/ema.selectiongenerator.js

Der Selectiongenerator dient zur Erstellung eines Selektionsschirms ähnlich wie in SAP. Über eine Config wir gesteuert, welche Felder der Selektion befüllt werden können und wie sich diese verhalten sollen. Gespeicherte Selektionen aus SAP erden abgefragt und können vom Benutzer ausgewählt werden, wodurch die entsprechenden Felder mit den gespeicherten Werten automatisch erstellt werden. Mit Ausnahme der Function generateNewSearch muss keine andere Function direkt aufgerufen werden, sie sind nur deshalb öffentlich, damit sie von Buttons im Formular aufgerufen werden können.

Öffentliche Funktionen:

* generateNewSearch(formDefinition, container, variantService, resultlistPath, labelHeader, labelButton)
  + Mit dieser Funktion wird eine neue Selektion erstellt
  + formDefinition: ist ein Array mit je einem Object pro verfügbarer Zeile – die Properties der Objects sind
    - fieldId: eindeutige Id des Feldes – wird mit einer Laufnummer versehen, damit auch mehrere Zeilen noch eindeutig sind
    - fieldType:
      * text: Textfeld
      * date: Datumsfeld
      * checkbox: Checkbox – nur bei kind „P“ möglich
    - required: true/false – ist true gesetzt muss zumindest eine Zeile dieses Feldes ausgefüllt werden
    - label: Labelclass die vor dem Feld steht
    - fieldLabel: Bezeichnung für Checkbox-Felder hinter der Checkbox
    - kind:
      * S: Selektion – hier sind mehrere Zeilen pro Feld möglich
      * P: Parameter – hier ist nur ein Wert möglich
    - Variantid: ID unter der Werte in der Selektion aus SAP zu finden sind
    - Sapid: ID unter der der Wert im json an SAP übergeben werden muss
  + Container: ID des Containers in dem die Selektion erstellt werden soll
  + variantService: Name des Services mit dem die Varianten abgefragt werden können
  + resultlistPath: Pfad des js-Files das die Liste erstellt
  + labelHeader: CSS Class für die Überschrift
  + labelButton: CSS Class für den Button „Weiter“ im Footer
* generateNewRow(fieldId, resultlistPath)
  + Diese Funktion erstellt eine neue Selektionszeile für ein Feld
  + fieldId: ID des Feldes
  + resultlistPath: ID der Selektion
* deleteRow(rowId)
  + diese Funktion löscht eine Zeile aus der Selektion
  + rowId: ID der Zeile die gelöscht werden soll
* checkOptChange(sourceField, fieldType)
  + diese Funktion wird ausgeführt, wenn die Option („=“, „[]“, „<“, „>“, „>=“, „<=“) für eine Zeile geändert wird und baut die Selektionsfelder für diese Zeile neu auf
* loadVariantCallback(json)
  + diese Funktion ist die Callbackfunktion für die Abfrage der gespeicherten Varianten. Sie erstellt ein Select-Feld in dem man die Varianten auswählen kann
* selectVariant(spirceFieldId, resultlistPath)
  + Mit dieser Funktion wird eine Variante ausgewählt und die Details der Variante aus SAP abgefragt
* selectVariantCallback(json)
  + Diese Funktion ist die Callbackfunktion für selectVariant. Sie wertet das übergebene Json aus und erstellt dessen Basis das Selektionsformular
* getSearchRequestObject(resultlistPath)
  + Diese Funktion wird vom Folgeschirm(der die Liste abfragt und anzeigt) aufgerufen und das Requestobject für die Liste zu bekommen
* resetFields(resultlistPath)
  + Diese Funktion setzt die Felder der Selektion zurück
* loadList(resultlistPath):
  + Diese Funktion prüft, ob die Selektionsfelder korrekt sind(Mussfeldprüfung), erstellt dann ein Requestobject und ruft die Folgeseite auf, die die tatsächliche Liste erstellt.

### ressources/scripts/ema.shell.js

In dieser Bibliothek finden sich viele der zentralen Funktionen der Applikation, sie ist damit der eigentliche Kern der EMA Anwendungen. Alle zentralen Parameter werden im ema.shell Objekt gespeichert, einige davon sind durch spezielle Funktionen abrufbar oder sogar änderbar.

Öffentliche Funktionen:

* handleCustomError(functionName, errorLabel, errorLevel)
  + diese Funktion erstellt ein Fehlerfenster für eine beliebige Fehlermeldung, die als Label in den Sprachdateien hinterlegt ist.
  + functionName: Name der Funktion, die den Fehler verursacht hat
  + errorLabel: Name des Labels dieses Fehlers
  + errorLevel: Stufe des Fehlers, mögliche Werte: e = rot, w = gelb, i oder s = grün
* handleError(functionname, error, errorLevel)
  + Diese Funktion gibt einen Fehler aus, der automatisch durch die Fehlerbehandlung erstellt wurde (try … catch in jeder Funktion)
  + functionName: Name der Funktion, die den Fehler verursacht hat
  + error: JS Fehlerobjekt
  + errorLevel: Stufe des Fehlers, mögliche Werte: e = rot, w = gelb, i oder s = grün
* handleWSError(functionName, xhr, status, errorThrown, errorLevel)
  + Diese Funktion gibt einen Fehler aus, der von einem WebserviceCall verursacht wurde
  + functionName: Name der Funktion, die den Fehler verursacht hat
  + xhr: Fehlerobejct von Jquery
  + status: Status der von Jquery zurückgegeben wurde
  + errorThrown: Fehlertext von Jquery
  + errorLevel: Stufe des Fehlers, mögliche Werte: e = rot, w = gelb, i oder s = grün
* hideError()
  + löscht angezeigte Fehler
* configModule()
  + Lädt das ConfigFile ein und speichert die Werte in Variablen
* initModule(container)
  + initialisiert die Applikation
* getLogoutminutes()
  + liefert auf wieviele Minuten die Logoutzeit eingestellt ist
* getKeyValueList(keyvaluetype)
  + liefert die Liste von Schlüsselwerten eines Typs zurück z.B. LVS\_STAT
* getKeyValue(keyvaluetype, keyvalue)
  + liefert einen bestimmten Schlüsselwert eines Typs zurück z.B. Status Offen des Typs LVS\_STAT
* getLanguageTextString(labelCategory, labelName)
  + liefert einen Textbaustein einer bestimmten Kategorie aus dem aktiven Übersetzungsfile – z.B. um eine bestimmte Fehlermeldung anzuzeigen.
* getLanguagesOptions()
  + liefert alle möglichen Sprachen zurück
* setSelectedLanguage()
  + setzt die Variable selected\_language
* setUserLoggedOn()
  + ändert die Variable isUserLoggedOn auf true oder false
* getStateMapValue(statemapValue)
  + liefert einen bestimmten stateMap Wert zurück
* getOnlineStatus()
  + liefert den aktuellen Onlinestatus zurück
* getConfigMapConfigValue()
  + liefert den Abschnitt „Config“ aus der configMap
* getConfigMapIndexedDb()
  + liefert den Abschnitt „IndexedDb“ aus der configMap
* getContentContainer()
  + liefert die Id des ContentContainers
* addContentDiv(divName, divId)
  + fügt einen Eintrag in die Liste contenIdList hinzu – diese Liste dient dazu ContentDivs aufgrund des divName wiederfinden und anzeigen zu können
  + divName: Name des Divs
  + divId: Id des Divs
* addOnloadFunction(divName, onloadFunction)
  + fügt eine Function in die Liste onloadFunctionList hinzu. Hier hinterlegte Funktionen werden ausgeführt, wenn das entsprechende Div eingeblendet wird.
  + divName: Name des Divs
  + onloadFunction: function die ausgeführt werden soll (nicht nur der Name als String)
* addOnUnloadFunction(divName, onUnloadFunction)
  + fügt eine Function in die Liste onunloadFunctionList hinzu. Hier hinterlegte Funktionen werden ausgeführt, wenn das entsprechende Div ausgeblendet wird.
  + divName: Name des Divs
  + onUnloadFunction: function die ausgeführt werden soll (nicht nur der Name als String)
* addOnloadFocusField(divName, fieldName)
  + fügt einen Feldnamen in die Liste onloadFocusField hinzu. Hier hinterlegte Fehler bekommen den Focus, wenn das Div angezeigt wird.
  + divName: Name des Divs
  + fieldName: Name des Feldes das den Focus bekommen soll
* activateSaveConfirm(containerName)
  + Diese Funktion aktiviert eine Abfrage „Daten können Verloren gehen, wollen Sie fortfahren?“ beim Verlassen einer Seite
  + containerName: Name des Containerdivs der Seite
* activateLoadingAnimation(path, container)
  + Diese Funktion aktiviert eine Ladeanimation beim Laden einer Seite
* setMenuGenerated(isMenuGenerated)
  + Ändert das Flag stateMap.isMenuGenerated
* loadModule(sourcePath, navText)
  + Diese Funktion lädt ein Modul. Wurde das Modul bereits geladen so wird das entsprechende Div nur eingeblendet, wurde es noch nicht geladen so wird das entsprechende JS File nachgeladen, ein neues Div erstellt und dann angezeigt. Das Dashboard wird dabei ebenso aktualisiert wie die Breadcrumb Navigation.
  + sourcePath: Pfad des JS-Files (ohne .js)
  + navText: Bezeichnung die im Breadcrumbmenü anzeigt werden soll
* changeHash(sourcePath, navText)
  + Diese Funktion ändert die aktuelle URL, wodurch ein Event ausgelöst wird, der in weiterer Folge loadModule aufruft. Dieses Vorgehen ist notwendig um die Funktionsweise der Vorwärts/Rückwärts Tasten sicherzustellen. Module sollten daher immer über diese Funktion geladen werden!
  + sourcePath: Pfad des JS-Files (ohne .js)
  + navText: Bezeichnung die im Breadcrumbmenü anzeigt werden soll
* loadEmbeddedUri(sourcePath, navText)
  + Diese Funktion lädt eine externe URL in einem iFrame in der Applikation
  + sourcePath: Pfad des JS-Files (ohne .js)
  + navText: Bezeichnung die im Breadcrumbmenü anzeigt werden soll
* showHideDashboard()
  + Zeigt die zweite Zeile des Dashboards an bzw. blendet diese aus
* historyBack()
  + Diese Funktion geht einen Schritt in der Browserhistorie zurück und aktualisert die Breadcrumb-Navigation entsprechend
* logout()
  + führt einen Logout durch und löscht die entsprechednen Werte im Localstorage
* toggleOnline()
  + schaltet zwischen online und offline Modus um.
* showHideSection(sectionId)
  + blendet eine Section mit der übergebenen Id ein/aus und ändert den dazugehörigen Button, damit dieser das richtige Icon anzeigt
* showSections(sectionIds)
  + blendet eine Liste von Sections ein und ändert die dazugehörigen Buttons.
* hideSections(sectionIds)
  + blendet eine Liste von Sections aus und ändert die dazugehörigen Buttons.
* loadKeyValues()
  + liest Schlüsselwerte entsprechend der Config aus SAP oder aus dem Filesystem ein
* loadSapKeyValueCallBack(json)
  + Callbackfunktion für die Schlüsselwertverwaltung
* resetLanguage()
  + löscht den aktuellen Sprachstring aus den Variablen – wird ausgeführt bevor eine neue Sprache geladen wird
* setLanguage
  + speichert den aktuellen Sprachstring als Variable
* showDialog(elName, scroll)
  + zeigt ein modales Dialogfester an oder blendet es aus
  + elName: Id des Dialogfensters
  + scroll: soll zum Beginn der Seite gescrollt werden true/false
* resetReqSourcePath
  + löscht die Variablen reqSourcePath und reqNavText – diese Variablen werden verwendet um einen direkten Link auf ein Modul zu ermöglichen, auch wenn vorher ein Login erforderlich ist.
* resizeCanvas(elemId)
  + Berechnet die Größe eines Canvas Elements (z.B. Unterschriftsfeld)
* updateSyncScreen()
  + aktualisiert den Bildschirm Synchronisation
* gotoMenu()
  + lädt das Menü
* showLoadingAnimLayer()
  + öffnet die Ladeanimation in einem eigenen Layer
* hideLoadingAnimLayer()
  + blendet die Ladeanimation wieder aus
* showHideBreadcrumb(hide)
  + zeigt bzw. blendet das Breadcrumb Menü ein/aus
  + hide: true/false – es soll ausgeblendet werden

### ressources/scripts/minutetimer.js

Dieses File wird von einem Webworker verwendet, der einmal pro Minute prüft, ob die Logoutzeit bereits erreicht ist.

## Externe Bibliotheken

### IndexedDBShim.min.js

### jquery-2.1.1.js bzw. jquery-2.1.1.min.js

### jquery-ui-1.10.4.custom.min.js

### modernizr.js

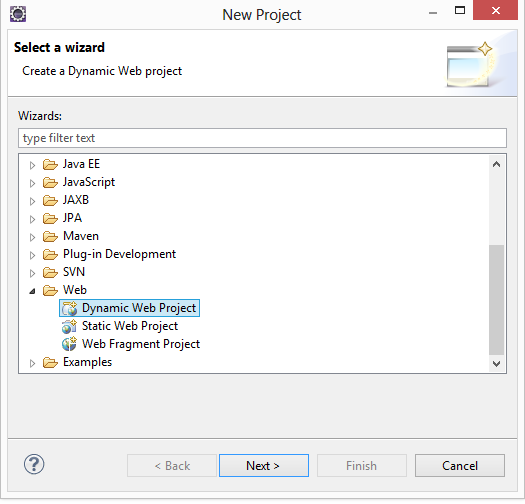
### sjl.min.js

### signature\_pad.min.js

# EMA Modulentwicklung

## Erstellen eines neuen Moduls

Die Erstellung eines neuen Moduls wird anhand des Moduls Inventur / LVS erklärt. Für die Entwicklung verwenden wir [Eclipse](#_Eclipse). Als Projekt erstellen wir „Dynamic Web Projects“.

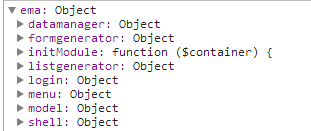


Beim Namen des Projektes ist einerseits darauf zu achten, dass der Name eindeutig ist, andererseits sollte er nicht über 9 Zeichen haben, da es sonst zu Problemen in SAP kommen kann.

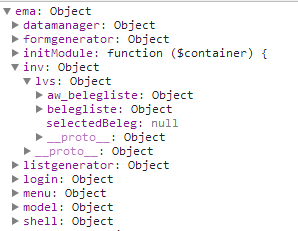
Um die Codequalität zu erhöhen verwenden wir das Eclipse Plugin [JSLint](#_JSLint).

### Festlegen des Namespace

Als erste Entscheidung muss definiert werden, wie der Namespace des Moduls lauten soll. Dabei ist es wichtig, dass es zu keinen Überschneidungen zwischen unterschiedlichen Modulen oder Core-Elementen von EMA kommt. Das Zentrale Objekt heißt „ema“, darunter finden sich alle Unterobjekte der Core-Elemente und Module. Im Debugger von Chrome sieht das Element folgendermaßen aus:



Wird ein Modul geladen, so wird das Objekt dynamisch erweitert:



Hier sieht man, dass unter „ema“ das Objekt „inv“ mit weiteren Unterobjekten hängt.

Um einen Überblick über die vorhandenen Objekte zu bekommen empfiehlt es sich den Debugger in Chrome oder anderen Browsern zu verwenden. Im Normalfall kann man die Developertools in Browsern mit F12 aufrufen.

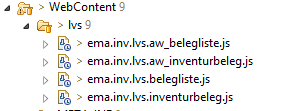
Der Namespace des neuen Moduls sollte eindeutig aber nicht zu lang sein wie z.B. inv für Inventur, tp für Transport oder ich für Instandhaltung. Gibt es darunter noch mehrere Untermodule kann man das durch eine weitere Gliederung strukturieren wie z.b. inv.lvs und inv.mm.

In diesem Modul-Objekt werden nun weitere Objekte für die einzelnen Formulare und Listen erstellt, wobei auf dieser Ebene jedes Objekt einem JS-File entspricht.

### Festlegen der Modulstruktur und der einzelnen Formulare

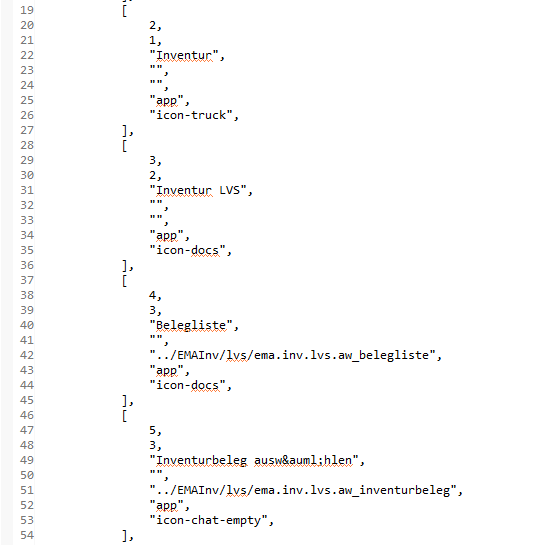
In diesem Schritt muss man sich überlegen, welche Formulare, Listen oder sonstige Seiten das Modul umfasst und wie diese für die Benutzer erreichbar sind. Will man, dass bestimmte Seiten im Menü zu finden sind so muss dieses entweder lokal in der menu.json oder im SAP angepasst werden.

Im Fall unseres Beispielprojekts werden vier Seiten erstellt:



Die Files haben bereits Namen, aus denen sich erkennen lässt wo in der Objekthierarchie sie zu finden sind. Die Files „aw\_belegliste“ und „aw\_inventurbeleg“ sollen über das Menü aufgerufen werden können, wobei noch 2 Menüebenen darüber („Inventur“ und „Inventur LVS“) implementiert werden sollen. Hierbei ist es wichtig, dass Dateinamen nicht länger als 40 Zeichen sind.

Die dafür notwendigen Einträge im menu.json sehen folgendermaßen aus:



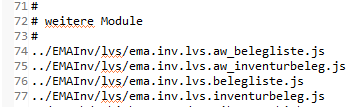
Den genauen Aufbau eines Menüeintrags können Sie hier nachlesen [menu.json](#_menu.json).

Damit nicht bei jedem neuen Modul oder der Änderung eines Moduls das gesamte Projekt aktualisiert werden muss werden einzelne Module getrennt in SAP eingespielt. Das hat zur Folge, dass das Menü erst in SAP eingerichtet werden kann und die darin angegebenen Pfade in jedem SAP-System unterschiedlichen sein können.

### ema.appcache anpassen

Sobald festgelegt ist, welche Dateien das Projekt umfasst können diese im Appcache eingetragen werden, damit sie beim Start der Applikation lokal gespeichert werden und das Modul offlinefähig wird. Damit ist allerdings erst die Ablauflogik der Seiten offline, damit die Applikation im Offlinemodus wirklich genutzt werden kann muss dies auch in der Entwicklung berücksichtigt werden, damit alle benötigten Daten vorhanden sind.

Die Einträge im File ema.appcache sehen folgendermaßen aus:



### Webservices und Synchronisation definieren

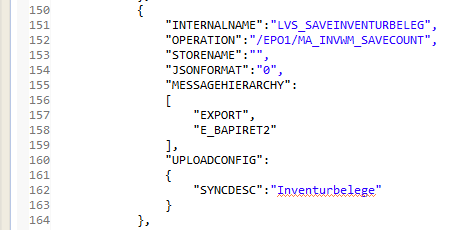
Als nächsten Schritt sollte man definieren, welche Webservices bzw. welche Funktionsbausteine in SAP aufgerufen werden sollen und ob diese gleich beim Start der Applikation aufgerufen werden sollen.

Die dafür notwendigen Anpassungen werden in der [config.properties](#_config.properties) durchgeführt. Im Fall unseres Moduls sehen die Einträge so aus:

* Inventurbelege laden: Dieser Baustein wird sowohl für einen einzelnen Inventurbeleg als auch für die Belegliste verwendet. Beim Start der Applikation werden alle Inventurbelege für Lagernummer 001 inklusive Details geladen und im Objectstore „inventurbelege“ gespeichert.

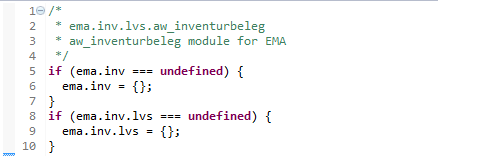


* Beleg speichern: Bei diesem Webservice werden beim Start keine Daten geladen, es dient nur dazu eine Zählung in SAP zu speichern. Daher haben wir eine Uploadconfig erstellt, wodurch Einträge im Offlinemodus im Objectstore „uploadsync“ gespeichert werden und später an SAP geschickt werden können.



### Aufbau eines JavaScript Files

#### Kopfbereich



In diesem Bereich wird die Objekthierarchie für dieses JS File geprüft und wenn nötig erstellt. Ist „aw\_inventurbeleg“ die erste Seite des Moduls die geladen wird, so werden sowohl „ema.inv“ als auch „ema.inv.lvs“ erstellt. Wird später die Seite „aw\_belegliste“ geladen, so sind diese Objekte bereits erstellt und müssen nicht neu erstellt werden.

#### Modulfunction

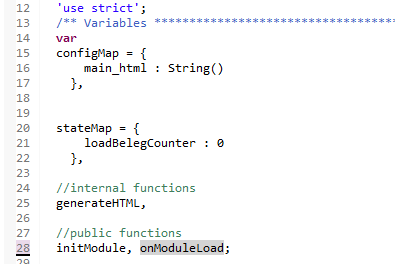


…



Im nächsten Schritt wird die Seite selbst als Objekt erstellt. Dem Objekt wird hierbei eine selbstausführende anonyme Function als Wert zugewiesen. Sämtlicher Code der Seite befindet sich nun in dieser Function.

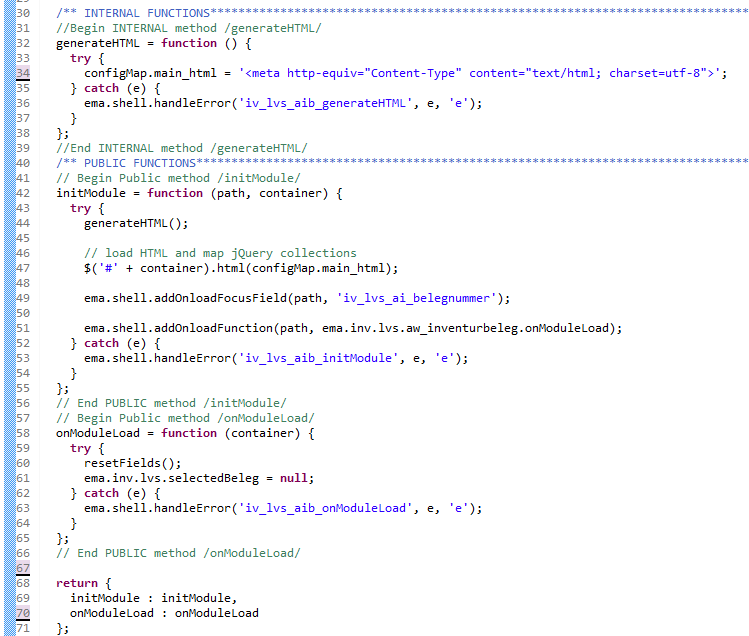
#### Variablenbereich



Im Variablenbereich werden alle Variablen und Funktionen der Seite deklariert. Der erste Befehl hier lautet ‘use strict‘, dadurch ist der Browser bei der Prüfung der JS Syntax strenger, was die Code Qualität heben soll.

Anschließend findet sich ein var Statement gefolgt von den Variablen, den internen Funktionen und den öffentlichen Funktionen, die auch von anderen Modulen aufgerufen werden können.

#### Functions und Return:



Im letzten und längsten Abschnitt finden sich die eigentlichen Funktionen der Seite. Dabei sollten der Übersichtlichkeit halber die internen und externen Funktionen getrennt werden. Externe Funktionen müssen zusätzlich im „return“ angegeben werden, damit sie von außen aufgerufen werden können.

Generell sind nur öffentliche Funktionen von anderen Seiten ansprechbar. Will man, dass auch Variablen von anderen Seiten gelesen oder geändert werden können so muss man entsprechende öffentliche Funktionen zu Verfügung stellen, die dies ermöglichen.

#### Erforderliche und optionale Standardfunktionen

Jede Seite benötigt die Funktion “initModule” mit den Parametern “path” und “container”. Beim erstmaligen Laden der Seite wird diese Funktion aufgerufen, dabei wird in path der Pfad der Seite übergeben und in container die ID des Div-Elements in dem die Seite angezeigt wird.

Der Parameter path kann verwendet werden, um eine onload Function und ein autofocus Feld zu registrieren.

Die onload Function wird im Gegensatz zur initModule Function bei jedem Öffnen der Seite ausgeführt und nicht nur beim ersten Öffnen. Um die Function zu registrieren muss man die Funktion

ema.shell.addOnloadFunction(path, function)

aufrufen. Als function muss hierbei die gewünschte Funktion übergeben werden, z.B. ema.inv.lvs.aw\_inventurbeleg.onModuleLoad.

Weiters ist es möglich ein Feld zu definieren das beim Öffnen der Seite den Focus bekommt. Dies ist notwendig, da die Html Option “autofocus” nur beim ersten Laden der Seite zieht, die Seite aber beliebig oft ein und ausgeblendet werden kann. Hierfür kann man die Funktion

ema.shell.addOnloadFocusField(path, Feldid)

genutzt werden.

#### Aufruf eines Webservices

Webservices zur Ermittlung oder Speicherung von Daten sollen in ema zentral über die Funktionen

ema.datamanager.doRequest

oder

ema.datamanager.requestRecordsByKey

erfolgen. Beide werden im Kapitel [ressources/scripts/ema.datamanager.js](#_ressources/scripts/ema.datamanager.) beschrieben. Grund dafür ist die standardisierte Verarbeitung der Daten sowie die automatische lokale Speicherung.

Die Funktion doRequest dient dazu eine größere Menge an Daten abzufragen (z.B. für eine Liste), bei jeder Abfrage wird der Requeststring und die Schlüssel aller gefundenen Datensätze in der lokalen Datenbank gespeichert, sodass die Abfrage in weiterer Folge auf im Offlinemodus durchgeführt werden kann.

Mit der Funktion requestRecordsByKey kann man zusätzlich zum Request auch noch eine Liste von Schlüsseln mitgeben nach denen im Offlinemodus gesucht werden kann. Diese Methode ist eher dazu gedacht einen bestimmten Datensatz zu laden.

Generell ist der Ablauf einer Datenabfrage:

Modul stellt Request zusammen

↓

Aufruf von doRequest bzw. requestRecordsByKey

↓

Automatischer Aufruf der Function doRequestSuccess durch das Webservice oder Ausgabe eines Fehlers

↓

Suchen nach dem Datenobjekts im abgefragten JSON

↓

Ermitteln des Schlüssels und definierter Indexwerte für jeden Einzeldatensatz

↓

Speichern jedes Einzeldatensatzes in der Indexeddb

↓

Aufruf der Callbackfunktion im Modul

Bei der Abfrage größerer Datenmengen ist es empfehlenswert den URL Parameter json\_format = 3 zu verwenden. Dadurch werden keine Bezeichnungen mehr übertragen und die Datensätze werden deutlich kleiner. Der Nachteil dabei ist, dass man den Aufbau des Datensatzes genau kennen muss, da alle Daten als Array und nicht mehr als Objekt mit Properties angeliefert werden.

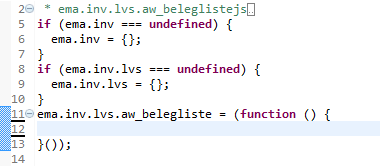
## Entwickeln der Auswahl für die Belegliste

Als erstes Formular für unser Modul entwickeln wir eine Auswahl für die Belegliste. Für dieses Formular müssen abgesehen von Schlüsselwerten keine Daten abgefragt werden und da Schlüsselwerte(wird beschrieben in [config.properties](#_config.properties) im Abschnitt keyvalues) automatisch ermittelt werden muss man sich an dieser Stelle keine Gedanken darüber machen.

Das fertige Formular soll so aussehen:

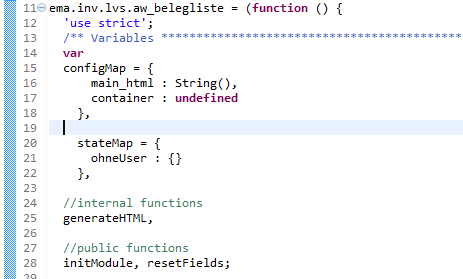


Nun erstellen wir ein File im Projekt EMAInv im Verzeichnis Webcontent/lvs mit dem Namen ema.inv.lvs.aw\_belegliste.js. Als erstes erstellen wir den Rumpf der Datei:



Damit stellen wir sicher, dass die Objekthierarchie für die Seite vorhanden ist und das Object aw\_belegliste beim Laden der Seite ausgeführt wird.

Nun ergänzen wir die Datei um den Variablenbereich:



In der configMap werden Variablen gespeichert, die zu Beginn belegt werden und dann nicht mehr geändert werden sollten. In der stateMap speichern wir Variablen die geändert werden können.

Die function generateHTML wird verwendet um das HTML der Seite dynamisch zu erstellen, ist es einmal erstellt so wird es in der configMap in main\_html gespeichert.

Die function initModule ist eine Standardfunktion die von der Applikation in jeder Seite erwartet wird und die als erstes aufgerufen wird.

Mit der function resetFields können Benutzer alle Auswahlfelder zurücksetzen.

Als Nächstes schreiben wir die function initModule:



Nachdem der Vorgabewert für die Checkbox „Ohneuser“ gesetzt wurde wird das HTML für die Seite erstellt und in der configMap gespeichert. Anschließend wird das HTML in den Container-Element geschrieben, das der function übergeben wurde.

In den Zeilen 72-86 erstellen wir ein onchange-Event für die Felder userzaehlung und ohneuser. Dieses Event soll sicherstellen, dass entweder das Feld userzaehlung befüllt oder das Feld ohneuser angehakt ist, aber nicht beide zur selben Zeit.

Sowohl die function als auch das onchange Event sind mit einer standardmäßigen Fehlerbehandlung versehen.

Nun wird die function generateHTML erstellt:

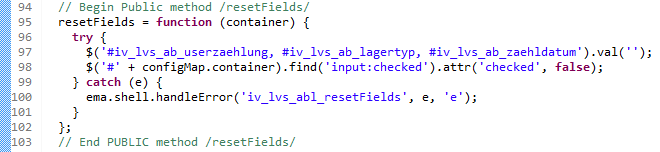


In dieser Funktion werden nur Methoden des Formgenerators ([ressources/scripts/ema.formgenerator.js](#_ressources/scripts/ema.formgenerato)) aufgerufen um die Seite aufzubauen. In generateFooter wird auf die nächste Seite verwiesen, in der dann die Daten für die Liste abgefragt werden.

Beim Erstellen des HTMLs ist es wichtig daran zu denken die Textbausteine in den Sprachfiles zu erstellen:



Als letzte Funktion muss resetFields erstellt werden:



Aufgrund der einfachen Syntax von jQuery ist das zurücksetzen mehrerer Felder schnell umgesetzt.

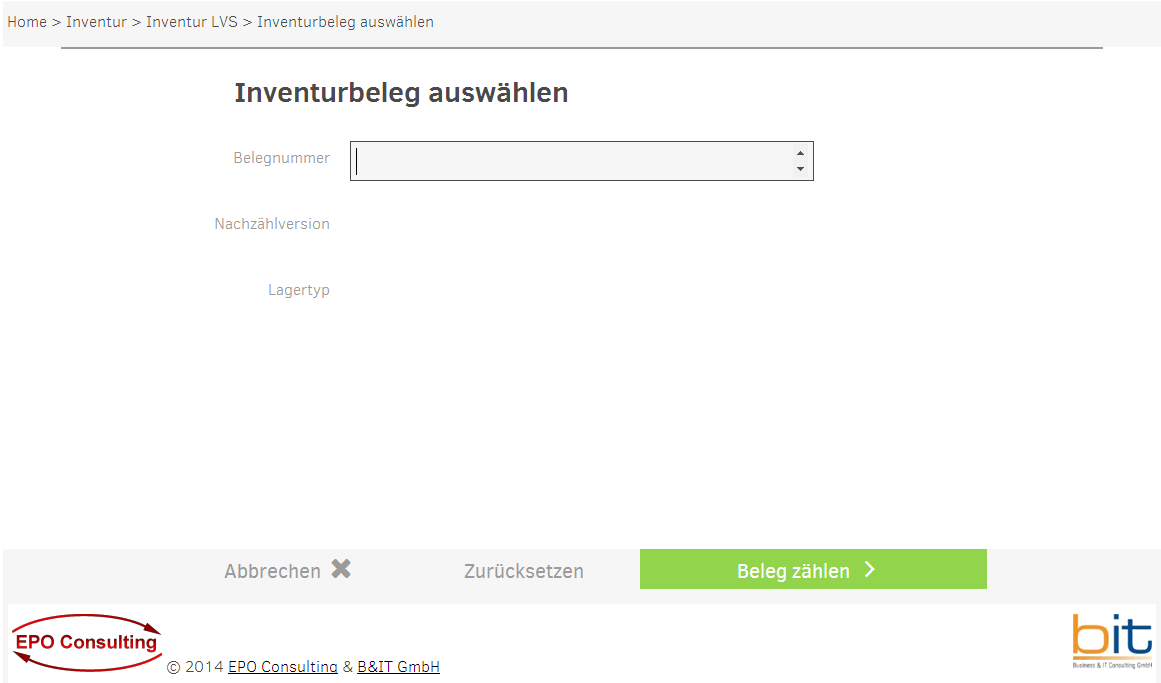
Abschließend müssen noch der return Block erstellt werden, damit die Public-functions von außen aufgerufen werden können:



Damit ist das erste Formular erstellt. Man kann von da aus zwar noch nicht weiter navigieren, der Aufruf dieser Seite aus dem Menü sollte aber schon funktionieren.

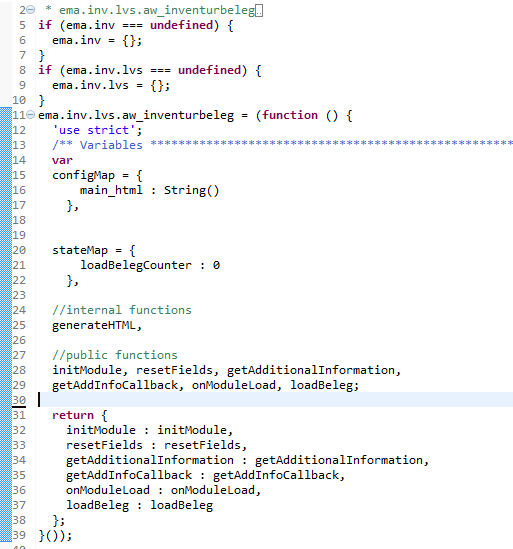
## Erstellen einer Auswahl für Inventurbelege

Diese Seite erlaubt es Benutzern durch Eingabe einer Belegnummer einen Beleg direkt zu laden. Das Formular soll dabei folgendermaßen aussehen:



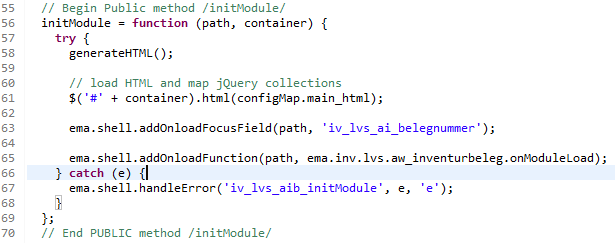
Analog zur ersten Seite erstellen wir auch hier ein neues Java Script File mit dem Namen ema.inv.lvs.aw\_inventurbeleg.js

Als Nächstes kommt der Rumpf des Files mit dem Abschnitt für Variablen und Returns:



Diese Seite benötigt schon deutlich mehr Functions, da wir beim Eingeben eines Wertes im Feld Belegnummer sofort den dazugehörigen Inventurbeleg laden wollen.

Nun erstellen wir die weiteren Funktionen:

* initModule:  
  

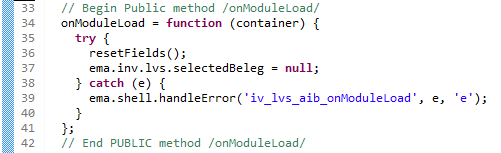
Nachdem das HTML erstellt wurde wird in diesem Fall wird sowohl ein Focus-Feld als auch eine onload-Funktion registriert.

* generateHTML:



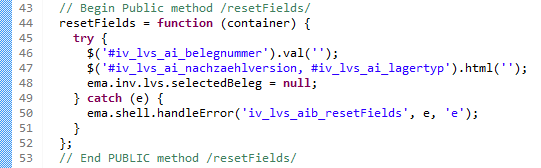
Diesmal wird nur ein Feld erstellt, dafür gibt es darunter noch 2 Zeilen in denen nur Text angezeigt wird. Im Footer kann auf der Seite die Folgeseite nicht direkt aufgerufen werden, da es sonst sein kann, dass noch kein Beleg geladen wurde. Daher wird hier eine weitere Funktion aufgerufen.

* onModuleLoad:



Im Gegensatz zu initModule wird diese Funktion bei jedem Einblenden der Seite ausgeführt. Sie setzt die Felder des Formulars zurück und löscht das Objekt selectedBeleg in dem der aktuelle Inventurbeleg gespeichert wurde.

* resetFields:



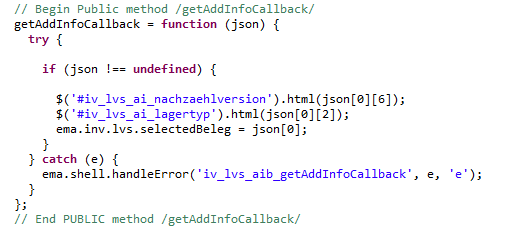
Hiermit wird das Feld, die zwei Zeilen darunter und das Objekt selectedBeleg gelöscht.

* getAdditionalInformation:



In dieser Funktion wird ein Request-Objekt erstellt und die Methode requestRecordsByKey aufgerufen. Damit werden erstmals Daten per Webservice abgefragt. Über die [config.properties](#_config.properties) wird gesteuert, wie die Daten dieses Aufrufs genau verarbeitet werden müssen.

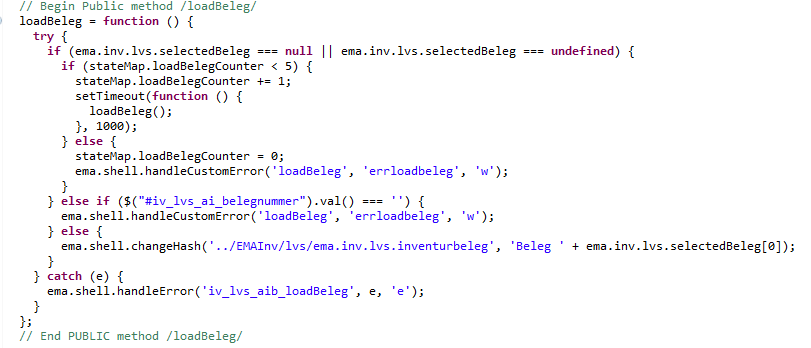
* getAddInfoCallback



Diese Funktion wird automatisch aufgerufen, nachdem das Webservice erfolgreich abgefragt wurde. Der Funktion werden die abgefragten Daten als json-Objekt übergeben. Fehler sollten bereits durch die Funktionen davor abgefangen worden sein, eine Prüfung auf json !== undefined ist in dieser Funktion aber dennoch sinnvoll.

Die übergebenen Daten werden hier nun in die zwei Zeilen unter dem Eingabefeld geschrieben und das Datenobjekt wird als selctedBeleg gespeichert.

* loadBeleg

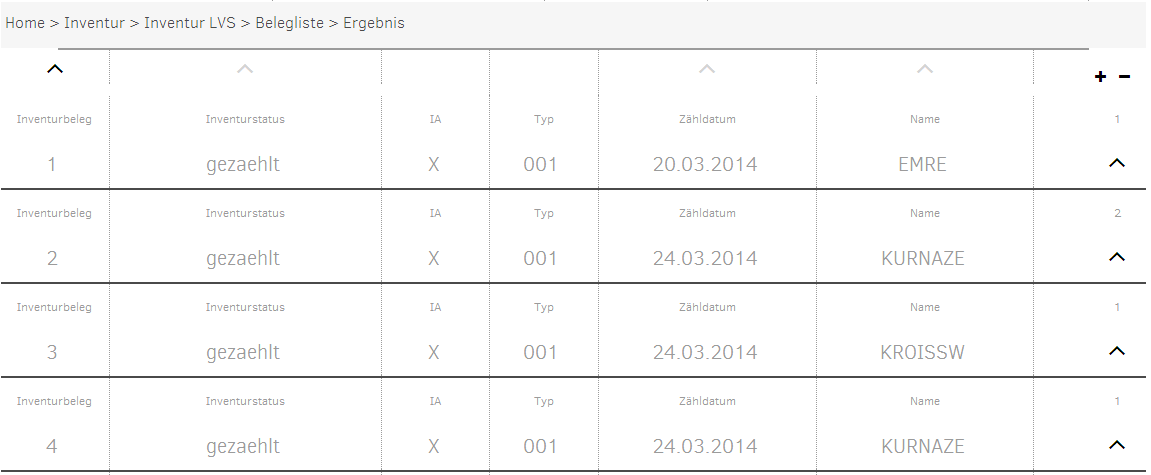


In dieser Funktion wird die Folgeseite Inventurbeleg geladen. Sollte die Funktion keinen selectedBeleg finden so wartet sie eine Sekunde und ruft sich erneut auf(bis zu 5 mal). Das ist notwendig, da Benutzer eine Belegnummer eingeben und dann direkt auf „Beleg zählen“ clicken können, wobei in manchen Fällen die Daten noch nicht abgefragt sind.

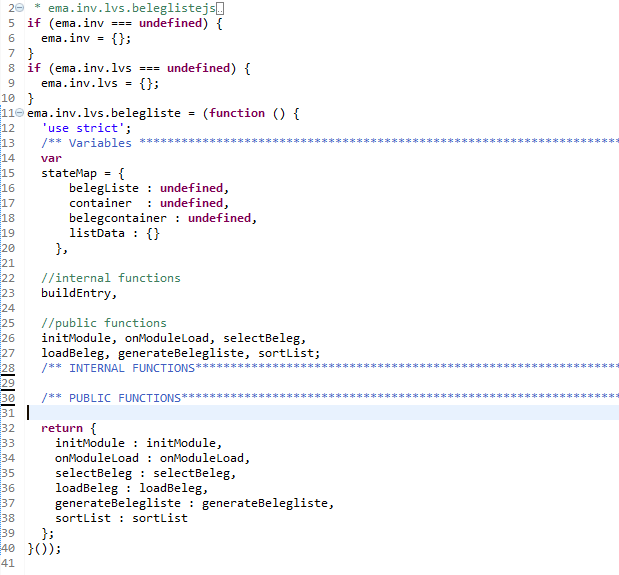
## Erstellen der Belegliste

Diese Seite zeigt eine Liste mit Belegen und dazugehörigen Positionen an. Jeder Eintrag in der Liste besteht aus einer sichtbaren Zeile und einem versteckten Abschnitt, der mittels Knopfdruck eingeblendet werden kann. Im versteckten Bereich ist eine weitere Zeile mit Daten des Haupteintrages und etwas eingerückt zusätzliche Zeilen mit Daten aus den Positionen.

Die Liste sieht im zugeklappten Zustand so aus:



Analog zu den vorigen Seiten erstellen wir auch hier das JavaScript File und darin den Rumpf:



In diesem Fall benötigen wir keine configMap, alle Daten sind veränderlich und werden daher in der stateMap gespeichert.

Dabei verwenden wir folgende Variable:

belegListe: eine JS Liste aller Belege die angezeigt werden

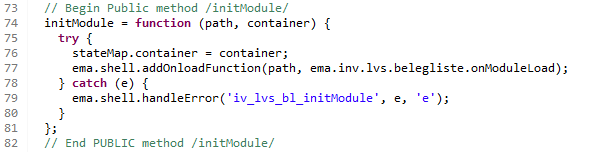
container: ID des Div-Elements in dem die Seite angezeigt wird

belegcontainer: ID des Div-Elements in dem die Liste angezeigt wird

listData: JS Objekt in dem alle Informationen(inkl. Spaltenwerte) gespeichert sind um die Liste aufbauen zu können

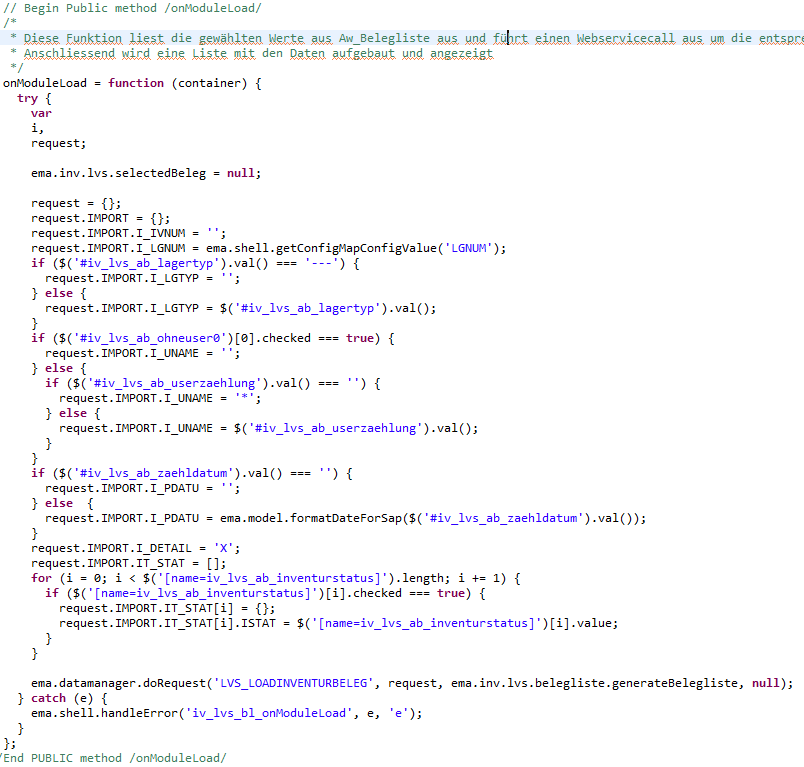
Nun erstellen wir die Funktionen, beginnend mit den Standardfunktionen:

* initModule



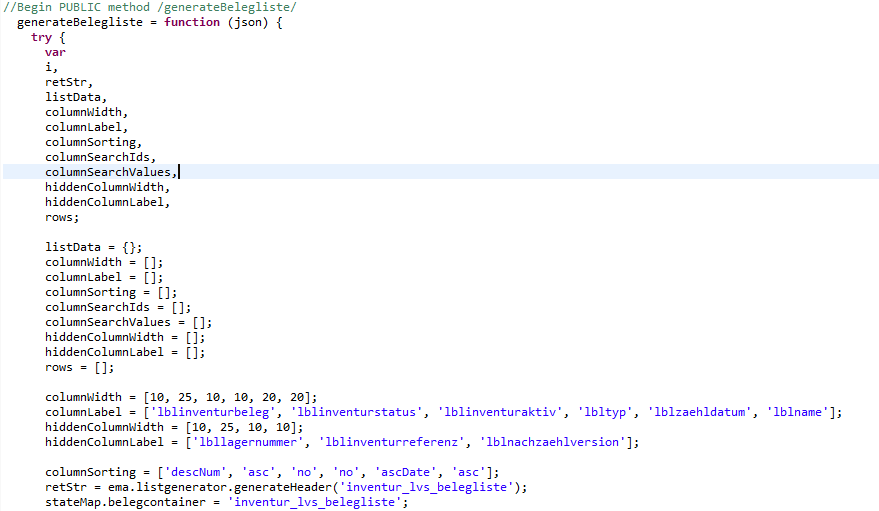
Hier wird nur die onload Funktion registriert

* onModuleLoad:



In dieser Funktion werden die gewählten Werte aus Auswahl Belegliste ausgelesen, das ist möglich da wir die ID der dortigen Felder kennen, anschliessend wird ein Request-Object erstellt und ein Webservice mit „doRequest“ aufgerufen. Als Callback Funktion übergeben wir generateBelegliste, wo die weitere Verarbeitung erfolgt.

* generateBelegliste





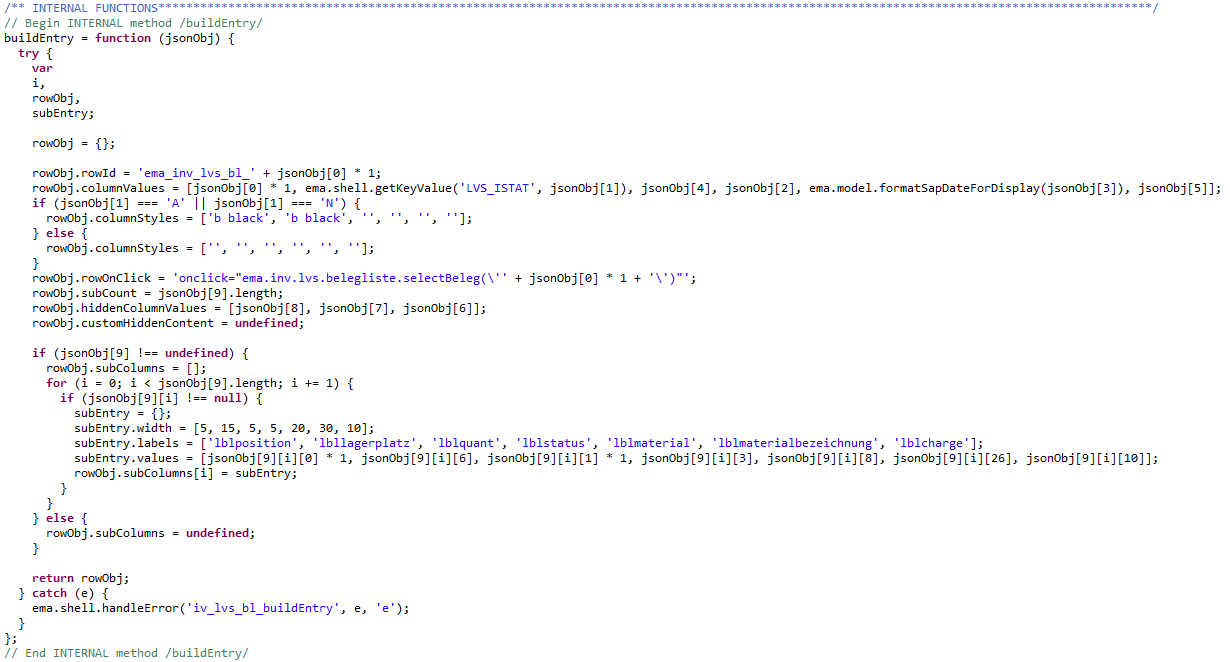
Diese Funktion wird nach dem Webservice als Callbackfunktion aufgerufen und bekommt die Daten als JS Objekt übergeben. Aufgrund der Config werden die Daten automatisch in der IndexedDB gespeichert, nachdem dies asynchron passiert läuft es je nach Datenmenge gleichzeitig mit dem Aufbau der Liste ab.

In diese Funktion werden Variablen und Arrays befüllt, die für die Liste benötigt werden, dabei werden alle Werte die in jeder Zeile verwendet werden nur einmal zentral definiert um das ListData Objekt möglichst klein zu halten. Nachdem die allgemeinen Daten geschrieben sind iteriert die Funktion über die abgefragten Datensätze und lässt aus diesen mit Hilfe der Funktion buildEntry einen rows-Eintrag erstellen.

Nach Abschluss dieser Funktion wird noch die Sortierung der Liste aufgerufen und damit die Anzeige erstellt.

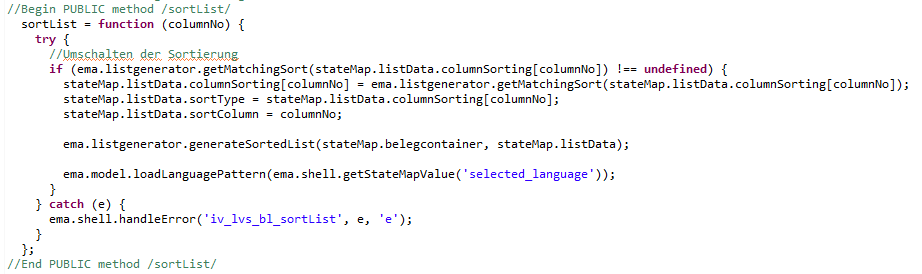
Vorteil dieser Funktion ist, dass die Liste als JS-Objekt erstellt wird und in weiterer Folge schnell und einfach gefiltert oder sortiert werden kann, ohne dass jedesmal die Werte aus den Datenobjekten ermittelt werden müssen.

* buildEntry



Dieser Funktion werden die Daten einer bestimmten Zeile übergeben und in ein Objekt geschrieben, dass dieser Zeile entspricht.

* sortList:



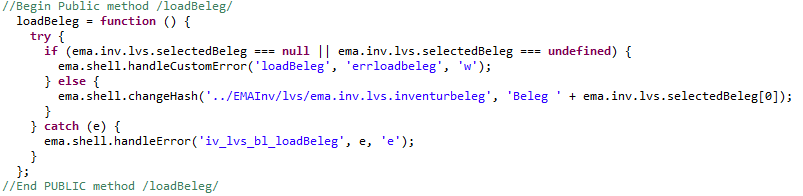
Diese Funktion ermittelt die korrekte neue Sortierung und ruft anschließend die Funktion generateSortedList auf, die die Sortierung tatsächlich durchführt und die Liste erstellt.

* selectBeleg:



Diese Methode wird in der Liste an das click-Event jeder Zeile gebunden, die angeclickte Zeile wird grün eingefärbt und der entsprechende Beleg wird als selectedBeleg gespeichert.

* loadBeleg:



Diese Methode prüft, ob ein selectedBeleg vorhanden ist, falls ja wird das Formular Inventurbeleg geladen.

## Erstellen des Formulars Inventurbeleg

Diese Seite ist die komplexeste des Moduls: Es handelt sich um ein Formular mit einer Liste, im Kopfbereich werden einige Daten ausgegeben und je nach Auswahl im Feld „Zählen von“ wird in der Tabelle darunter eine Zeile aufgeklappt.

Nachdem die Datenabfrage bereits von den Formularen vor dem Inventurbeleg durchgeführt wurde ist die Anzeige dieser Seite sehr schnell.

Das fertige Formular sieht so aus:



Bei den Positionen ist die Zahl in den Zeilen die noch nicht gezählt wurden fett gedruckt, clickt man in der Überschrift auf „3 noch offen“ so springt der Cursor in die erste offene Position.

Das Feld „Zählen von“ hat eine Autocomplete Funktion, die alle möglichen Lagerplätze und Materialien anbietet, wird ein Wert gewählt so stehen im Feld nur noch dazupassende Werte zur Verfügung, die Auswahl wird also dynamisch angepasst.

Aufgrund der Länge einiger Funktionen findet sich der Code dieser Seite unter [SourceCode von Inventurbeleg](#_SourceCode_von_Inventurbeleg).

Die Funktionen der Seite im Detail sind:

* initModule:

Hier werden wieder die onload-function und das Focusfeld gesetzt.

* onModuleLoad:

In dieser Funktion wird als erstes das HTML generiert, anschließend wird die Autocomplete-Funktion und die Event onchange und onkeyup an das Feld „Zählen von“ gehängt. Alle Felder mit der Klasse „lvs\_\_ib\_fc“ werden mit einem change-Event versehen, durch das die Anzahl der offenen Positionen berechnet wird und das die Positionsnummer bei gezählten Positionen auf normal(von fett) setzt.

* generateHTML:

Diese Funktion erstellt das HTML der Seite wobei im Gegensatz zur Belegliste die Liste der Positionen nicht sortierbar ist und direkt Zeile für Zeile erstellt wird. Dafür wird die Methode generateRow im Listgenerator aufgerufen.

* generateBeleg

Diese Methode konvertiert das JS Objekt aus selectedBeleg in eine Form die der Speicherbaustein versteht. Um die Datenmenge zu reduzieren werden die Belege mit Json\_Format 3 abgefragt, der Speicherbaustein benötigt aber Format 0.

* addEntryToList:

Diese Funktion verwaltet halle Listen für das Autocomplete im Feld „Zählen von“

* processAutocompleteResult:

Diese Funktion verarbeitet Eingaben im Feld „Zählen von“ und ändert die Auswahl im Feld entsprechend. Weiters klappt die Funktion Postionen auf/zu je nachdem was im Feld „Zählen von“ eingegeben ist.

* checkMatchingValues:

Hierbei handelt es sich um eine Hilfsfunktion um die Eingabe im Feld „Zählen von“ mit den möglichen Werten zu vergleichen. Gibt man im Feld z.b. 01-01-01 ein und gibt es einen Wert „Lagerplatz: 01-01-01“ so sollen diese als Übereinstimmung gewertet werden. Wertet die Funktion einen Wert als passend so wird processAutocompleteResult aufgerufen.

* checkCount:

Diese Funktion erstellt das Belegobjekt und prüft, ob zumindest eine Position gezählt wurde. Wenn ja wird das Unterschriftspad eingeblendet.

* saveCount:

wird vom Unterschriftspad aus aufgerufen und ruft das Webservice zum Speichern der Zählung auf.

* saveCountCallback:

Callbackfunktion nach dem Speichern. Diese Funktion blendet das Unterschriftspad wieder aus.

* expandAll:

Klappt alle Positionen auf

* collapseAll:

Klappt alle Positionen zu

* resetValue:

Löscht die gewählten Werte und setzt die Auswahl in „Zählen von“ zurück.

* showPosition:

Klappt die erste Position auf bei der die Positionsnummer fett ist.

# Ressourcen

## Eclipse

<https://www.eclipse.org/downloads/>

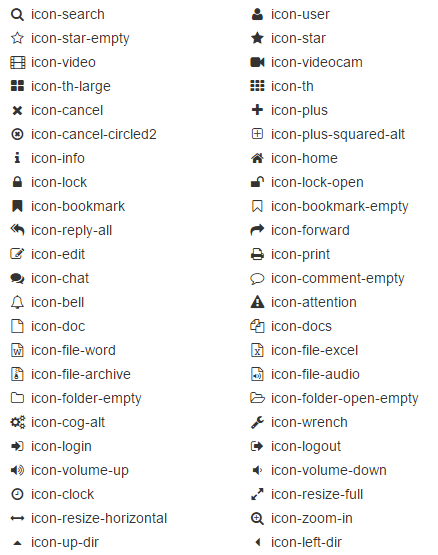
Eclipse IDE for Java EE Developers

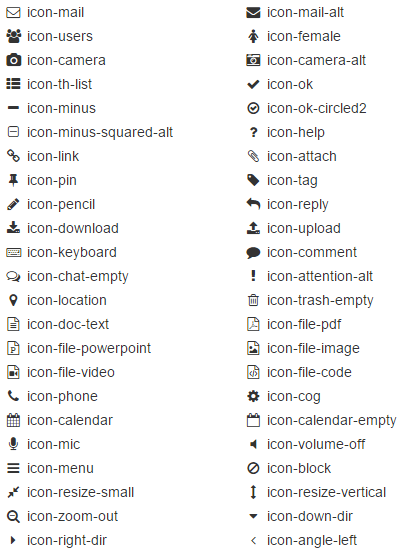
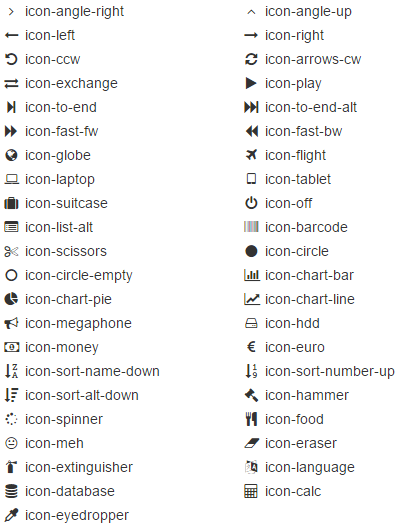
## JQuery

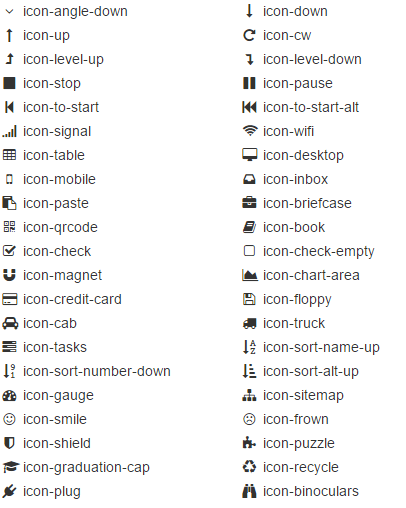
http://jquery.com/

## Icons

Es werden Icons von der Seite <http://fontello.com/> verwendet. Aktuell stehen folgende Icons zur Verfügung:





## Modernizr

<http://modernizr.com/>

## Can i use

<http://caniuse.com/>

## HTML 5

<http://www.w3schools.com/>

## Single Page Applications

<http://www.manning.com/mikowski/>

## JSON

<http://json.org/>

<http://www.softpedia.com/get/Programming/Other-Programming-Files/EPO-Transmitter.shtml>

## Use Strict

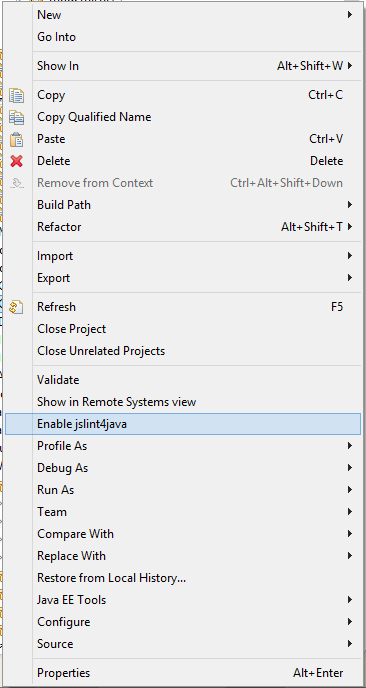
<http://www.w3schools.com/js/js_strict.asp>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions_and_function_scope/Strict_mode>

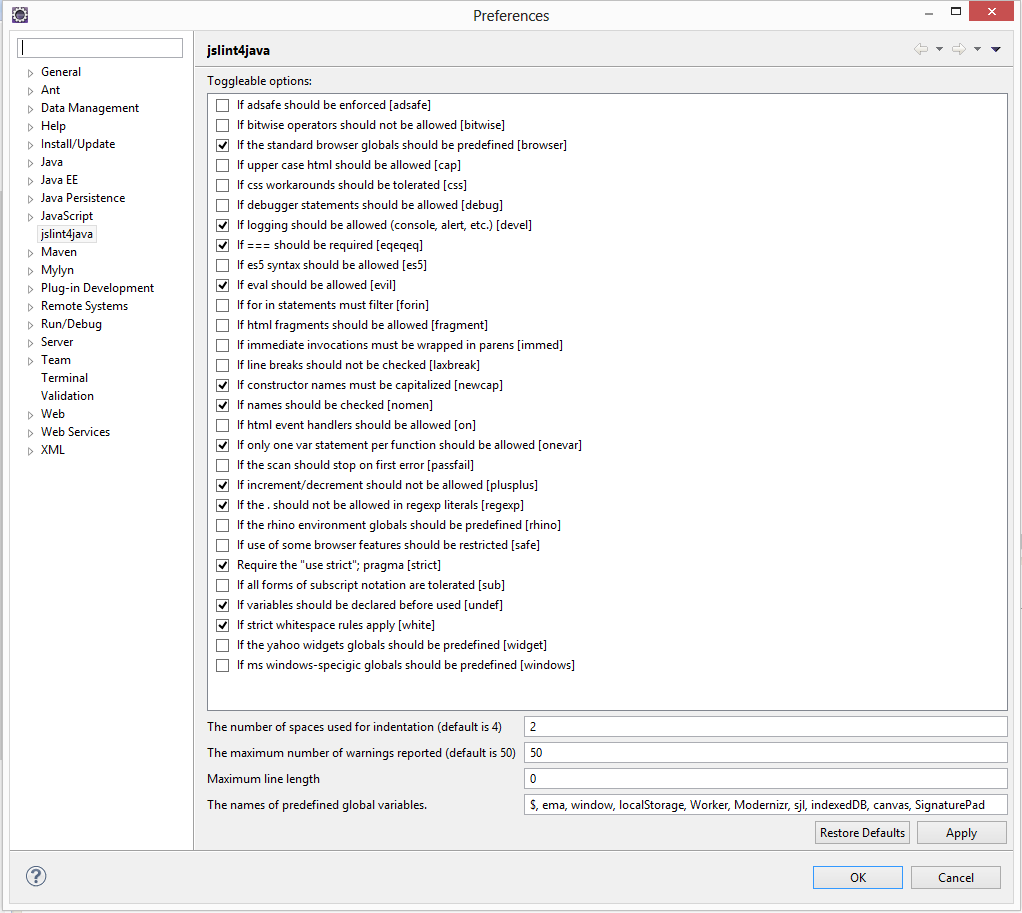
## JSLint

<https://code.google.com/p/jslint4java/>

Dieses Plugin muss für jedes Projekt einzeln aktiviert werden (Rechtsclick auf das Projekt):



Unter Window/Preferences müssen noch weitere Einstellungen getroffen werden:



# Anhang

## SourceCode von Inventurbeleg

/\*

\* ema.inv.lvs.inventurbeleg

\* inventurbeleg module for EMA

\*/

if (ema.inv === undefined) {

ema.inv = {};

}

if (ema.inv.lvs === undefined) {

ema.inv.lvs = {};

}

ema.inv.lvs.inventurbeleg = (function () {

'use strict';

/\*\* Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

var

configMap = {

main\_html : String()

},

stateMap = {

noPositions : 0,

openPositions : 0,

nummernListeTotal : [],

nummernListeSub : {},

nummerZeilenMatching : {}

},

signaturePad,

beleg,

//internal functions

generateHTML, generateBeleg, addEntryToList,

processAutocompleteResult, checkMatchingValues,

//public functions

initModule, onModuleLoad, checkCount,

saveCount, saveCountCallback, expandAll,

collapseAll, resetValue, showPosition;

/\*\* INTERNAL FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Begin INTERNAL method /generateHTML/

generateHTML = function (json) {

try {

var

i,

strHtml,

nachzaehlversion,

zaehlmenge,

zaehlmengeDisp,

mengeneinheit,

columnWidth,

columnLabel,

columnValue,

columnStyles,

hiddenColumnWidth,

hiddenColumnLabel,

hiddenColumnValue,

customHiddenContent,

dialogHtml;

nachzaehlversion = json[6] \* 1;

stateMap.openPositions = 0;

//Listen zurücksetzen

stateMap.nummerZeilenMatching = {};

stateMap.nummernListeTotal = [];

stateMap.nummernListeSub = {};

columnWidth = [5, 15, 20, 30, 15, 15];

columnLabel = ['lblposition', 'lbllagerplatz', 'lblmaterial', 'lblmaterialbezeichnung', 'lblmengeneinheit', 'lblzaehlmenge'];

columnStyles = ['', '', '', '', '', ''];

hiddenColumnWidth = [5, 15, 20, 30, 15, 15];

hiddenColumnLabel = ['lblbestandsqualifikation', 'lblquant', 'lbllagereinheit', 'lbllagereinheitentyp', '', 'lblcharge'];

strHtml = '<meta http-equiv="Content-Type" content="text/html; charset=utf-8">' +

//Start Formcontainer

'<div class="mhcontent">' +

ema.formgenerator.generateHeader('', 'lblinventurbelegzaehlenlvs', '', '', '', '') +

ema.formgenerator.generateRow('empty', 'iv\_lvs\_ib\_belegnummer', json[0], '', false, false, '', 'lblinventurbeleg', '<div class="cell r" ><button onclick="ema.shell.showHideSection(\'inventurbeleghead\')" class="tbg"><i class="icon-angle-down" id="buttoninventurbeleghead"></i></button></div>', '', true) +

'<div id="sectioninventurbeleghead\_hidden" class="w100p">' +

ema.formgenerator.generateRow('text', 'iv\_lvs\_ib\_zaehlenvon', '', '', false, false, '', 'lblzaehlenvon', '', '', true) +

ema.formgenerator.generateRow('empty', 'iv\_lvs\_ib\_lagerplatz', '<span id="iv\_lvs\_ib\_lagerplatz\_span"></span>', '', false, false, '', 'lbllagerplatz', '<button onclick="ema.inv.lvs.inventurbeleg.resetValue(\'lagerplatz\');"><i class="icon-minus"></i></button>', '', true) +

ema.formgenerator.generateRow('empty', 'iv\_lvs\_ib\_material', '<span id="iv\_lvs\_ib\_material\_span"></span>', '', false, false, '', 'lblmaterial', '<button onclick="ema.inv.lvs.inventurbeleg.resetValue(\'material\');"><i class="icon-minus"></i></button>', '', true) +

'</div>' +

//Hier kommen die Positionen

ema.formgenerator.generateHeader(ema.shell.getLanguageTextString('inventur', 'lblpositionen') + ' <span id="iv\_lvs\_ib\_offen\_span">&nbsp;</span>', '', '', '', '<button onclick="ema.inv.lvs.inventurbeleg.expandAll();"><i class="icon-plus"></i></button><button onclick="ema.inv.lvs.inventurbeleg.collapseAll();"><i class="icon-minus"></i></button>', 'r') +

'<div class="w100p" id="iv\_lvs\_ib\_positionen">';

stateMap.noPositions = -1;

if (json[9] !== undefined) {

for (i = 0; i < json[9].length; i += 1) {

//Merken, in welcher Zeile die Position angezeigt wird:

if (nachzaehlversion === json[9][i][28] \* 1) {

ema.inv.lvs.selectedBeleg[9][i][27] = i;

if (json[9][i][27] === 'X') {

zaehlmenge = '0';

mengeneinheit = json[9][i][13];

} else if (json[9][i][15] !== '' && json[9][i][15] !== json[9][i][13]) {

zaehlmenge = json[9][i][14];

mengeneinheit = json[9][i][15];

} else {

zaehlmenge = json[9][i][12];

mengeneinheit = json[9][i][13];

}

if (zaehlmenge \* 1 === 0) {

zaehlmenge = '';

zaehlmengeDisp = '';

} else {

zaehlmengeDisp = zaehlmenge.replace('.', ',');

}

if (zaehlmenge === '') {

columnValue = ['<span id="ema\_inv\_lvs\_pn\_' + i + '" class="b black">' + json[9][i][0] \* 1 + '</span>', json[9][i][6], json[9][i][8], json[9][i][26], '<span id="ema\_inv\_lvs\_me\_2\_' + i + '">' + mengeneinheit + '</span>', '<span id="ema\_inv\_lvs\_zm\_' + i + '">' + zaehlmengeDisp + '</span>'];

stateMap.openPositions += 1;

} else {

columnValue = ['<span id="ema\_inv\_lvs\_pn\_' + i + '">' + json[9][i][0] \* 1 + '</span>', json[9][i][6], json[9][i][8], json[9][i][26], '<span id="ema\_inv\_lvs\_me\_2\_' + i + '">' + mengeneinheit + '</span>', '<span id="ema\_inv\_lvs\_zm\_' + i + '">' + zaehlmengeDisp + '</span>'];

}

hiddenColumnValue = [json[9][i][1] \* 1, json[9][i][18], json[9][i][22], json[9][i][11], '&nbsp;', json[9][i][10]];

customHiddenContent = '<div class="w10p h92">&nbsp;</div>';

if (json[9][i][27] === 'X') {

customHiddenContent += '<div class="w20p dbr dbl"><div class="cellheader lblzaehlmenge" >&nbsp;</div><div class="cell"><input id="ema\_inv\_lvs\_zaehlmenge\_' + i + '" class="tbs lvs\_\_ib\_fc" type="number" value="" ></input></div></div>';

customHiddenContent += '<div class="w20p dbr"><div class="cellheader lblmengeneinheit" >&nbsp;</div><div class="cell"><span id="ema\_inv\_lvs\_me\_' + i + '">' + json[9][i][13] + '</span>&nbsp;</div></div>';

customHiddenContent += '<div class="w20p dbr"><div class="cellheader lblame" >&nbsp;</div><div class="cell"><input id="ema\_inv\_lvs\_ame\_' + i + '" class="tbs lvs\_\_ib\_fc" value="' + json[9][i][15] + '" ></input></div></div>';

customHiddenContent += '<div class="w20p dbr"><div class="cellheader lblnull" >&nbsp;</div><div class="cell"><input name="ema\_inv\_lvs\_null\_' + i + '" class="css-checkbox lvs\_\_ib\_fc" id="ema\_inv\_lvs\_null\_' + i + '" type="checkbox" checked><label class="css-label" for="ema\_inv\_lvs\_null\_' + i + '"></label></div></div>';

} else {

customHiddenContent += '<div class="w20p dbr dbl"><div class="cellheader lblzaehlmenge" >&nbsp;</div><div class="cell"><input id="ema\_inv\_lvs\_zaehlmenge\_' + i + '" class="tbs lvs\_\_ib\_fc" type="number" value="' + zaehlmenge + '" ></input></div></div>';

customHiddenContent += '<div class="w20p dbr"><div class="cellheader lblmengeneinheit" >&nbsp;</div><div class="cell"><span id="ema\_inv\_lvs\_me\_' + i + '">' + json[9][i][13] + '</span>&nbsp;</div></div>';

customHiddenContent += '<div class="w20p dbr"><div class="cellheader lblame" >&nbsp;</div><div class="cell"><input id="ema\_inv\_lvs\_ame\_' + i + '" class="tbs lvs\_\_ib\_fc" value="' + json[9][i][15] + '" ></input></div></div>';

customHiddenContent += '<div class="w20p dbr"><div class="cellheader lblnull" >&nbsp;</div><div class="cell"><input name="ema\_inv\_lvs\_null\_' + i + '" class="css-checkbox lvs\_\_ib\_fc" id="ema\_inv\_lvs\_null\_' + i + '" type="checkbox"><label class="css-label" for="ema\_inv\_lvs\_null\_' + i + '"></label></div></div>';

}

customHiddenContent += '<div class="w10p h92">&nbsp;</div>';

strHtml += ema.listgenerator.generateRow('ema\_inv\_lvs\_ib\_p\_' + i, columnWidth, columnLabel, columnValue, columnStyles, '', true, '',

hiddenColumnWidth, hiddenColumnLabel, hiddenColumnValue, customHiddenContent, undefined);

//Selection über "Zaehlen von"

addEntryToList(json[9][i][6], json[9][i][8], i);

} else {

ema.inv.lvs.selectedBeleg[9][i][27] = -1;

}

}

stateMap.noPositions = i;

}

strHtml += '</div>' +

'<div class="w100p h50">&nbsp;</div>' +

'</div>' +

//End Formcontainer

ema.formgenerator.generateFooter(true, false, '', true, 'lblzaehlungabschliessen', 'ema.inv.lvs.inventurbeleg.checkCount()');

dialogHtml = '<div id="ema\_inv\_lvs\_ib\_signature\_pad" class="m-signature-pad">' +

'<div class="w100p">' +

'<canvas></canvas>' +

'</div>' +

'<div class="w100p">' +

'<div class="signaturebutton"><button class="button\_nok w100p lblbuttondelete" onclick="ema.inv.lvs.inventurbeleg.resetSignature()"></button></div>' +

'<div class="signaturebutton"><button class="button\_ok w100p lblzaehlungsichern" onclick="ema.inv.lvs.inventurbeleg.saveCount()"></button></div>' +

'</div>' +

'</div>';

strHtml += ema.formgenerator.generateActionDialog('ema\_inv\_lvs\_ib\_signature', dialogHtml, 'Unterschrift');

configMap.main\_html = strHtml;

stateMap.nummernListeTotal.sort(function (a, b) {

return a > b;

});

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_generateHTML', e, 'e');

}

};

//End INTERNAL method /generateHTML/

//Begin INTERNAL method /generateBeleg/

generateBeleg = function () {

try {

var retObj, beleg, position, i, j;

//1. Ein BelegObject erstellen, dass von SAP interpretiert werden kann

retObj = {};

retObj.IMPORT = {};

retObj.IMPORT.I\_IVNUM = ema.inv.lvs.selectedBeleg[0];

retObj.IMPORT.I\_LGNUM = ema.shell.getConfigMapConfigValue('LGNUM');

retObj.IMPORT.I\_LGTYP = ema.inv.lvs.selectedBeleg[2];

retObj.IMPORT.IT\_LINK = [];

beleg = {};

beleg.IVNUM = ema.inv.lvs.selectedBeleg[0];

beleg.LINK\_ISTAT = ema.inv.lvs.selectedBeleg[1];

beleg.LINK\_LGTYP = ema.inv.lvs.selectedBeleg[2];

beleg.PDATU = ema.inv.lvs.selectedBeleg[3];

beleg.IVAKT = ema.inv.lvs.selectedBeleg[4];

beleg.UNAME = ema.model.loadFromLocalStorage('curr\_username');

beleg.NVERS = ema.inv.lvs.selectedBeleg[6];

beleg.IRNUM = ema.inv.lvs.selectedBeleg[7];

beleg.LGNUM = ema.inv.lvs.selectedBeleg[8];

beleg.DETAIL = [];

j = 0;

for (i = 0; i < ema.inv.lvs.selectedBeleg[9].length; i += 1) {

if (ema.inv.lvs.selectedBeleg[9][i][27] \* 1 > -1) {

if ($('#ema\_inv\_lvs\_zaehlmenge\_' + ema.inv.lvs.selectedBeleg[9][i][27]).val() !== '' || $('#ema\_inv\_lvs\_null\_' + ema.inv.lvs.selectedBeleg[9][i][27])[0].checked === true) {

position = {};

position.IVPOS = ema.inv.lvs.selectedBeleg[9][i][0];

position.LQNUM = ema.inv.lvs.selectedBeleg[9][i][1];

position.NANUM = ema.inv.lvs.selectedBeleg[9][i][2];

position.ISTAT = ema.inv.lvs.selectedBeleg[9][i][3];

position.IDATU = ema.inv.lvs.selectedBeleg[9][i][4];

position.LGTYP = ema.inv.lvs.selectedBeleg[9][i][5];

position.LGPLA = ema.inv.lvs.selectedBeleg[9][i][6];

position.PLPOS = ema.inv.lvs.selectedBeleg[9][i][7];

position.MATNR = ema.inv.lvs.selectedBeleg[9][i][8];

position.WERKS = ema.inv.lvs.selectedBeleg[9][i][9];

position.CHARG = ema.inv.lvs.selectedBeleg[9][i][10];

position.LETYP = ema.inv.lvs.selectedBeleg[9][i][11];

position.MENGE = $('#ema\_inv\_lvs\_zaehlmenge\_' + ema.inv.lvs.selectedBeleg[9][i][27]).val();

position.MEINS = ema.inv.lvs.selectedBeleg[9][i][13];

position.MENGA = ema.inv.lvs.selectedBeleg[9][i][14];

position.ALTME = $('#ema\_inv\_lvs\_ame\_' + ema.inv.lvs.selectedBeleg[9][i][27]).val();

position.UMREZ = ema.inv.lvs.selectedBeleg[9][i][16];

position.UMREN = ema.inv.lvs.selectedBeleg[9][i][17];

position.BESTQ = ema.inv.lvs.selectedBeleg[9][i][18];

position.SOBKZ = ema.inv.lvs.selectedBeleg[9][i][19];

position.SONUM = ema.inv.lvs.selectedBeleg[9][i][20];

position.GESME = ema.inv.lvs.selectedBeleg[9][i][21];

position.LENUM = ema.inv.lvs.selectedBeleg[9][i][22];

position.VFDAT = ema.inv.lvs.selectedBeleg[9][i][23];

position.LGORT = ema.inv.lvs.selectedBeleg[9][i][24];

position.UNAME = ema.model.loadFromLocalStorage('curr\_username');

position.MAKTX = ema.inv.lvs.selectedBeleg[9][i][26];

if ($('#ema\_inv\_lvs\_null\_' + ema.inv.lvs.selectedBeleg[9][i][27])[0].checked === true) {

position.KZNUL = 'X';

} else {

position.KZNUL = '';

}

beleg.DETAIL[j] = position;

j += 1;

}

}

}

retObj.IMPORT.IT\_LINK[0] = beleg;

return retObj;

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_generateBeleg', e, 'e');

}

};

//End INTERNAL method /generateBeleg/

//Begin INTERNAL method /addEntryToList/

addEntryToList = function (lagerplatz, material, rowId) {

try {

var

i,

foundLp,

foundMa,

newLagerplatz,

newMaterial;

newLagerplatz = 'Lagerplatz: ' + lagerplatz;

newMaterial = 'Material: ' + material;

//1. Liste von Lagerplatz & Material & Zeilennummer -> hier nur das erste Vorkommen eintragen

if (stateMap.nummerZeilenMatching[(lagerplatz + '\_\_\_' + material).toLowerCase()] === undefined) {

stateMap.nummerZeilenMatching[(lagerplatz + '\_\_\_' + material).toLowerCase()] = 'ema\_inv\_lvs\_ib\_p\_' + rowId;

}

//2. Liste aller Lagerplätze und Materialien -> für Autocomplete wenn noch nichts gewählt wurde

foundLp = false;

foundMa = false;

for (i = 0; i < stateMap.nummernListeTotal.length; i += 1) {

if (newLagerplatz === stateMap.nummernListeTotal[i]) {

foundLp = true;

}

if (newMaterial === stateMap.nummernListeTotal[i]) {

foundMa = true;

}

}

if (foundLp === false) {

stateMap.nummernListeTotal[stateMap.nummernListeTotal.length] = newLagerplatz;

}

if (foundMa === false) {

stateMap.nummernListeTotal[stateMap.nummernListeTotal.length] = newMaterial;

}

//3. Liste aller Lagerplätze und Materialien mit Unterliste der jeweils verknüpften Punkt -> für Autocomplete wenn ein Wert gewählt wurde

if (stateMap.nummernListeSub[newLagerplatz.toLowerCase()] === undefined) {

stateMap.nummernListeSub[newLagerplatz.toLowerCase()] = [];

stateMap.nummernListeSub[newLagerplatz.toLowerCase()][0] = newMaterial;

} else {

foundMa = false;

for (i = 0; i < stateMap.nummernListeSub[newLagerplatz.toLowerCase()].length; i += 1) {

if (newMaterial === stateMap.nummernListeSub[newLagerplatz.toLowerCase()][i]) {

foundMa = true;

}

}

if (foundMa === false) {

stateMap.nummernListeSub[newLagerplatz.toLowerCase()][stateMap.nummernListeSub[newLagerplatz.toLowerCase()].length] = newMaterial;

}

}

if (stateMap.nummernListeSub[newMaterial.toLowerCase()] === undefined) {

stateMap.nummernListeSub[newMaterial.toLowerCase()] = [];

stateMap.nummernListeSub[newMaterial.toLowerCase()][0] = newLagerplatz;

} else {

foundLp = false;

for (i = 0; i < stateMap.nummernListeSub[newMaterial.toLowerCase()].length; i += 1) {

if (newLagerplatz === stateMap.nummernListeSub[newMaterial.toLowerCase()][i]) {

foundLp = true;

}

}

if (foundLp === false) {

stateMap.nummernListeSub[newMaterial.toLowerCase()][stateMap.nummernListeSub[newMaterial.toLowerCase()].length] = newLagerplatz;

}

}

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_addEntryToList', e, 'e');

}

};

//End INTERNAL method /addEntryToList/

//Begin INTERNAL method /processAutocompleteResult/

processAutocompleteResult = function (selectedValue) {

try {

var

selectedValueSplit,

lagerplatz,

material,

tmpZeilenId,

zaehlmengeId,

fieldDeleted;

lagerplatz = '';

material = '';

fieldDeleted = false;

//1. Prüfen, ob und wenn was für ein Wert übergeben wurde:

if (selectedValue !== '') {

selectedValueSplit = selectedValue.toLowerCase().split(': ');

if (selectedValueSplit[0] === 'lagerplatz') {

$('#iv\_lvs\_ib\_lagerplatz\_span').html(selectedValueSplit.slice(1).join());

} else if (selectedValueSplit[0] === 'material') {

$('#iv\_lvs\_ib\_material\_span').html(selectedValueSplit.slice(1).join());

} else if (selectedValueSplit[0] === 'deletefield') {

fieldDeleted = true;

}

}

//2. Prüfen, ob schon beide Werte ausgewählt wurden

if ($('#iv\_lvs\_ib\_lagerplatz\_span').html() !== '') {

lagerplatz = $('#iv\_lvs\_ib\_lagerplatz\_span').html();

}

if ($('#iv\_lvs\_ib\_material\_span').html() !== '') {

material = $('#iv\_lvs\_ib\_material\_span').html();

}

//3. sind beide Werte leer wird die gesamte Liste ans Autocomplete gehängt

if (material === '' && lagerplatz === '') {

collapseAll();

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('enable');

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('option', 'source', stateMap.nummernListeTotal);

} else if (material === '' || lagerplatz === '') {

//4. wenn nur ein Wert befüllt ist prüfen, ob es eine Liste passender ... gibt

if (material !== '') {

//prüfen, ob es nur einen passenden Wert gibt

if (stateMap.nummernListeSub[('Material: ' + material).toLowerCase()].length === 1) {

if (fieldDeleted === false) {

//Lagerplatz befüllen

selectedValueSplit = stateMap.nummernListeSub[('Material: ' + material).toLowerCase()][0].split(': ');

lagerplatz = selectedValueSplit.slice(1).join();

$('#iv\_lvs\_ib\_lagerplatz\_span').html(lagerplatz);

collapseAll();

ema.shell.showHideSection(stateMap.nummerZeilenMatching[(lagerplatz + '\_\_\_' + material).toLowerCase()]);

tmpZeilenId = stateMap.nummerZeilenMatching[(lagerplatz + '\_\_\_' + material).toLowerCase()].split('\_');

zaehlmengeId = '#ema\_inv\_lvs\_zaehlmenge\_' + tmpZeilenId[tmpZeilenId.length - 1];

//$(zaehlmengeId)[0].focus();

$(zaehlmengeId).select();

} else {

//2. Feld löschen und Komplettliste anzeigen

collapseAll();

$('#iv\_lvs\_ib\_material\_span').html('');

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('enable');

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('option', 'source', stateMap.nummernListeTotal);

}

} else {

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('enable');

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('option', 'source', stateMap.nummernListeSub[('Material: ' + material).toLowerCase()]);

}

}

if (lagerplatz !== '') {

//prüfen, ob es nur einen passenden Wert gibt

if (stateMap.nummernListeSub[('Lagerplatz: ' + lagerplatz).toLowerCase()].length === 1) {

if (fieldDeleted === false) {

//Material befüllen

selectedValueSplit = stateMap.nummernListeSub[('Lagerplatz: ' + lagerplatz).toLowerCase()][0].split(': ');

material = selectedValueSplit.slice(1).join();

$('#iv\_lvs\_ib\_material\_span').html(material);

collapseAll();

ema.shell.showHideSection(stateMap.nummerZeilenMatching[(lagerplatz + '\_\_\_' + material).toLowerCase()]);

tmpZeilenId = stateMap.nummerZeilenMatching[(lagerplatz + '\_\_\_' + material).toLowerCase()].split('\_');

zaehlmengeId = '#ema\_inv\_lvs\_zaehlmenge\_' + tmpZeilenId[tmpZeilenId.length - 1];

//$(zaehlmengeId)[0].focus();

$(zaehlmengeId).select();

} else {

//2. Feld löschen und Komplettliste anzeigen

collapseAll();

$('#iv\_lvs\_ib\_lagerplatz\_span').html('');

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('enable');

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('option', 'source', stateMap.nummernListeTotal);

}

} else {

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('enable');

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('option', 'source', stateMap.nummernListeSub[('Lagerplatz: ' + lagerplatz).toLowerCase()]);

}

}

} else {

//5. beide Werte sind befüllt - versuchen die entsprechende Zeile aufzuklappen

collapseAll();

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete('disable');

ema.shell.showHideSection(stateMap.nummerZeilenMatching[(lagerplatz + '\_\_\_' + material).toLowerCase()]);

tmpZeilenId = stateMap.nummerZeilenMatching[(lagerplatz + '\_\_\_' + material).toLowerCase()].split('\_');

zaehlmengeId = '#ema\_inv\_lvs\_zaehlmenge\_' + tmpZeilenId[tmpZeilenId.length - 1];

//$(zaehlmengeId)[0].focus();

$(zaehlmengeId).select();

}

$('#iv\_lvs\_ib\_zaehlenvon').val('');

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_processAutocompleteResult', e, 'e');

}

};

//End INTERNAL method /processAutocompleteResult/

//Begin INTERNAL method /checkMatchingValues/

checkMatchingValues = function (enteredValue) {

try {

if (stateMap.nummernListeSub[('Lagerplatz: ' + enteredValue).toLowerCase()] !== undefined) {

processAutocompleteResult(('Lagerplatz: ' + enteredValue).toLowerCase());

} else if (stateMap.nummernListeSub[('Material: ' + enteredValue).toLowerCase()] !== undefined) {

processAutocompleteResult(('Material: ' + enteredValue).toLowerCase());

} else if (stateMap.nummernListeSub[enteredValue] !== undefined) {

processAutocompleteResult(enteredValue);

}

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_checkMatchingValues', e, 'e');

}

};

//End INTERNAL method /checkMatchingValues/

/\*\* PUBLIC FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Begin Public method /initModule/

initModule = function (path, container) {

try {

ema.shell.addOnloadFunction(path, ema.inv.lvs.inventurbeleg.onModuleLoad);

ema.shell.addOnloadFocusField(path, 'iv\_lvs\_ib\_zaehlenvon');

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_initModule', e, 'e');

}

};

// End PUBLIC method /initModule/

// Begin Public method /onModuleLoad/

onModuleLoad = function (container) {

try {

var

wrapper,

canvas;

generateHTML(ema.inv.lvs.selectedBeleg);

$('#' + container).html(configMap.main\_html);

$('#iv\_lvs\_ib\_offen\_span').html(' (<button onclick="ema.inv.lvs.inventurbeleg.showPosition();" class="header">' + stateMap.openPositions + ' ' + ema.shell.getLanguageTextString('inventur', 'lblpositionenoffen') + '</button>)');

$('#iv\_lvs\_ib\_zaehlenvon').autocomplete({

source : stateMap.nummernListeTotal,

messages: {

noResults: '',

results: function () {}

},

select: function (event, ui) {

processAutocompleteResult(ui.item.value);

$(this).val('');

return false;

}

});

$('#iv\_lvs\_ib\_zaehlenvon').change(function (event) {

try {

checkMatchingValues(this.value);

} catch (e) {

ema.shell.handleError('onValueChange', e, 'e');

}

});

$('#iv\_lvs\_ib\_zaehlenvon').keyup(function (event) {

try {

if (event.keyCode === 13) {

checkMatchingValues(this.value);

}

} catch (e) {

ema.shell.handleError('onKeyUp', e, 'e');

}

});

$('.lvs\_\_ib\_fc').change(function (event) {

try {

var

rowNo,

fieldType,

tmpFieldName,

i;

tmpFieldName = this.id.split('\_');

rowNo = tmpFieldName[tmpFieldName.length - 1];

fieldType = tmpFieldName[tmpFieldName.length - 2];

//die letzen beiden Einträge im Array entfernen

tmpFieldName.pop();

tmpFieldName.pop();

if (fieldType === 'zaehlmenge') {

//null zurücksetzen, zaehlmenge in die 1. Zeile übernehmen

if (this.value > 0) {

$('#ema\_inv\_lvs\_null\_' + rowNo).attr('checked', false);

$('#ema\_inv\_lvs\_zm\_' + rowNo).html(this.value);

}

} else if (fieldType === 'ame') {

//ame in die erste Zeile übernehmen, wenn '' wird me übernommen

if (this.value === '') {

$('#ema\_inv\_lvs\_me\_2\_' + rowNo).html($('#ema\_inv\_lvs\_me\_' + rowNo).html());

} else {

$('#ema\_inv\_lvs\_me\_2\_' + rowNo).html(this.value);

}

} else if (fieldType === 'null') {

//ame und zaehlmenge leeren

if (this.checked === true) {

$('#ema\_inv\_lvs\_zaehlmenge\_' + rowNo).val('0');

$('#ema\_inv\_lvs\_zm\_' + rowNo).html('0');

}

}

//Prüfen, ob die Position gezählt ist

if ($('#ema\_inv\_lvs\_zaehlmenge\_' + rowNo).val() > 0 || document.getElementById('ema\_inv\_lvs\_null\_' + rowNo).checked === true) {

if ($('#ema\_inv\_lvs\_pn\_' + rowNo).hasClass('b') === true) {

$('#ema\_inv\_lvs\_pn\_' + rowNo).toggleClass('b');

}

if ($('#ema\_inv\_lvs\_pn\_' + rowNo).hasClass('black') === true) {

$('#ema\_inv\_lvs\_pn\_' + rowNo).toggleClass('black');

}

} else {

if ($('#ema\_inv\_lvs\_pn\_' + rowNo).hasClass('b') === false) {

$('#ema\_inv\_lvs\_pn\_' + rowNo).toggleClass('b');

}

if ($('#ema\_inv\_lvs\_pn\_' + rowNo).hasClass('black') === false) {

$('#ema\_inv\_lvs\_pn\_' + rowNo).toggleClass('black');

}

}

stateMap.openPositions = 0;

for (i = 0; i < stateMap.noPositions; i += 1) {

if ($('#ema\_inv\_lvs\_pn\_' + i).hasClass('b') === true) {

stateMap.openPositions += 1;

}

}

$('#iv\_lvs\_ib\_offen\_span').html(' (<button onclick="ema.inv.lvs.inventurbeleg.showPosition();" class="header">' + stateMap.openPositions + ' ' + ema.shell.getLanguageTextString('inventur', 'lblpositionenoffen') + '</button>)');

} catch (e) {

ema.shell.handleError('onValueChange', e, 'e');

}

});

wrapper = document.getElementById('ema\_inv\_lvs\_ib\_signature\_pad');

canvas = wrapper.querySelector('canvas');

signaturePad = new SignaturePad(canvas, {

minWidth: ema.shell.getConfigMapConfigValue('SIGNATUREMINWIDTH'),

maxWidth: ema.shell.getConfigMapConfigValue('SIGNATUREMAXWIDTH'),

penColor: ema.shell.getConfigMapConfigValue('SIGNATUREPENCOLOR')

});

ema.shell.resizeCanvas('ema\_inv\_lvs\_ib\_signature\_pad');

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_onModuleLoad', e, 'e');

}

};

// End PUBLIC method /onModuleLoad/

// Begin Public method /checkCount/

checkCount = function () {

try {

beleg = generateBeleg();

if (beleg.IMPORT.IT\_LINK[0].DETAIL.length > 0) {

ema.shell.showDialog('ema\_inv\_lvs\_ib\_signature');

ema.shell.resizeCanvas('ema\_inv\_lvs\_ib\_signature\_pad');

} else {

ema.shell.handleCustomError('iv\_lvs\_ib\_saveCount', 'errsavecountnodata', 'e');

}

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_checkCount', e, 'e');

}

};

// End PUBLIC method /saveCount/

// Begin Public method /saveCount/

saveCount = function () {

try {

ema.datamanager.doRequest('LVS\_SAVEINVENTURBELEG', beleg, ema.inv.lvs.inventurbeleg.saveCountCallback, beleg.IMPORT.IT\_LINK[0].IVNUM);

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_saveCount', e, 'e');

}

};

// End PUBLIC method /saveCount/

// Begin Public method /saveCountCallback/

saveCountCallback = function (json) {

try {

ema.shell.showDialog('ema\_inv\_lvs\_ib\_signature');

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_saveCountCallback', e, 'e');

}

};

// End PUBLIC method /saveCountCallback/

// Begin Public method /expandAll/

expandAll = function () {

try {

var i = 0;

if (stateMap.noPositions > -1) {

for (i = 0; i <= stateMap.noPositions; i += 1) {

if ($('#buttonema\_inv\_lvs\_ib\_p\_' + i).hasClass('icon-angle-up') === true) {

$('#buttonema\_inv\_lvs\_ib\_p\_' + i).toggleClass('icon-angle-down icon-angle-up');

}

if ($('#sectionema\_inv\_lvs\_ib\_p\_' + i).hasClass('greybg') === false) {

$('#sectionema\_inv\_lvs\_ib\_p\_' + i).toggleClass('greybg');

}

$('#sectionema\_inv\_lvs\_ib\_p\_' + i + '\_hidden').show();

}

}

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_expandAll', e, 'e');

}

};

// End PUBLIC method /expandAll/

// Begin Public method /collapseAll/

collapseAll = function () {

try {

var i = 0;

if (stateMap.noPositions > -1) {

for (i = 0; i <= stateMap.noPositions; i += 1) {

if ($('#buttonema\_inv\_lvs\_ib\_p\_' + i).hasClass('icon-angle-down') === true) {

$('#buttonema\_inv\_lvs\_ib\_p\_' + i).toggleClass('icon-angle-down icon-angle-up');

}

if ($('#sectionema\_inv\_lvs\_ib\_p\_' + i).hasClass('greybg') === true) {

$('#sectionema\_inv\_lvs\_ib\_p\_' + i).toggleClass('greybg');

}

$('#sectionema\_inv\_lvs\_ib\_p\_' + i + '\_hidden').hide();

}

}

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_collapseAll', e, 'e');

}

};

// End PUBLIC method /collapseAll/

// Begin Public method /resetValue/

resetValue = function (valueName) {

try {

$('#iv\_lvs\_ib\_' + valueName + '\_span').html('');

processAutocompleteResult('deletefield');

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_resetValue', e, 'e');

}

};

// End PUBLIC method /resetValue/

// Begin Public method /showPosition/

showPosition = function () {

try {

var

i,

sectionExpanded;

sectionExpanded = false;

for (i = 0; i < stateMap.noPositions; i += 1) {

if ($('#ema\_inv\_lvs\_pn\_' + i).hasClass('b') === true) {

if (sectionExpanded === false) {

sectionExpanded = true;

ema.shell.showHideSection('ema\_inv\_lvs\_ib\_p\_' + i);

//$('#ema\_inv\_lvs\_zaehlmenge\_' + i)[0].focus();

$('#ema\_inv\_lvs\_zaehlmenge\_' + i).select();

}

}

}

} catch (e) {

ema.shell.handleError('iv\_lvs\_ib\_showPosition', e, 'e');

}

};

// End PUBLIC method /showPosition/

return {

initModule : initModule,

onModuleLoad : onModuleLoad,

checkCount : checkCount,

saveCount : saveCount,

saveCountCallback : saveCountCallback,

expandAll : expandAll,

collapseAll : collapseAll,

resetValue : resetValue,

showPosition : showPosition

};

}());