Dokumentation zum Dürr HMI Statusfenster Editor

Inhaltsverzeichnis

1 Der Statusfenster Editor - Einleitung

Durch einen kleinen Umbau in der emos.js ist es nun möglich, für jede FU eine externe JSON-Datei zu laden, um das Statusfenster aufzubauen. Diese JSON-Datei wird nur noch bei Bedarf geladen, also wenn ein User auf die entsprechende FU im Anlagenbild klickt. Dies hat den Vorteil, dass der Speicher nicht sämtliche FU's aufnehmen muss, sondern immer nur den Bauplan für eine Einheit.

Um den bestehenden StatusWindow-Designer abzulösen, wurde ein neuer Editor in das neue Framework integriert. Diese moderne, webbasierte Software soll dem Nutzer eine völlig neue und verbesserte Userexperience bieten.

2 Der Editor

Den neuen StatusWindow-Designer erreicht der angemeldete Benutzer mit entsprechenden Administratorrechten über die Framework-Konfiguration. Dort findet er dann den Button „Editor“.

2.1 Die Übersichtsliste

Klickt der User dann auf den Eintrag Editor, so wird ihm der Inhalt des Ordners 1 aus dem StatWin-Verzeichnis präsentiert. Dieser Inhalt sind sämtliche bislang angelegte JSON-Dateien für die Library 1. Oberhalb dieser umfassenden Liste hat der Benutzer die Möglichkeit, in eine andere Library zu schalten. Eine Umschaltung bewirkt, dass eine Liste der vorhandenen JSON-Dateien aus der selektierten Library erstellt wird. Um die Lister zu erstellen, wird die Funktion

#3.2.function buildJsonList($(".configcenter"));

aufgerufen. In dieser Funktion wird aber nicht nur die Liste erstellt, es wird auch ein DropDown generiert, mit dessen Hilfe man sich schnell in einer größeren Liste bewegen kann. Auch werden die Listeneinträge nach Versionen gebündelt. Sprich, wenn es für eine FU mehrere Versionen gibt, wird der Listeneintrag mit „Versions“ gekennzeichnet. Klickt der Benutzer nun auf einen der 4 angezeigten Buttons in einem Listenelement, wird wiederum ein DropDown mit den einzelnen Versionen gezeigt und zur Auswahl angeboten. Die genannten 4 Buttons bieten folgende Dienste an:

Download: der Benutzer kann die gewünschte JSON-Datei z.B. auf einen USB-Stick downloaden. Dazu wird die Funktion

#3.47.function downloadJson(filename);

aufgerufen.

Delete: der Benutzer kann die gewünschte JSON-Datei vom Server löschen. Dafür ist die Funktion

getDeleteDialog(dataFile, true);

zuständig(siehe Framework-Doku 6.44 getDeleteDialog(filename, path)).

Copy: der User kann hiermit eine JSON-Datei klonen. Es wird eine komplette Kopie der Datei zur Verfügung gestellt, mit Ausnahme des Feldes „DiagnosisControl“. Dieses Feld muss mit einem anderen Text versehen werden, da dies der Name der neuen JSON-Datei wird.

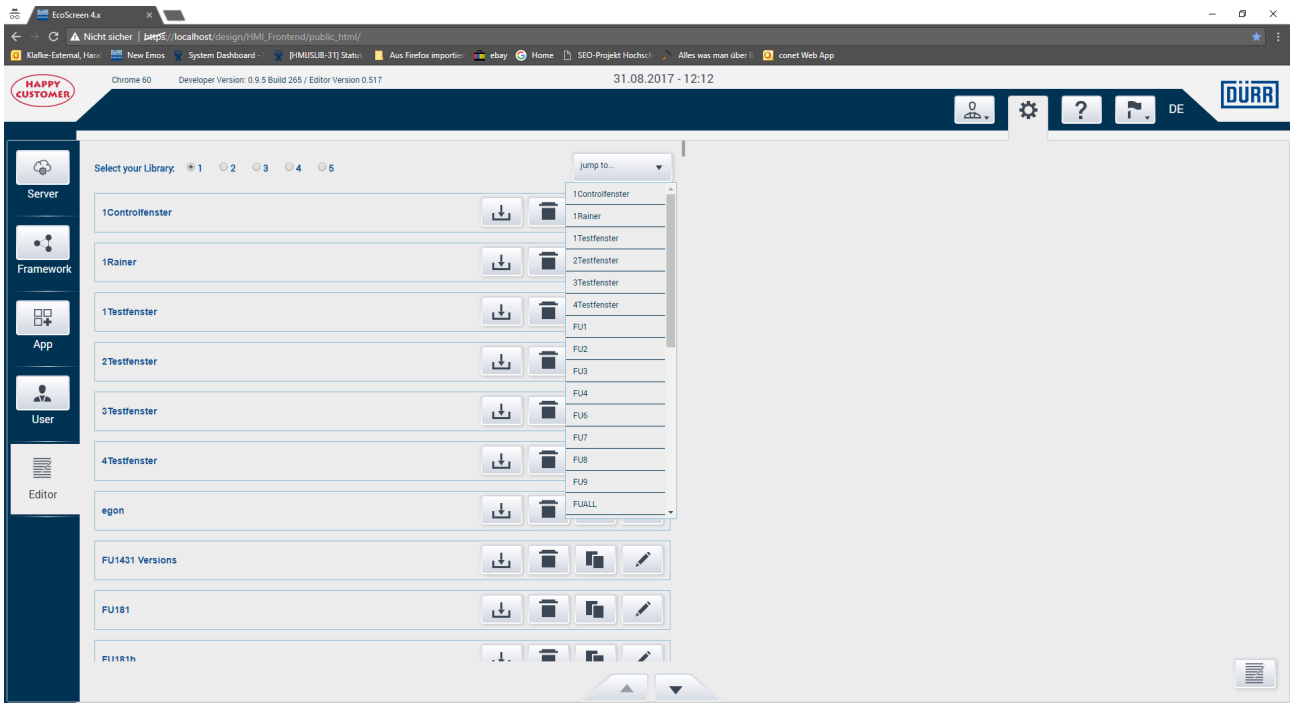
Edit: der Benutzer kann die gewünschte JSON-Datei bearbeiten. Copy und Edit nutzen beide die Funktion

3.5.function editJson(parent.parent(), false, true); (copy)

bzw.

3.5.function editJson(parent, false); (edit)

Zusätzlich befindet sich im unteren rechten Bereich noch ein Button, um eine neue JSON-Datei zu erzeugen.



2.2 Eine neue JSON-Datei erstellen

Klickt der User unten rechts auf den „new JSON“-Button wird die Funktion

3.4.function newJson(editjson, reload, duplicateFile)

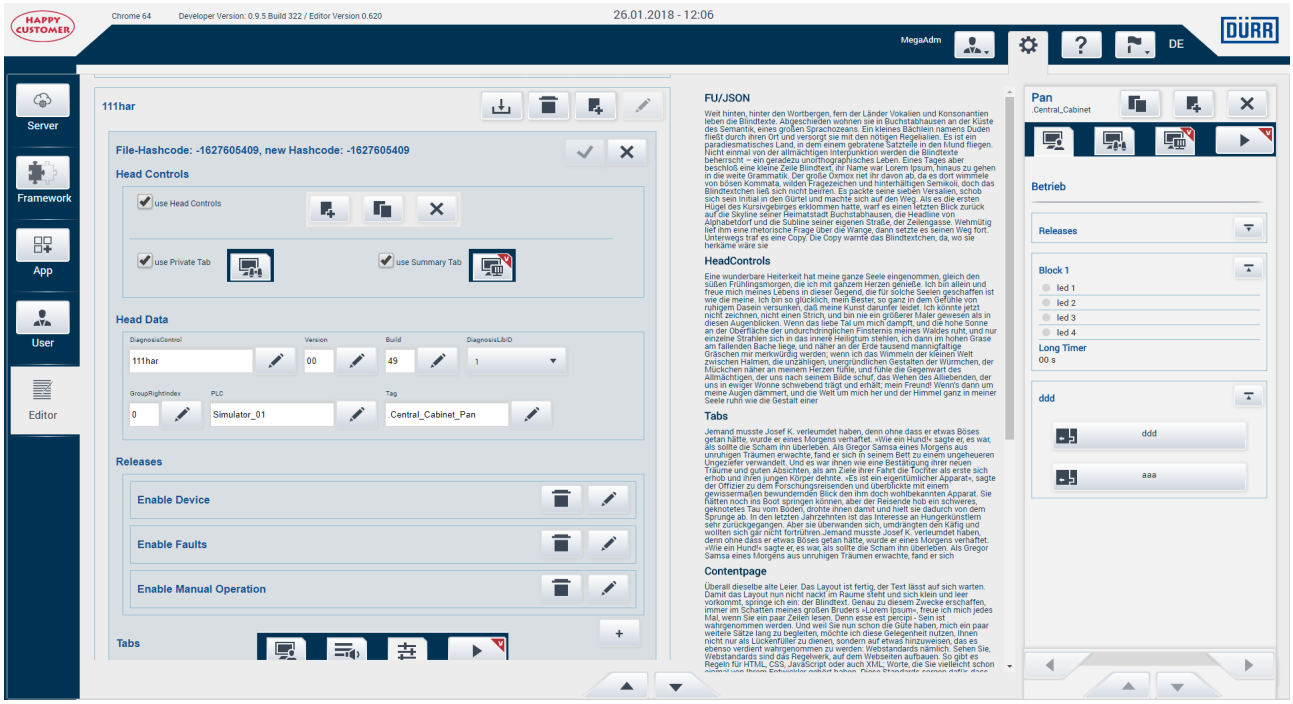
ausgeführt. Diese Funktion stellt die Eingabefelder der Kopfdaten zur Verfügung. Damit der freundliche Dateneingeber auch immer genau weiß, was er in die entsprechenden Felder eintragen muss, findet sich rechts neben dem Eingabebereich ein Hilfetext, mit ausführlichen Erklärungen.

2.2.1 Kopfdaten

Die Kopfdaten sind in 2 Bereiche aufgeteilt: die Head Controls und die Head Data. Die Head Controls sind wählbare Zusatzfunktionen. Der User kann entscheiden, ob er die Funktionsleiste im Head des Statusfensters haben möchte. Diese Leiste bietet die 3 Funktionen „gleiche FU in weiterem Fenster öffnen“, „eine weitere FU in einem weiteren Fenster öffnen“ oder „Statusfenster schließen“.

Auch besteht die Möglichkeit, die Private-Tabs und/oder den Summary-Tab abzuwählen. Hierbei ist aber zu beachten, dass eine Preview des Statusfenster erst erfolgen kann, wenn min. 1 Tab angelegt wurde. Aus diesem Grund sind die Tabs „Private“ und „Summary“ auch per default angewählt. So kann direkt nach der Eingabe der Head-Data eine Preview generiert werden. Selbstverständlich sind sowohl die Funktionsleiste als auch die beiden default-Tabs auch zu einem späteren Zeitpunkt an- bzw. abwählbar.

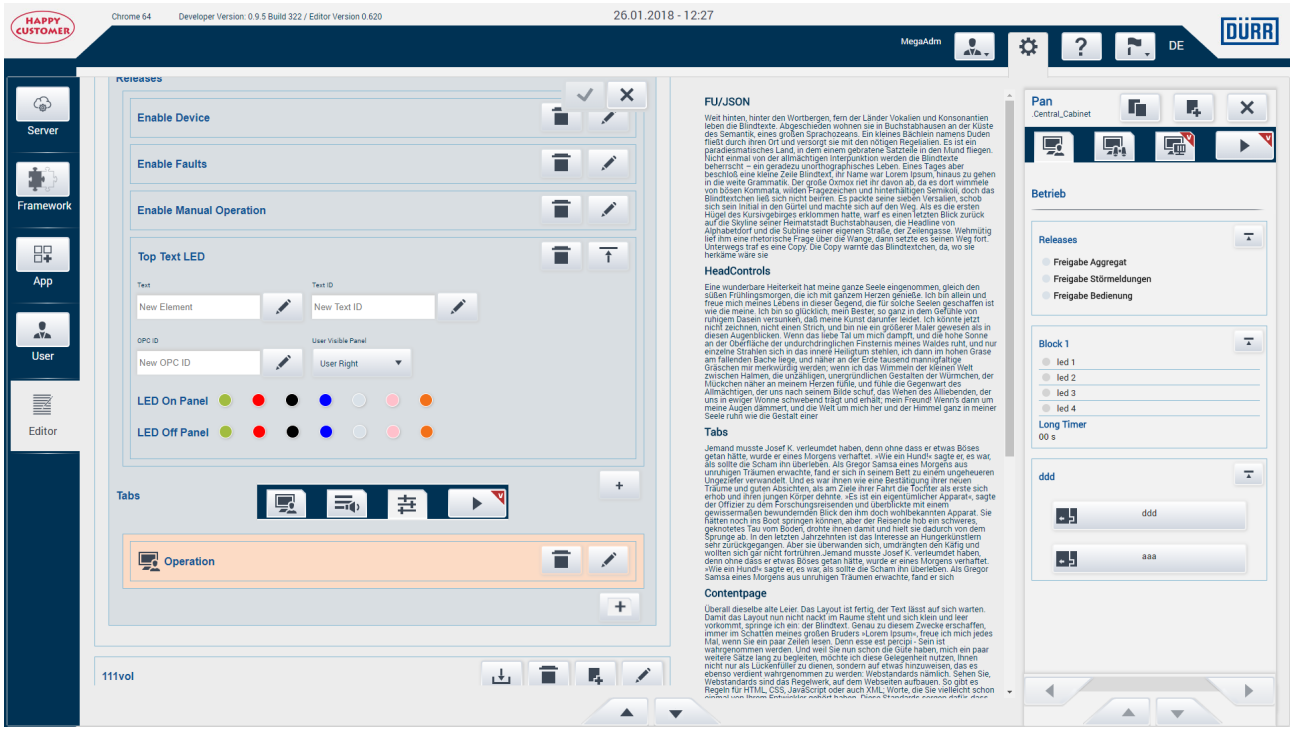
Die unter „Head Data“ einzugebenden Daten sind werden benötigt, um eine realistische Preview zu erstellen.



2.2.2 Releases

Unterhalb der Head-Data folgen nun die Releases. Diese werden im Statusfenster → Tab Betrieb und im Tab Summaries(Dürr-Tab) angezeigt. Möchte man nun ein Release hinzufügen, klickt man azf den Button „add a new Release“, welcher sich unterhalb der Release-Liste befindet. Dadurch wird die Eingabemaske für die Releases geladen und steht zum ausfüllen bereit.

3.6.function newHeadLED(true)



2.2.3 Tabs

Wie oben bereits erwähnt, muss es mindestens einen Tab auf dem Statusfernster geben. Um einen solchen Tab hinzuzufügen, klickt der geneigte User einfach auf den Button „new Tab“. Daraufhin wird dann mittels der Funktion

3.6.function newTab(addNew, selector, afterDelete)

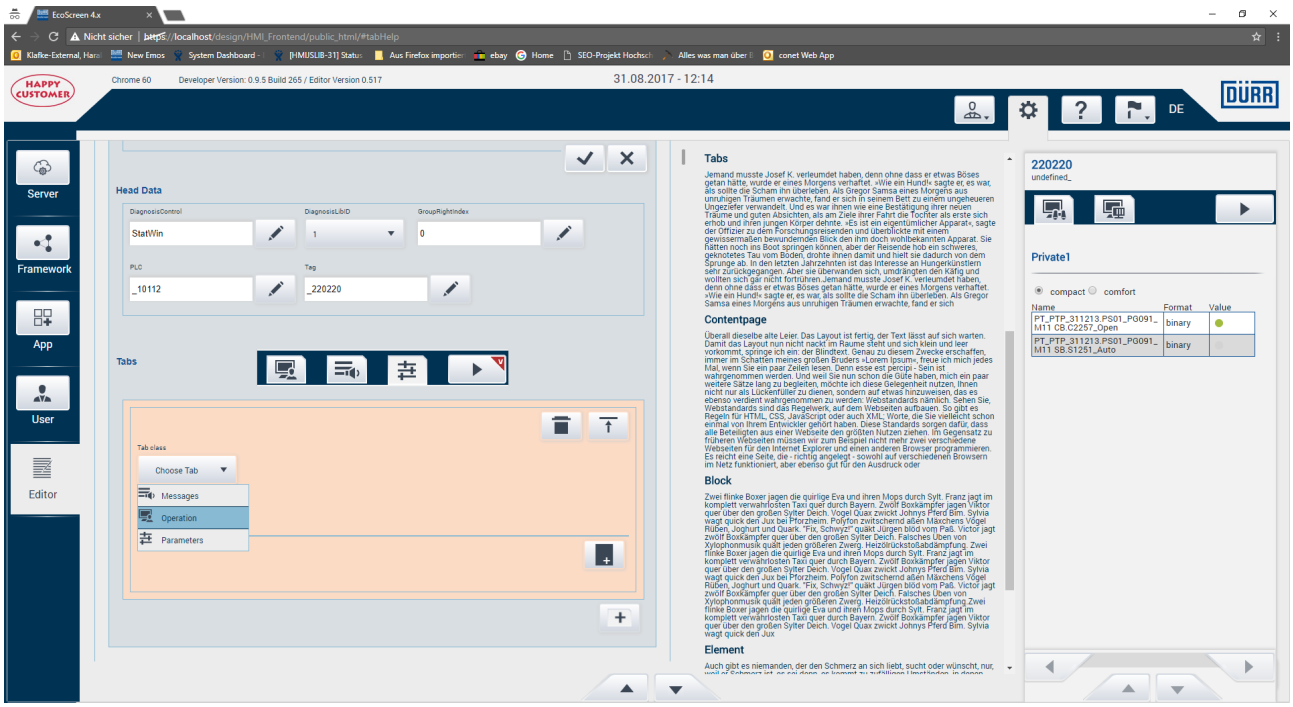
die Tabsektion geladen, die ein DropDown zur Verfügung stellt, das die möglichen Tabs beinhaltet. Das DropDown wird mittels

buildClassList('svg/tabicons', id, selector);

erstellt(siehe Framework-Doku 6.25 buildClassList(path, id, selector)). Dazu wird der Ordner mit den TabIcons ausgelesen. Sollte der Benutzer einen Tab anlegen wollen, der noch nicht existiert, muss zuerst ein entsprechendes Icon in den „tabicons“-Ordner gelegt und eine kleine Anpassung in der CSS vorgenommen werden(siehe _header.scss z.B. Zeile 992).

Macht der User nun eine Auswahl, z.B. „Operation“, wird automatisch auch die erste Contentpage erstellt. Die Contentpages bieten die Möglichkeit, die Tabs übersichtlich und besser nutzbar zu halten. Statt endlosem scrollen kann sich der User dann durchblättern.

Diese erste Contentpage erhält automatisch den Namen des Tabs, was aber natürlich änderbar ist.

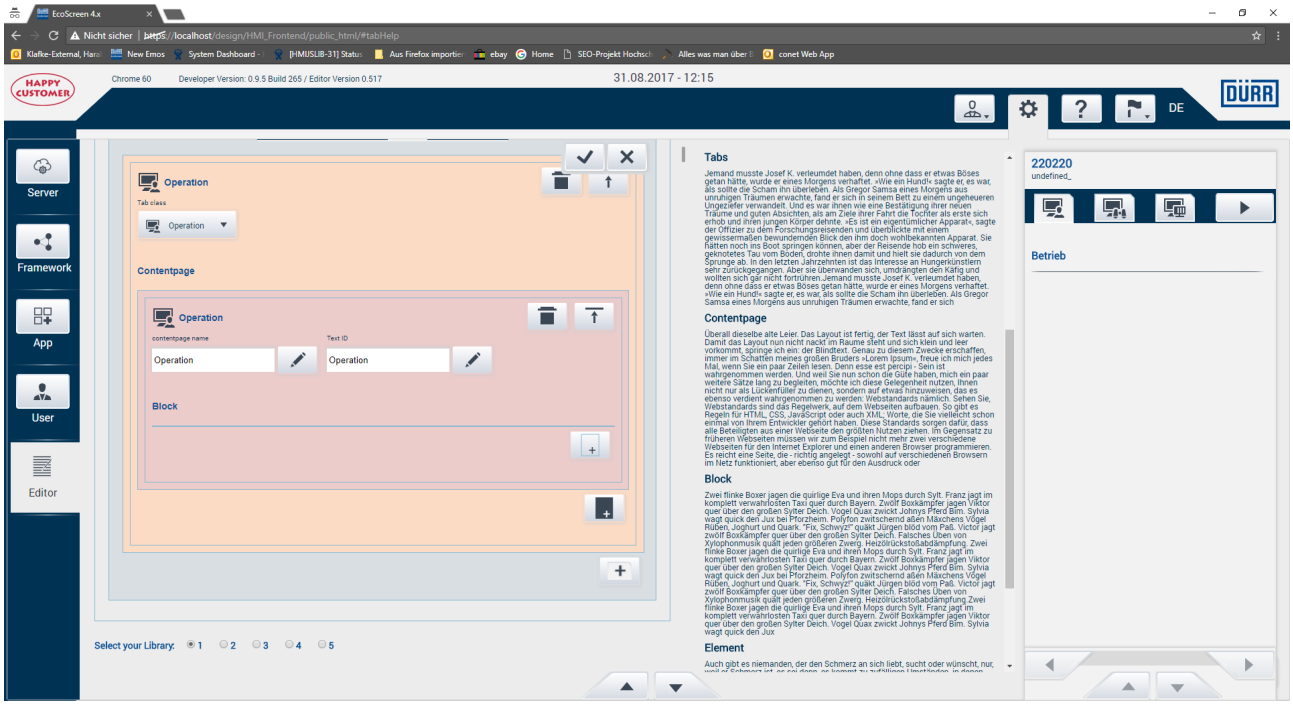


2.2.4 Contentpage

Die Menge der Contentpages(Seiten) innerhalb eines Tabs sind derzeit nicht begrenzt. Werden also noch weitere Contentpages benötigt, kann der User diese leicht durch einen Klick auf den Button „add Contentpage“ hinzufügen. Durch die Funktion

3.8.function newContentPage(addNew, datanum, myText, afterDelete)

wird die Eingabemaske geladen.

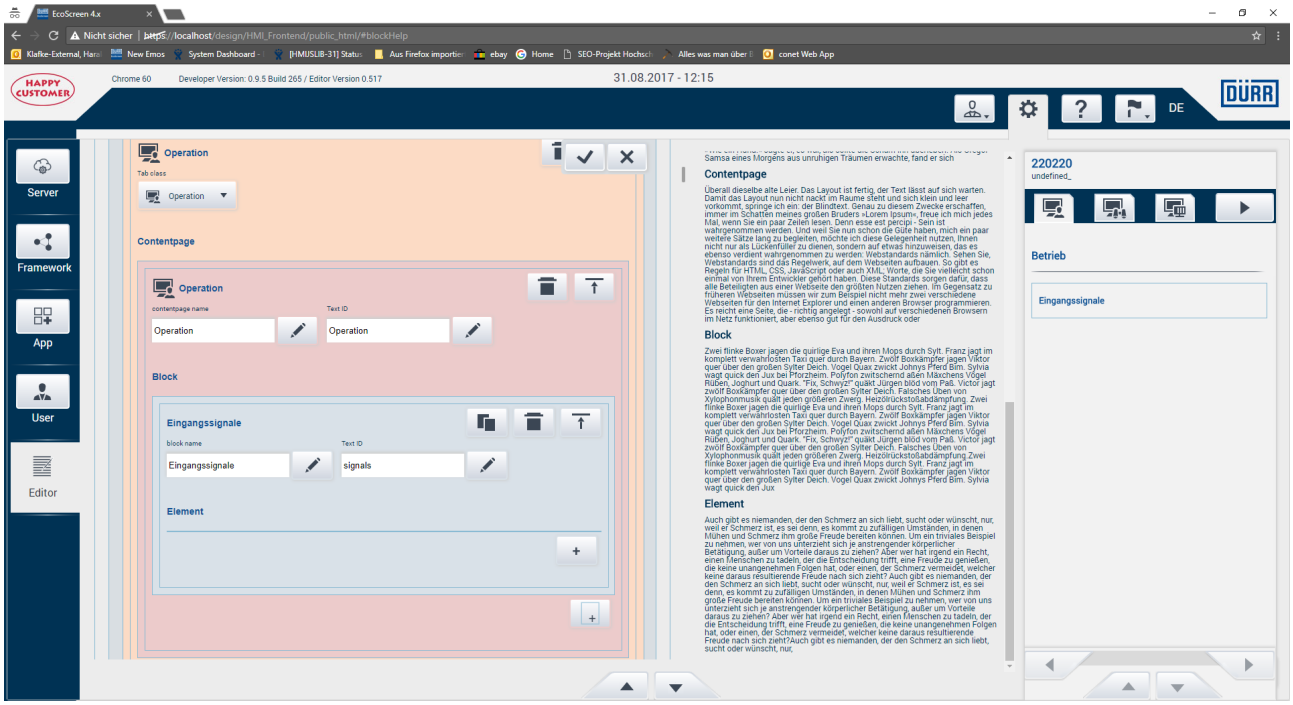


2.2.5 Blöcke

Das neue Design des Frameworks erleichtert die sichtbare Abgrenzung thematisch gegliederter Gruppen, sog. Blöcke. Jede Contentpage muss mindestens einen Block enthalten. Mit Klick auf „add Block“ sorgt die Funktion

3.10.function newBlock(addNew, selector, data)

dass die Block-Maske geladen wird. Der hier eingetragene Text wird dann automatisch die Überschrift des Blockes. Soll der Block aber keine Überschrift bekommen, so ist nur ein Leerzeichen in das Feld einzugeben.



2.2.6 Elemente

Da der Benutzer einen Block kreiert hat, stehen nun auch die Elemente zur Verfügung. Ein Klick auf den Button „add Element“ öffnet eine Auswahlliste mit allen zur Verfügung stehenden Elementen. Diese Auswahlliste wird aus der element.json generiert. In der element.json finden sich die Elementnamen, wie sie im alten StatusWindow-Designer vorkamen und der dazugehörende Bildname. Klickt der User nun in der Auswahlliste auf das gewünschte Element, wird mittels

3.45.function getElementMask(val, parents, parent, newChildID)

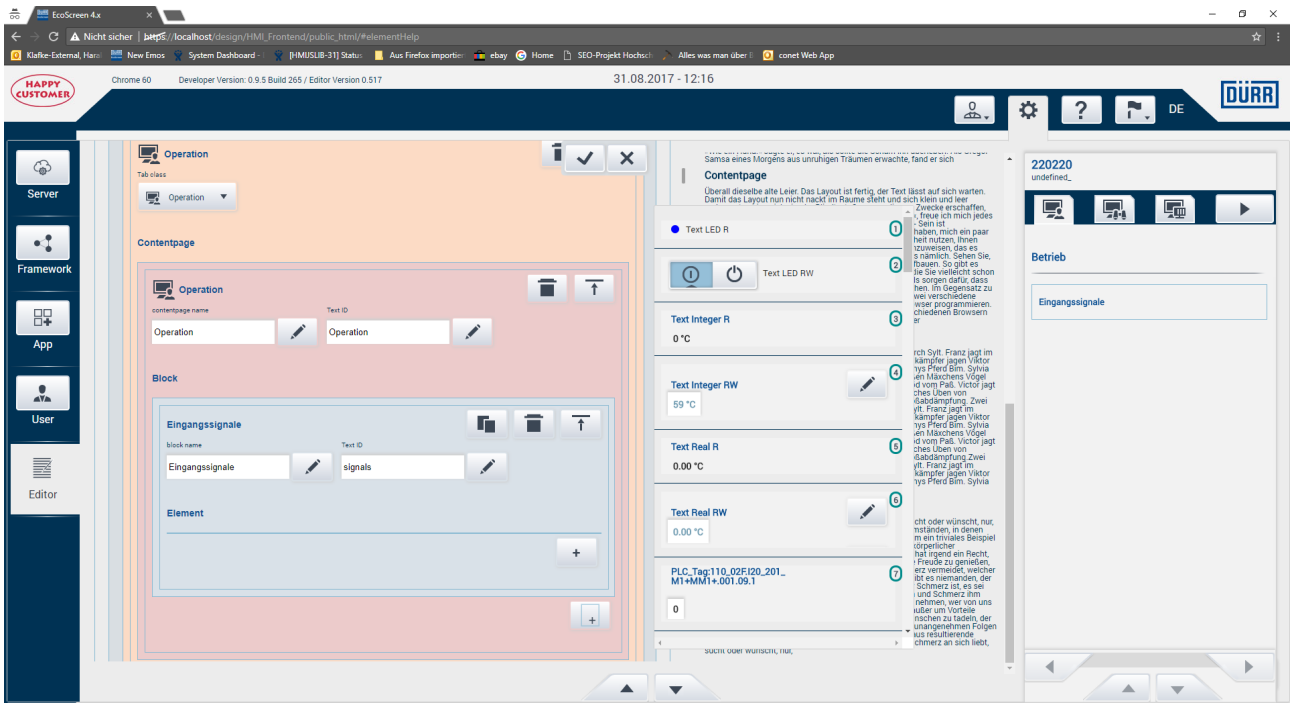
die entsprechende Datenstruktur vorbereitet und dann mittels

3.12.function newElement(whatElement, parents, val, structure, newChildID)

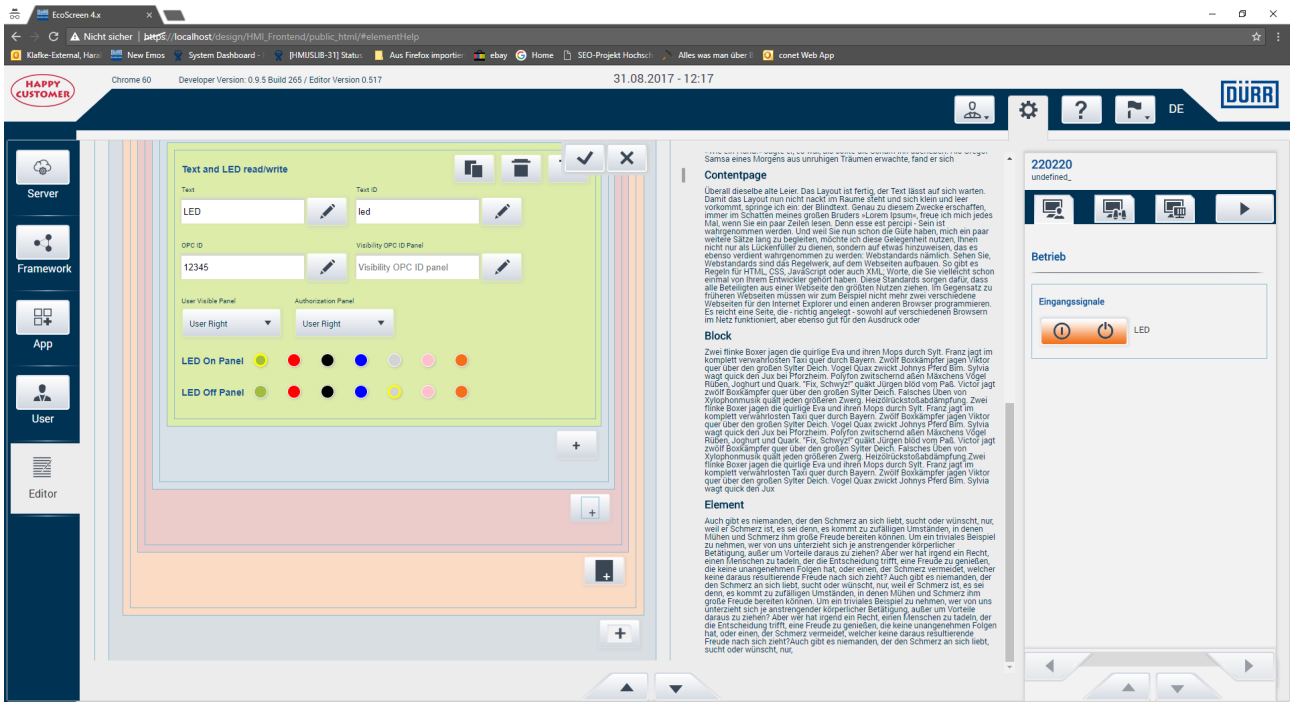
und dem Elementspezifischen Aufruf wie z.B.

3.25.function getAddTextLED(auth)

dann die Benötigte Eingabemaske zusammengestellt.



Sind dann alle Pflichtfelder ausgefüllt, wird das neue Element auch in der Vorschau sichtbar.



2.3 Editieren einer vorhandenen JSON-Datei

Haben sich in einem Stausfenster Fehler eingeschlichen, oder sind dort veraltete oder Falsche Daten eingetragen, so ist es nun ein Leichtes, dieses zu beheben.

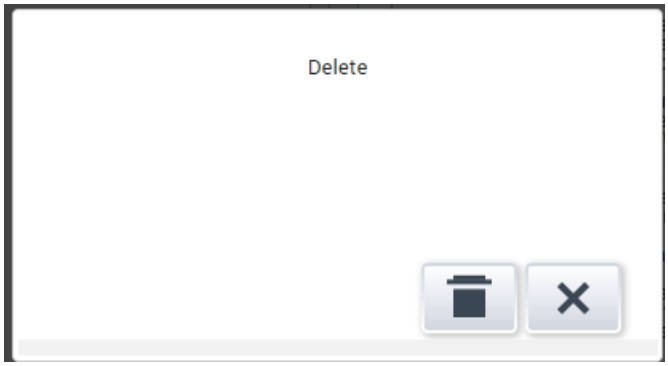
2.3.1 Edit JSON

Um eine JSON-Datei zu bearbeiten, muss ich sie natürlich öffnen. Dazu suchen wir in der entsprechenden Library nach der gewünschten Datei. Durch klick auf den „edit“-Button wird diese Datei nun im Editor geöffnet und man sieht direkt die Kopfdaten der Datei vor sich. Diese Daten sind wie wie oben bereits beschrieben nur für die Preview im Editor nötig.

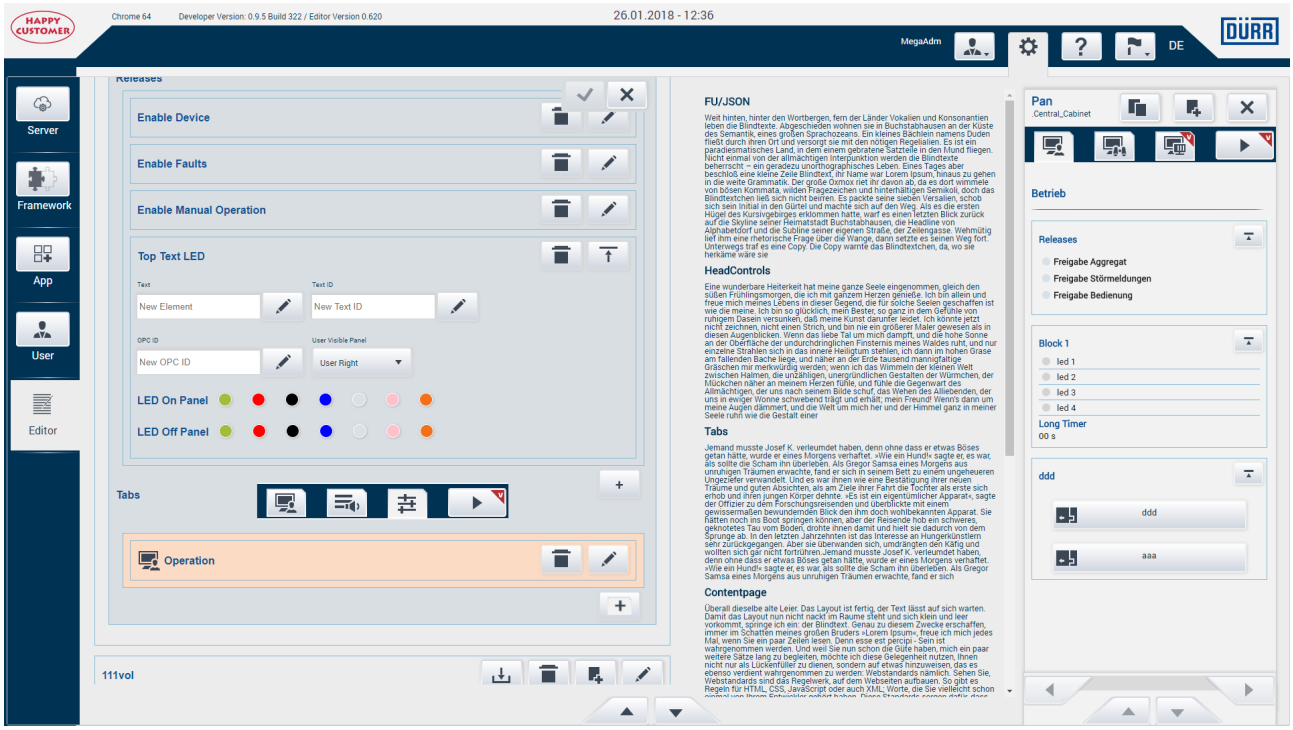
2.3.2 Edit Releases

Die Releases/Freigaben werden im Statusfenster als LED's angezeigt.

Zum Löschen einer bereits angelegten Freigabe, einfach klick auf den löschen-Button und dann eine weitere Bestätigung zum Löschen abgeben und die Freigabe ist gelöscht.



Möchte man nun eine Freigabe bearbeiten, klickt man wie immer auf den „edit“-Button und die Eingabe-Maske wird geladen.



2.3.3 Edit Tab

2.3.4 Edit Contentpage

2.3.5 Edit Block

2.3.6 Edit Element

2.4 Cloning

Statt immer wieder neu Blöcke oder Elemente anlegen zu müssen, können diese auch geklont werden

2.5 Drag and Drop

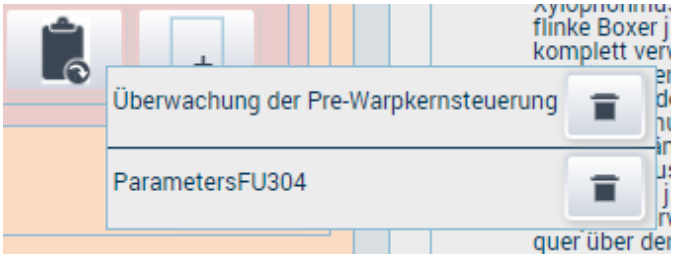
Angelegt Tabs, Contentpages, Blöcke oder Elemente können ganz einfach durch Drag and Drop noch oben bzw. unten verschoben werden, aber nur innerhalb der Liste in der sie sich befinden.

2.6 Toolbox

Viele Elemente oder gar ganze Blöcke werden in vielen Fu`s benötigt. Damit man diese nicht immer von Neuem eingeben muss, wurde die Magic der Toolbox geschaffen. Sie ist persistent und kann ganz leicht durch einen Klick befüllt werden. Die Toolbox-Daten kommen in/aus einer JSON-Datei. Fügt man ein Element oder einen Block in die Toolbox, so wird diese JSON-Datei um den entsprechenden Eintrag ergänzt. Um nun ein Element oder einen Block in die Toolbox aufzunehmen, klicken wir einfach auf den entsprechenden Button, der auf jeden Block und Element zu finden ist.

Möchte man nun ein Element oder einen Block aus der Toolbox hinzufügen, klickt man auf

und erhält eine Auswahlliste von Blöcken oder Elementen.



Dann ein Klick auf den Block/Element und es wird zur Liste hinzugefügt.

3 Funktionen

3.1 function openstatwin(data)

data = msg = die Daten, die in der JSON-Datei stehen bzw. stehen werden;

Um ein Statusfenster zu öffnen, werden die data an die emos-Funktion

new emosWS.HTMLDiagnosisWindow(msg, onReady, onAlarmReady, '.configcenter');

gesendet.

OnReady und on AlarmReady sind die Callbacks, die ausgeführt werden, sobald die DOM-Struktur und die Daten des anzuzeigenden Statusfensters vorliegen. Innherhalb diese Callbacks werden dann Funktionen ausgeführt, die das fertige Statusfenster benötigen. Soll zum Beispiel eine html-Datei in das Statusfenster eingebunden sein und sollen die PrivateTabs geladen werden. Erst wenn der Gesamte Inhalt geladen ist, kann die Headline der ersten sichtbaren Contentpage gesetzt werden, genauso wie erst dann errechnet werden kann, ob eine Scrollmöglichkeit gezeigt werden muss und wie groß dann der mögliche Scrollbalken ist.

Die onAlarmReady sorgt dafür, das im Falle vorhandener Alarme, der Message-Tab im Vordergrund steht.

3.2 function buildJsonList(parent, scroll)

Parent = parent

scroll = true/false;

Diese Funktion erstellt die Liste sämtlicher JSON-Dateien in einer Library. Betritt ein User den Editor durch Klick auf den Editor-Button wird eine Auswahlliste(radiobuttons) der vorhandenen Librarys generiert. Dann wird der Ordner der Lib 1 ausgelesen und aus der daraus resultierenden Liste eine neue Liste erstellt, in dem Versionen einzelner JSONs zusammen gefasst sind. Dann wird aus dieser neuen Liste noch zusätzlich ein DropDown als Sprung-Liste erstellt. So können JSONs, die weit unten in der Liste liegen, ohne langes Scrollen angesprungen werden.

Ist eine JSON im Editiermodus und der User schließt diese wird der Parameter scroll dem Funktionsaufruf hinzugefügt. Dies bewirkt, dass die JSON-Liste bis zu der eben editierten JSON-Datei gescrollt wird.

3.3 function rebuildJson()

Wenn eine JSON-Datei zum Editieren geladen wird, wird diese zuerst in eine andere Art Objekt umgewandelt. Dies empfinde ich als hilfreich, da sämtliche Element in der JSON-Struktur bereits eine ID haben, z.B. 10000024-AddTextLED. Da wir im Editor Elemente hinzufügen, Löschen oder auch verschieben können, müsste diese ID-Zahl vor dem Elementnamen bei jeder Preview neu berechnet werden. Da dies der Fall ist, werden diese Zahlen beim Laden der JSON herausgelöscht. Das neue Datenobjekt erhält eine etwas andere Struktur, da die Element-IDs so nicht ohne die Zahlen funktionieren können, es gibt ja mehrere AddTextLEDs.

Da aber die emos für den Aufbau des Stausfensters die alte Struktur benötigt, das

Datenobjekt natürlich wieder umgebaut werden. Dazu dient diese Funktion. Es wird mittels eines Klons des Datenobjektes, der in vielen Schleifen durchlaufen wird um die benötigten IDs zu bilden, eine neue JSON-Struktur erzeugt, die dann von der emos verarbeitet wird.

Durch spätere Änderungen, von mir TabsinTabsinTabsinTabs genannt, durchlaufen wir hier 15 Schleifen. Aufgrund dieser Menge bin ich nicht mehr sicher, ob mein ursprünglicher Ansatz eine gute Idee war. Ich persönlich würde mich freuen, wenn ein geneigter Leser, der/die eine besser Lösung für dieses Thema parat hat, mir diese dann auch mitteilt :-)

3.4 function newJson(editjson, reload, duplicateFile)

editjson = true/false; true wenn eine JSON editiert wird

reload = true/false; true wenn ein Tab gelöscht wurde

duplicateFile = true/false; true wenn eine JSON-Datei geklont wird

Diese Funktion wird direkt benötigt wenn man eine neue JSON anlegen, oder eine bestehende JSON editieren möchte. Hier wird dann die Datenstruktur und das benötigte HTML-Gerüst aufgebaut.

3.5 function editJson(parent, reload, duplicateFile)

parent = parent;

reload = true/false; true wenn ein Tab gelöscht wurde

duplicateFile = true/false; true wenn eine JSON-Datei geklont wird

Diese Funktion wird aufgerufen, wenn eine JSON editiert, dupliziert oder ein Tab aus der JSON gelöscht wird. Hier wird dann zuerst das HTML-Gerüst für den Head-Bereich aufgebaut und dann bestehende Daten in die entsprechenden Felder gesetzt. Dann wird für die vorhandenen Tabs eine Liste erstellt. Da diese Listenelemente sortierbar sind, gibt es hier die dafür zuständige Funktion. Genutzt wurde dafür die von jQuery mitgelieferte „sortable“-Funktion.

3.6 function newHeadLED(addNew)

addNew = true/false; true wenn eine neue HeadLED hinzugefügt wird

Im Headerbereich einer JSON gibt es den Bereich „Releases“. Dieser Bereich beinhaltet die Freigaben, im Statusfenster angezeigt als LEDs, als Liste dargestellt. Wenn eine JSON-Datei neu erstellt wird, werden automatisch 3 default Releases angelegt. Diese können natürlich bei Bedarf gelöscht, geändert oder ergänzt werden.

Diese Funktion stellt nun die Datenstruktur und das HTML-Gerüst zur Verfügung

3.7 function editHeadLED(parent, datanum)

parent = parent;

datanum = Die Nummer der Freigabe im Datenobjekt;

Diese Funktion stellt die Eingabemaske mit den bereits vorhandenen Daten zur Verfügung.

3.8 function newTab(addNew, selector, afterDelete)

addNew = true/false;

selector = Name des Tabs, beim Editieren

afterDelete = true/false; wird nach dem Löschen eine Contentpage benötigt

Die Tabs sind ja quasi das zentrale Menü im Statusfenster. Hier werden die Hauptbereiche von einander getrennt dargestellt. Übliche Bereiche sind z.B. Operation, Messages und Parameters. Diese sind aber nicht per default vorhanden.

Diese Funktion stellt die Datenstruktur und das Html-Gerüst, dessen Hauptteil ein DropDown-Menü ist, zur Verfügung. Dieses DropDown wird aus den im Ordner svg/tabicons liegenden Icons generiert. Möchte man einen Tab anlegen, der nicht zu den üblichen Bereichen zählt, sollte man zuerst ein entsprechendes Icon in den beschrieben Ordner legen. Danach wird dieses Icon auch in dem DropDown zur Verfügung stehen. Wählt man nun aus dem DropDown einen neuen Tab aus, wird automatisch auch die erste Contentpage generiert – siehe function newContentPage

3.9 function editTab(parent, dataFile, afterDelete)

parent = parent;

dataFile = Name des Tabs;

afterDelete = true/false; wird nach dem Löschen eine ContentPage(CP) benötigt

Ein Tab, ausgenommen der Message-Tab, besteht aus 1 – n CP. Diese Funktion stellt also das HTML-Gerüst zur Verfügung und baute eine Liste mit den vorhandenen CP auf. Da die CP sortierbar sind, findest sich hier auch eine Sortierfunktion.

3.10 function newContentPage(addNew, datanum, myText, afterDelete)

addNew = true/false;

datanum = obsolete? prüfen

myText = Name der CP; nach newTab autom. Der Tabname

afterDelete = true/false; wird benötigt nach löschen eines Blocken 1stLevel und nach duplizieren eine Blockes 1stLevel oder Einfügen einen Blockes 1stLevel aus der Toolbox.

Eine CP ist eine „Seite“ auf einem Tab. Ein Tab kann beliebig viele CPs enthalten. Ist mehr als eine CP vorhanden, kann man dies im Statusfenster an den aktivierten rechts/links Scrollbuttons und dem Pager zwischen diesen Buttons erkennen. Im Editor zeigt die Liste, welche CP bereits vorhanden sind.

Diese Funktion stellt die Datenstruktur und das Html-Gerüst zur Verfügung.

3.11 function editContentPage(parent, pageName, afterDelete)

parent = parent;

pageName = Name der ContentPage

afterDelete = true/false; wird benötigt nach löschen eines Blocken 1stLevel und nach duplizieren eine Blockes 1stLevel oder Einfügen einen Blockes 1stLevel aus der Toolbox.

Eine CP besteht aus der Headline und aus einer beliebigen Anzahl von Blöcken. Diese Funktion stellt also das HTML-Gerüst zur Verfügung und baute eine Liste mit den vorhandenen Blöcken auf. Da die Blöcke sortierbar sind, findest sich hier auch eine Sortierfunktion.

3.12 function newBlock(addNew, secondLevel)

addNew = true/false;

secondLevel = 'secondLevel' / 'thirdLevel'

Was im alten Status Window Designer ein Text bzw, eine überschrift war, wird nun zu einem Block. Warum ein Block. Es wurde definiert, dass alles, was auf einen AddText oder addHeadLine folgt, innerhalb eines blauen Rahmens gelistet wird.

Diese Funktion stellt die Datenstruktur und das Html-Gerüst zur Verfügung.

3.13 function editBlock(parent, secondLevel, childs, myChildsIDs, data, selfObjNum)

parent = parent;

secondLevel = 'secondLevel' / 'thirdLevel'

childs = die dem Block zugeordenten Elemete;

myChildsIDs = die Ids dieser Elemente innerhalb des Datenobjekts

data = die Blockdaten

selfObjNum = die Id des Blocks innerhalb des Datenobjekts

Diese Funktion stellt also das HTML-Gerüst zur Verfügung und baute eine Liste mit den zugehörigen Elementen auf. Da die Elemente sortierbar sind, findest sich hier auch eine Sortierfunktion.

3.14 function newElement(whatElement, parents, val, structure, newChildID, secondLevel)

whatElement = Name des Elements im alten emosStil, z.B. AddLed, AddTextIntegerInput

parents = array mit den Informationen zu Tab, contentpage, Block etc

val = Elementbeschreibungstext;

structure = Datenstruktur;

newChildID = die ID des neuen Elementes,

secondLevel = 'secondLevel' / 'thirdLevel';

Diese Funktion stellt die Datenstruktur und gemeinsamen Teil des Html-Gerüsts zur Verfügung. Der individuelle Teil des HTML-Gerüstes holt sich dann die entsprechende Funktion des Elementes.

3.15 function setBlockParameter()

Diese Funktion sorgt dafür, dass ein Block immer seine Kinder (Elemente) kennt und auch in der richtigen Reihenfolge.

3.16 function editElement(parent)

parent = parent;

Schaltet die Sortierung aus bevor die benötigte Elementmaske geladen und mit daten befüllt wird.

3.17 function setElementData(parent, data)

parent = parent;

data = aktueller Datensatz;

Diese Funktion fügt die Daten aus dem aktuellen Datensatz in die aktuell sichtbaren Inputfelder, DropDownauswahlisten und Checkboxen ein.

3.18 function getJsonId()

Liefert die von der emos benötigte ID für die einzelnen Elemente zurück, z.B. 1000024

3.19 function validatePreviewJson()

Bevor eine Preview auf das Statusfenster generiert wird, überprüft diese Funktion, ob alle benötigten Felder ausgefüllt wurden. Pflichtfelder, die keine Daten enthalten werden rot umrandet dargestellt. Sind alle Felder ausgefüllt, wird aus dem aktuellen Datenobjekt eine

für die emos verwertbare Jsonstruktur erzeugt und an die emos gesendet um ein Statusfenster aufzubauen.

3.20 function buildLEDPanel(status)

status = 'on'/'off';

Diese Funktion generiert die LED-Farbauswahl.

3.21 function getAuthDropDown()

Generiert das Dropdown für die Rechteauswahl beim User visible Panel und Authorization Panel.

3.22 function getElementHead()

Generiert den Kopf einer Elementmaske

3.23 function getMainInput(withoutText, withoutOPCID, secondLevel)

withoutText = true/false;

withoutOPCID = true/false;

secondLevel = '' / 'secondLevel' / 'thirdLevel';

Generiert die Eingabefelder, die bei den meisten Elementmasken benötigt werden.

3.24 function getAuthMinMAx(minmax, intTime, secondLevel)

minmax = true/false;

intTime = true/false;

secondLevel = '' / 'secondLevel' / 'thirdLevel';

Liefert das HTML für die Bedienerberechtigung, min-max Eingabefelder für Value und Time.

3.25 function getUnitInput(secondLevel)

secondLevel = '' / 'secondLevel' / 'thirdLevel';

Liefert die Eingabefelde für die Units metric, imperial und local, sowie einen Satz Radiobuttons für die Auswahl.

3.26 function getValueBase(secondLevel)

secondLevel = '' / 'secondLevel' / 'thirdLevel';

Liefert die Eingabefelder für die Valuebase.

3.27 function getHeadLED()

Liefert die Eingabemaske für die Releases.

3.28 function getAddTextLED(auth), function getAddTextInteger(auth, real), function getAddString(auth), function getAddTextByte(auth), function getAddDoubleText(), function getAddTextLEDInteger(auth), function getAddTextIntegerInteger(auth), function getAddTextRealRealLong(auth), function getAddLEDTextReal(auth), function getAddLEDTextRealReal(auth), function getAddTextTimer(auth, valuebase), function getAddDoubleTextTimer(auth), function getAddDoubleTextTimerMin(auth), function getAddTextSiemensDate(auth), function getAddStringField(), function getAddBrowserPage(), function getAddBitBar(), function getAddBar(), function getAddSetpointBar()

Diese Funktionen stellen jeweils die Eingabemaske für die entsprechenden Elemente zur Verfügung.

3.29 function saveJson(parent)

Bevor das aktuelle Datenobjekt umgewandelt und abgespeichert wird, wird eine neue Buildnummer errechnet und die Version angepasst. Danach wird das Datenobjekt in die benötigte JSON-Struktur umgebaut, ein neuer Hashwert errechnet und auf dem Server als JSON-Datei abgespeichert.

3.30 function getElementMask(val, parents, parent, newChildID, img)

val = Beschreibungstext des Elementes;

parents = array mit den Informationen zu Tab, contentpage, Block etc

parent = parent

newChildID = die ID des neuen Elementes,

img = Name des Bildes für eine Elementansicht

Bildet für jedes Element die benötigte Datenstruktur, den Elementrahmen und die Elementmaske.

3.31 function getElementEditMask(val, parent)

val = emos-Name des Elementes (z.B. AddTextLED);

parent = parent;

Sorgt dafür, dass die für das Element benötigte Eingabemaske geladen wird.

3.32 function downloadJson(filename)

filename = Name der JSON-Datei;

Funktion um eine der JSON-Dateien auf den lokalen Computer zu speichern.

3.33 Array.prototype.move

Prototype.Funktion um eine Element in einem Array zu verschieben.

3.34 String.prototype.hashCode

Prototype.Funktion um einen hashCode zu erzeugen.