# Créer un form2

## Objectif :



## Mise en place :

On indique :

* L’enregistrement à afficher au moyen de la propriété ***record***
* Eventuellement les champs à afficher au moyen de la propriété ***fields***

<!DOCTYPE html>

<html>

<head>

<title>**Form**</title>

<meta charset=**"UTF-8"**/>

<link href=**"css/jsa.css"** rel=**"stylesheet"** media=**"all"** type=**"text/css"**>

<link href=**"css/multiDisplay.css"** rel=**"stylesheet"** media=**"all"** type=**"text/css"**>

<link href=**"css/editor2.css"** rel=**"stylesheet"** media=**"all"** type=**"text/css"**>

<link href=**"img/favicon.png"** rel=**"icon"** media=**"all"** type=**"text/css"**>

<meta name=**"viewport"** content=**"width=450, user-scalable=no"**>

<meta charset=**"utf-8"**>

<style>

</style>

</head>

<body>

<script src=**"js/jsa.js"**></script>

<script src=**"js/jsa.cls.js"**></script>

<script src=**"js/jsa.appliMobile.js"**></script>

<script src=**"js/multiDisplay.js"**></script>

<script src=**"js/form2.js"**></script>

<script src=**"js/formInput.js"**></script>

<script src=**"js/formBlock.js"**></script>

<script src=**"js/selector.js"**></script>

<script src=**"js/editor2.js"**></script>

<script src=**"js/jsa.application.js"**></script>

<script>

</script>

<script>

jsa.appliMobile**({**

 id**:**"myOrganization"**,**

 label**:**"My organization"**,**

 objects**:[],**

 displayBody**:*function*(){**

 ***return*** **{**

 tag**:**"div"**,**

 className**:**"appliMobile\_body"**,**

 owner**:*this*,**

 isElt**:**"\_divBody"**,**

 children**:[{**

 tag**:**"form2"**,**

 fields**:[**

 **{**tag**:**"formInput"**,** id**:**"id"**,** label**:**"Id"**,** type**:**"string"**},**

 **{**tag**:**"formInput"**,** id**:**"prop1"**,** label**:**"Property 1"**,** type**:**"string"**},**

 **{**tag**:**"formInput"**,** id**:**"prop2"**,** label**:**"Property 2"**,** type**:**"uint"**},**

 **{**tag**:**"formInput"**,** id**:**"prop3"**,** label**:**"Property 3"**,** type**:**"date"**}**

 **],**

 record**:{**id**:**"item1"**,** label**:**"Item 1"**,** prop1**:**"aaa1"**,** prop2**:**15**,** prop3**:**"15-03-2023"**}**

 **}]**

 **};**

 **}**

 **});**

appli.mobile**=*true*;**

appli.smallScreen**=*true*;**

</script>

</body>

</html>

**Remarque**

*fields* est optionnelle : si elle est omise les champs seront déduits des propriétés de l’enregistrement :



### fields

**field**={

 id (string)

 label (string)

 type2 (string) Type de données

 idObj #objectDefinition Seulement si type2=obj ou array

 placeholder (string) : Texte à afficher par défaut

 enabled (boolean|exprJs|fn)

 visible (boolean|exprJs|fn)

 showLabel (boolean) Affiche le label

 possibleValues {possibleValues}

 proposedValues {possibleValues}

 dependant (boolean) la liste déroulante (le cas échéant) se mettra à jour à chaque fois que la valeur d'un champ sera modifié dans le formulaire

 block #block

 style {cssStyle}

 }

#### type2

Les valeurs possibles sont les suivantes :

* *number*,
* *int*,
* *uint*,
* *boolean*
* *string*,
* *longText*
* *date*
* *time*
* *dateTime*,
* *phone*
* *email*
* *obj* : Correspond à un enregistrement (à afficher en principe dans un onglet). A associer à la propriété *idObj* pour indiquer la définition de l’objet
* *array* Correspond à un tableau d’enregistrements (à afficher en principe dans un onglet). A associer à la propriété idObj pour indiquer la définition de l’objet
* *calculated* : champ calculé (voir fiche form2 - remplir un champ calcule.docx)

#### enabled et visible

3 façons de leur affecter une valeur :

* Par **valeur booleene**

{tag:"formInput", id:"multiselect", label:tr("Multiselect"), main:true, type:"boolean", defaultValue:true, enabled:**false**},

* Par **expression javascript**

{tag:"formInput", id:"multiselect", label:tr("Multiselect"), main:true, type:"boolean", defaultValue:true, enabled:**"$[info1]>15"**, block:"behaviour"},

* Par **fonction**

{tag:"formInput", id:"multiselect", label:tr("Multiselect"), main:true, type:"boolean", defaultValue:true, enabled:**function(record, form){return record.info1 > 15}**, block:"behaviour"},

#### possibleValues et proposedValues

Il est possible de leur affecter une valeur de multiples façons :

* Par **tableau de valeurs**

{tag:"formInput", id:"height", label:tr("Height"), type:"uint", block:"styles", possibleValues**:["10px", "20px", "30px"]**},

* Par tableau de paires clés-valeurs

{tag:"formInput", id:"height", label:tr("Height"), type:"uint", block:"styles", possibleValues**:[**

 **{id:"size1", label:"10px"},**

 **{id:"size2", label:"20px"},**

 **{id:"size3", label:"30px"}**

 **]**

},

* Par fonction

Elle renvoie un tableau de valeurs ou de paires clés-valeurs

{tag:"formInput", id:"height", label:tr("Height"), type:"uint", block:"styles", possibleValues:**function(formInput){**

 **return ["10px", "20px", "30px"]**

 **}**

},

* Par expression javascript

fields:[

 {tag:"formInput", id:"height", label:tr("Height"), type:"uint", block:"styles", possibleValues**:"$ listValues('[info1]', '[info2]')"**},

 {tag:"formInput", id:"info1"},

 {tag:"formInput", id:"info2"}

 ]

var listValues=function(val1, val2){

 debugger;

 return [

 {id:"10px", label:"10 px"},

 {id:"20px", label:"20 px"},

 {id:"30px", label:"30 px"},

 {id:val1, label:val1},

 {id:val2, label:val2}

 ];

 }

* Par référence à un lexique

{tag:"formInput", id:"height", label:tr("Height"), type:"uint", block:"styles", possibleValues:**"lexicon1"**},

appli.lexicons=[

 {

 id:"lexicon1",

 children:[

 {id:"10px", label:"10 px"},

 {id:"20px", label:"20 px"},

 {id:"30px", label:"30 px"}

 ]

 }

 ];

#### Rendre une liste dépendante

Une liste peut être rendue **dépendante** des autres champs. Cela veut dire qu'elle sera mise à jour à chaque fois que l'utilisateur change la valeur de n'importe quel autre contrôle dans le formulaire

dependant (boolean)

ex:

{tag:"formInput", id:"type", label:"Type", type2:"string", block:"tabMain", dependant:**true**, possibleValues:"$ appli.getPossibleTypesObjects(record)"},

#### Taille des labels

La propriété ***widthLabels*** permet de définir la longueur de tous les labels des champs





### Organisation en blocs

Il est possible d’organiser les champs en blocs comme ci-dessous :



Pour cela on définit la propriété ***blocks*** :



Au sein de la propriété ***fields***, chaque champ référence le bloc sous lequel apparaître par sa propriété ***block***

**Remarque**

Ne pas oublier d’inclure le fichier ***formBlock.js*** :



**Attention**

L’identifiant des blocs est une chaîne de caractère, le **point décimal est interdit**.

#### Bordures

La propriété ***border*** permet de définit le style de bordure. Elle peut recevoir toutes les valeurs d’attribut de style CSS *border-style* : *solid*, *dotted*, *dashed*, *double*, *none*, …

On peut s’en servir notamment pour rendre une bordure invisible :





#### Blocs rétractables

La propriété ***collapsable*** permet à l’utilisateur de rétracter ou développer un bloc :





Une fois déployé :



Le bloc apparaît contracté à l’ouverture. Pour changer ce comportement par défaut on utilise la propriété ***collapsed*** :



#### Blocs masqués

La propriété ***visible*** permet d’afficher ou non bloc. Elle peut recevoir une valeur booléenne, une expression JavaScript ou une fonction

Voir fiche ***bloc - rendre invisible.docx***

#### Blocs imbriqués

Il est possible d’imbriquer les blocs les uns dans les autres :





#### Onglets

Il est possible de créer des onglets :





**Remarque**

Il est nécessaire d’inclure le fichier ***tabs.js***



Ils peuvent contenir d’autres blocs :





Les onglets peuvent inclure d’autres onglets :





Ces derniers peuvent contenir à leur tour d’autres blocs (autres *tab*, *formBlock* ou autres, sur autant de niveaux d’inclusion que désiré)



#### Autres conteneurs

A la place des *tab* et des *formBlock* tous autre type d’élément HTML est permis :





**Remarque**

On peut se servir de cette possibilité pour inclure ce que l’on veut en HTML, par exemple des boutons :





### Affichage de sous-objets

Les propriétés de l’enregistrement record peuvent être des objets ou des tableaux. On peut alors les afficher sous la forme de form ou de grid imbriqués :



Dans cet exemple l’enregistrement se présente ainsi :



On remarque que

* la propriétés prop3 est un objet
* la propriété prop4 est un tableau

Il faut définir les champs ainsi :



On remarque qu’il faut définir :

* ***type*** : *obj* ou *array*
* **idObj** : identifiant de la définition d’objet à utiliser. Cela oblige à créer une définition d’objet au préalable mais nous verrons plus loin une alternative.

Enfin il faut définir les objets en question dans l’application :



Le plus adapté graphiquement est de les placer dans des onglets :







Remarquer l’emploi de la propriété showLabel=false pour que les labels n’apparaissent pas (celui indiqué dans l’onglet est suffisant) :



**Alternative pour ne pas avoir à définir d’objet**



Le résultat :



### Quelques propriétés utiles

* ***\_oldValues*** : donne la liste des valeurs qui ont été changées avant leur changement

\_oldValues = {id1 :oldValue1, …, idN :oldValueN}

* ***\_changedValues*** : donne la liste des valeurs qui ont été changées après leur changement :

\_changedValues = {id1 :newValue1, …, idN :newValueN}

### Fonctions standards

#### onOpen()

Elle s’exécute à l’ouverture de la form2

onOpen:function(){

 // The form was opened

},

#### onchange()

Elle s’exécute à chaque modification d’un champ par l’utilisateur

onChange:function(ctrl, evt){

 // A value has been changed by the user

 },

### Fonctions personnalisées

Il est possible de définir ses **propres fonctions**. Il est aussi possible d’écraser des **fonctions standard** (comme *onOpen*()). Pour cela on utilise la propriété ***fns***.

Voir fiche ***form2 - definir des fonctions personnalisees.docx***

### Accès aux contrôles et aux blocs

* Les contrôles sont accessibles par la propriété **\_ctrls**
* Les blocs par la propriété **\_*blocks***

