

Planning Guide (de)

Apparo Fast Edit

Version 3.2.2



Inhalt

1	Einleitung	3
2	JavaScript Selektor ID	4
2.1	Aufbau der ID	4
3	Geschäftslogik im Web-Browser	5
3.1	Limitierungen	6
4	Einsatz in einem Table Business Case.....	7
4.1	Aktivieren der Funktion.....	7
4.2	Verfügbare JavaScript Methoden	8
5	Einsatz in einem Single Business Case	9
5.1	Aktivieren der Funktion.....	9
5.2	Verfügbare JavaScript Methoden	9
6	Lesen/Schreiben von Widget-Werten	10
6.1	Lesen von Widget-Werten.....	10
6.2	Schreiben von Widget-Werten	10
6.3	Beispielfunktion.....	10
6.3.1	Im Detail.....	10
6.3.2	Einsatz in Apparo.....	11
6.4	Möglichkeiten einer Checkbox.....	12
6.5	Möglichkeiten von Lookup-Widgets.....	13
6.5.1	Lookup-Schlüsselwerte.....	13
6.5.2	Lookup-Ausgabewerte (Label)	13
6.6	Aggregieren aller Werte einer Spalte in einem Table Business Case	14
6.6.1	Beispiel für Summe über eine Spalte	14
6.7	Einsatz von Variablen	15
7	Einsatz von größeren JavaScript-Programmen	16
8	Enter-Key für den Aufruf der JavaScript-Routine.....	16
9	Beispiel eines Table Business Cases für Planung	17

1 Einleitung

Für die Abbildung von Planungsfunktionalitäten verwenden wir die Möglichkeit von Fast Edit, **Client-seitig JavaScript auszuführen**. Hierfür gibt es eine Reihe von zusätzlichen JavaScript Methoden zum Lesen/Schreiben von/in Widgets im Edit- und Kalkulationsbereich, die es uns ermöglichen Geschäftslogiken zu definieren.

Client-seitig bedeutet, dass JavaScript im Web-Browser ohne Interaktion mit dem Server ausgeführt wird.

Vorteil: Zahlen verteilen auf Monate und Summierungen aktualisieren erfolgen sofort nachdem der Anwender mit der Tab- oder Enter-Taste ein Wert angepasst hat. Es treten keine Wartezeiten auf, da es keine Kommunikation mit dem Server gibt.

Als Beispiel für den Einsatz wird eine kleine Planungsapplikation entwickelt.

2 JavaScript Selektor ID

Für die eindeutige Zuordnung verfügt jedes Widget über eine JavaScript Selektor ID.

Die JavaScript Selektor ID finden Sie in den Widget-Einstellungen unter Widget Typ:



Zum Kopieren der ID, können sie den Button rechts neben der ID verwenden

2.1 Aufbau der ID

Die Selektor ID sieht wie folgt aus:

.jsID_E_0_0

Der erste Teil der ID ist die Abkürzung für JavaScript ID

.jsID_E_0_0

Der zweite Teil der ID beschreibt den Bereich, in dem das Widget verwendet wird

.jsID_E_0_0

E steht für Edit-Bereich und C steht für Kalkulationsbereich

Die beiden Ziffern stehen für die jeweilige Spalte und Nummer, in der das Widget verwendet wird

.jsID_E_0_0

Der Zähler startet bei Null und wird immer um 1 erhöht.

Die erste Ziffer identifiziert die Spalte, in der das Widget steht und wird in Single Business Cases verwendet. In Table Business Cases bleibt ist der Zähler immer 0, da hier keine Spalten verwendet werden.

Die zweite Ziffer, bildet die fortlaufende Nummerierung der Widgets ab. Verändern Sie die Reihenfolge der Widgets, dann ändert sich auch die ID.

3 Geschäftslogik im Web-Browser

Eigene JavaScript-Geschäftslogik kann **automatisch** ausgeführt werden, wenn der Anwender im

- **Single Business Case** oder
- **Table Business Case**

im Einfüge oder Änderungs-Modus:

- eine **Checkbox** setzt und löscht
- ein **Eingabefeld** verlässt (oder die Eingabetaste drückt)
- in einem **Lookup Widget (für alle Tabellen)** einen Wert auswählt, ohne Eingabemöglichkeit (d.h. Anwender kann nur Werte auswählen aber nicht die Auswahl der Werte einschränken)

Danach kann automatisch im Browser eine JavaScript-Routine ausgeführt werden, um andere Widget-Werte zu verändern:

- Widget Label
- Widget Label mit Variablen
- Eingabefeld Widget

Achtung: Nur die Widgets der **aktuellen** Datenzeile sind änderbar, ebenso alle Kalkulations-Widgets.

Beispiel:

1. Der Anwender setzt eine Checkbox oder ändert eine Zahl in einem Eingabefeld und verlässt dieses Eingabefeld
2. Die selbstdefinierte JavaScript-Routine wird gestartet.

Die Routine kann nun Werte von anderen Widgets lesen und den Widget-Wert vom Typ Label, Label mit Variablen oder Eingabefeld ändern ohne das ein Submit erfolgt.

In einem Table Business Case können auch die aktuellen Werte einer Spalte summiert werden und z.B. in einem Kalkulations-Widget ausgegeben werden.

3.1 Limitierungen

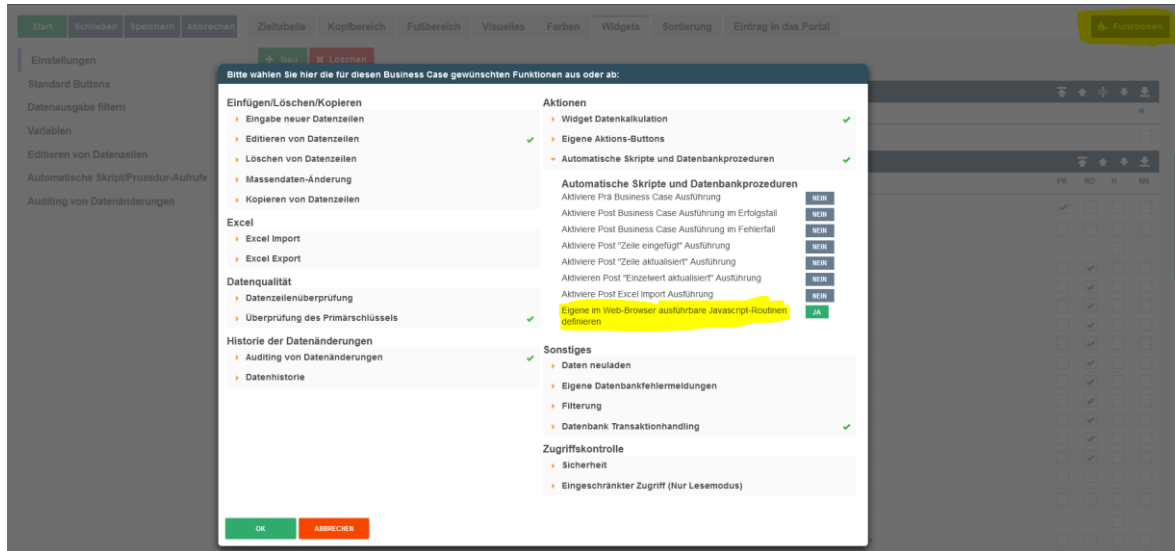
Durch die Ausführung im Web-Browser ergeben naturgemäß einige Einschränkungen:

- Variablen können nur eingeschränkt verwendet werden, diese werden vorab Server-seitig berechnet
- Einen ActionBC könnten Sie zwar mit `window.open` starten, aber die (Zwischen)Ergebnisse der Berechnungen können so nicht gespeichert werden, da der Zugriff auf die Widget-Referenzen der Berechnungen fehlt

4 Einsatz in einem Table Business Case

4.1 Aktivieren der Funktion

Aktivieren Sie die Funktion „**Eigene im Web-Browser ausführbare JavaScript-Routinen definieren**“ unter Funktionen in den Widget-Einstellungen:



4.2 Verfügbare JavaScript Methoden

Das allgemeine Format für **get** und **set** Methoden:
getAfeWidgetValue(targetElementSelector)

muss im Table Business Case im Edit-Bereich wie folgt erweitert werden:
getAfeTableWidgetValue(sourceElement, targetElementSelector)

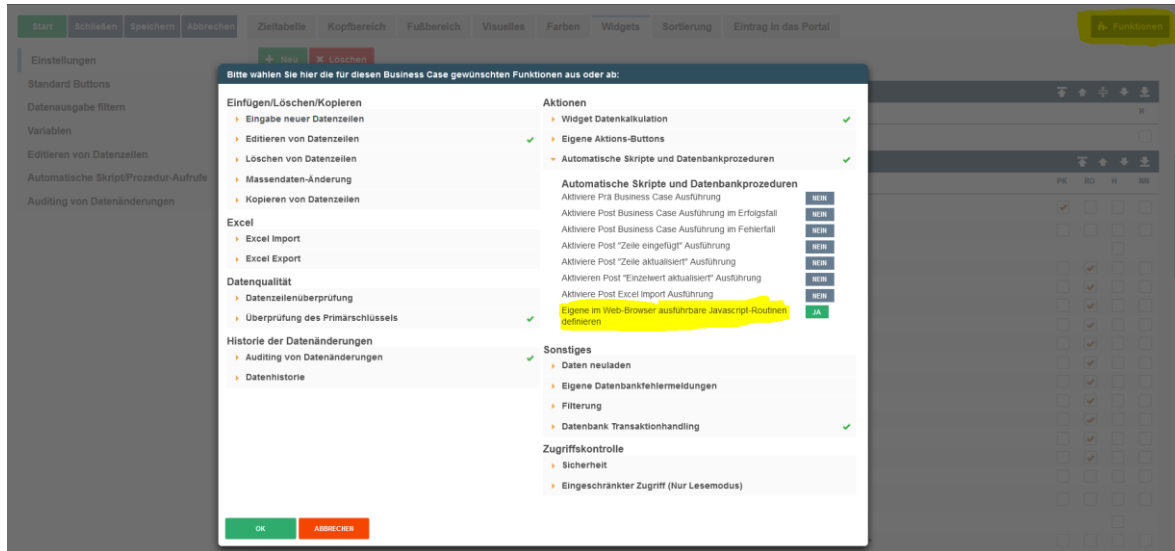
Table ist ein Hinweis für das Programm und sourceElement bezieht sich auf die aktuelle Zeile und **ist immer** 'this'.

Ziel	Befehl
Lesen des Widget-Wertes der aktuellen Zeile, Wert ist eine Zahl	<code>getAfeTableWidgetNumValue(this, '.jsID_E_0_1');</code>
Lesen des Widget-Wertes der aktuellen Zeile, Wert ist ein String	<code>getAfeTableWidgetStringValue(this, '.jsID_E_0_1');</code>
Schreiben in ein Widget der aktuellen Zeile, Wert ist eine Zahl	<code>setAfeTableWidgetNumValue(this, '.jsID_E_0_2', calcValueNum);</code>
Schreiben in ein Widget der aktuellen Zeile, Wert ist ein String	<code>setAfeTableWidgetStringValue(this, '.jsID_E_0_2', calcValueNum);</code>
Lesen des Wertes eines Kalkulations-Widgets , Wert ist eine Zahl	<code>getAfeWidgetNumValue('.jsID_E_0_1');</code>
Schreiben eines Wertes in ein Kalkulations-Widget , Wert ist ein String	<code>setAfeWidgetStringValue('.jsID_C_0_1', 'CHANGED');</code>
Lesen eines Lookup-Widget Labels	<code>var myLabelValue = getAfeTableWidgetLookupLabel(this, '.jsID_E_0_3');</code>
Lesen/aggregieren aller numerischen Werte einer Spalte der aktuellen Seite	<code>getAfeTableColumnFunction('.jsID_E_0_9', 'sum'); getAfeTableColumnFunction('.jsID_E_0_9', 'min'); getAfeTableColumnFunction('.jsID_E_0_9', 'max'); getAfeTableColumnFunction('.jsID_E_0_9', 'avg');</code> Null-Werte werden übersetzt mit 0-Werte

5 Einsatz in einem Single Business Case

5.1 Aktivieren der Funktion

Aktivieren Sie die Funktion „Eigene im Web-Browser ausführbare JavaScript-Routinen definieren“ unter Funktionen in den Widget-Einstellungen:



5.2 Verfügbare JavaScript Methoden

Ziel	Befehl
Lesen eines Widget-Wertes, Wert ist eine Zahl	<code>getAfeWidgetNumValue('.jsID_E_0_1');</code>
Lesen des Widget-Wertes, Wert ist ein String	<code>getAfeWidgetStringValue('.jsID_E_0_1');</code>
Schreiben in ein Widget, Wert ist eine Zahl	<code>setAfeWidgetNumValue('.jsID_E_0_2', calcValueNum);</code>
Schreiben in ein Widget, Wert ist ein String	<code>setAfeWidgetStringValue('.jsID_E_0_2', calcValueNum);</code>
Lesen des Wertes eines Kalkulations-Widgets , Wert ist eine Zahl	<code>getAfeWidgetNumValue('.jsID_E_0_1');</code>
Schreiben eines Wertes in ein Kalkulations-Widget , Wert ist ein String	<code>setAfeWidgetStringValue('.jsID_C_0_1', 'CHANGED');</code>
Lesen eines Lookup-Widget Labels	<code>var myLabelValue = getAfeWidgetLookupLabel('.jsID_E_0_3');</code>

6 Lesen/Schreiben von Widget-Werten

6.1 Lesen von Widget-Werten

Mit der Methode ***getAfeWidgetValue(JavaScriptSelektorID)*** können Sie beliebige Widget-Werte auslesen. Die *JavaScriptSelektorID* identifiziert dabei das Widget, dessen Wert wir auslesen wollen.

Für numerische Werte (Zahlen) verwenden Sie *getAfeWidgetNumValue* und
Für String-Werte (Zeichenketten) verwenden Sie *getAfeWidgetStringValue*

6.2 Schreiben von Widget-Werten

Mit der Methode ***setAfeWidgetValue(JavaScriptSelektorID, Wert)*** können Sie in Widgets Werte schreiben. Die *JavaScriptSelektorID* identifiziert dabei das Widget, in das wir schreiben wollen und ***Wert*** den Wert, (z.B. eine Zahl) der geschrieben werden soll.

Für numerische Werte (Zahlen) verwenden Sie *setAfeWidgetNumValue* und
Für String-Werte (Zeichenketten) verwenden Sie *setAfeWidgetStringValue*

6.3 Beispielfunktion

In diesem Beispiel wird der Wert des Widgets mit der Referenz ID ***.jsID_E_0_0*** gelesen und der Wert des Widgets **** 2*** wieder in das Widget ***.jsID_E_1_2*** gespeichert, wenn der Anwender das Widget ***.jsID_E_0_0*** verlässt.

```
$(document).on('change', '.jsID_E_0_0', function(){
    var myValue = getAfeWidgetNumValue('.jsID_E_0_0');
    setAfeWidgetNumValue('.jsID_E_1_2', myValue * 2);
})
```

6.3.1 Im Detail

```
$(document).on('change', '.jsID_E_0_0', function()
```

Startet eine JavaScript Funktion ***function()*** wenn im Web-Browser ***\$(document)*** ein Wert im Widget ***.jsID_E_0_0*** geändert wird ***on('change')***.

Der Inhalt der JavaScript Funktion steht in den geschweiften Klammern.

```
{ var myValue = getAfeWidgetNumValue('.jsID_E_0_0');
```

Definiert ***var*** und füllt die JavaScript Variable ***myValue*** mit dem Wert des Widgets ***.jsID_E_0_0***

```
setAfeWidgetNumValue('.jsID_E_1_2', myValue * 2); }
```

Schreibt den Inhalt der JavaScript Variablen ***myValue*** multipliziert mit ***2 * 2*** in das Widget ***.jsID_E_1_2*** als numerischer Wert ***setAfeWidgetNumValue***

Hinweis

Numerischer Wert ist hier wichtig, das System kann so sprach-spezifische Zahlenformate automatisch verwerten, andernfalls gäbe es Probleme z.B. bei der Verwendung verschiedener Dezimaltrenner (123.45 und 123,45)

6.3.2 Einsatz in Apparo

Start
Schließen
Speichern
Abbrechen

Einstellungen

Standard Buttons

Datenausgabe filtern

Variablen

Einfügen von neuen Datenzeilen

Editieren von Datenzeilen

Datenzeilen manuell löschen

Excel Import

Excel Export

Automatische Skript/Prozedur-Aufrufe

Automatische Skript/Prozedur-Aufrufe

Mit der Prä/Post-Ausführung ist es möglich (Shell-, SQL-, JS-)Skripte und Datenbankprozeduren/-funktionen automatisch auszuführen bei Eintritt von bestimmten Ereignissen. Es ist möglich diese zu starten,

- bevor ein Business Case(auch server-seitig automatisch) den Dateiimport startet oder nachdem er ihn beendet.
- beim Start oder Beenden des Excel Zeilenimports
- nachdem ein Anwender Daten eingefügt oder geändert hat

Dieses Verhalten kann global oder für bestimmte Anwendergruppen definiert werden. So können für bestimmte Anwendergruppen gesonderte Skripte usw. ausgeführt werden und in allen anderen Fällen werden die voreingestellten Skripte aufgerufen. Momentan unterstützt Apparo Oracle, Microsoft SQL Server, IBM DB/2, IBM dashDB, Sybase ASE, Teradata und SAP HANA Datenbanken.

Ein SQL Script ist eine Textdatei, die SQL-Anweisung enthält und die Endung .sql hat. Die Anweisung werden innerhalb der Datenbanksitzung ausgeführt, die der Business Case nutzt. Anweisungen sind mit Strichpunkt getrennt.

Eigenes Javascript, das im Browser ausgeführt wird, kann mit dem Feature "Eigenes Javascript" hier definiert werden.

Eigene im Web-Browser ausführbare Javascript-Routinen definieren

Sie können auch Variablen verwenden. Das vollständige Javascript ist Teil der Webbrowser-Ausgabe und wird nur im Webbrowser ausgeführt.

Wenn Sie größere Routinen verwenden, können Sie <%FILE_CONTENT(Dateipfad + Name)%> verwenden und die Befehle als Datei auf dem Server speichern.

Sie können Widgets je nach Benutzerverhalten ein- / ausblenden / berechnen. Weitere Informationen finden Sie im Benutzerhandbuch.

```

$(document).on('change', '#jsID_E_0_0', function(){
    var myValue = getAfeWidgetValue('#jsID_E_0_0');
    setAfeWidgetValue('#jsID_E_1_2', myValue * 2);
})
                
```

6.4 Möglichkeiten einer Checkbox

Checkboxes können unabhängig vom gesetzten Wert (meist 0 und 1 oder Y und N) gesetzt werden. Mit true wird das Häkchen gesetzt, mit false wird das Häkchen entfernt.

```
setAfeWidgetStringValue('.jsID_E_0_3', true);
```

Mit diesem Aufruf wird das Häkchen (=true) im Checkbox-Widget mit der Referenz `'.jsID_E_0_3'` gesetzt. Da true eine Zeichenkette ist (String Wert) verwenden wir die Methode **setAfeWidgetStringValue**

Checkboxes können auch einfach versteckt werden.

Im folgenden Beispiel wird die Checkbox, abhängig von ihrem Wert versteckt:

Beispiel:

```
$(document).on('change', '.jsID_E_0_3', function(){  
  var myValue = getAfeWidgetNumValue('.jsID_E_0_3');  
  if(true == myValue) {
```

```
  document.querySelector('.jsID_E_0_4').parentElement.parentElement.parentElement.parentElement.style.display = "none";  
  }  
  else {
```

```
  document.querySelector('.jsID_E_0_4').parentElement.parentElement.parentElement.parentElement.style.display = "table-row";  
  }  
});
```

Falls die überprüfte Checkbox gesetzt ist, wird das Checkbox Widget `.jsID_E_0_4` **versteckt** oder andernfalls (wieder) **angezeigt**.

6.5 Möglichkeiten von Lookup-Widgets

Lookup Widgets können nur gelesen, aber nicht gesetzt werden.

6.5.1 Lookup-Schlüsselwerte

Die Lookup-Schlüsselwerte können so gelesen werden:

Im Table Business Case:

```
getAfeTableWidgetNumValue(sourceElement, targetElementSelector)
var myLabelValue = getAfeTableWidgetNumValue(this, '.jsID_E_0_4');
```

Im Single Business Case:

```
getAfeWidgetNumValue(targetElement)
var myLabelValue = getAfeWidgetNumValue('.jsID_E_0_4');
```

6.5.2 Lookup-Ausgabewerte (Label)

Die Lookup-Ausgabewerte können so gelesen werden:

Im Table Business Case:

```
getAfeTableWidgetLookupLabel(sourceElement, targetElementSelector)
var myLabelValue = getAfeTableWidgetLookupLabel(this, '.jsID_E_0_4');
```

Im Single Business Case:

```
getAfeWidgetLookupLabel(targetElement)
var myLabelValue = getAfeWidgetLookupLabel('.jsID_E_0_4');
```

6.6 Aggregieren aller Werte einer Spalte in einem Table Business Case

Es ist möglich Berechnungen über alle verwendeten Zeilen eines Widgets (=Spalte) durchzuführen. Die Spalte muss numerisch sein.

Es werden alle Werte nur der aktuellen Seite (=alle sichtbaren Datenzeilen) beachtet.

Die Summe kann z.B. in einem Kalkulations-Widget ausgegeben werden.

Produkt	Summe Jahr	Januar	Februar	März	April	Mai	Juni	Juli	August	September	Oktober	November	Dezember	Arbeitsschritt
Bino Man	689.679,40					9.000,00	50.000,00	60.000,00	320,10	410.317,50	190.317,51	-30.317,61	41,90	offen
Dark Cap	133.829,40	1.000,00	1.000,00	1.000,00	1.000,00	22.623,56	110.000,00	8.000,00	30.000,00	-23.433,00	-8.680,08	-8.680,08	999,00	Bereit für Controlling
Bags New York	9.906,70	222,00				815,50	-5,36	-5,36	-5,36	-5,36	8.888,00	-5,36	8,00	Bereit für Controlling
T-Shirt 69's	20.010,00		222,00			2.482,25	2.472,25	2.472,25	2.472,25	2.472,25	2.472,25	2.472,25	2.472,25	Bereit für Controlling
		849.912,35	1.222,00	1.222,00	1.000,00	1.000,00	34.921,31	162.466,89	70.466,89	32.786,99	389.351,39	192.997,68	-37.530,80	3.521,15

getAfeTableColumnFunction(targetColumnSelector, functionName)

targetColumnSelector bezieht sich auf das Widget, für das die Funktion über alle Zeilen berechnet werden soll.

functionName bezieht sich auf die Funktion und kann eine der folgenden sein:

- **Summe:** `getAfeTableColumnFunction('.jsID_E_0_3', 'sum')`
- **Minimum:** `getAfeTableColumnFunction('.jsID_E_0_3', 'min')`
- **Maximum:** `getAfeTableColumnFunction('.jsID_E_0_3', 'max')`
- **Durchschnitt:** `getAfeTableColumnFunction('.jsID_E_0_3', 'avg')`

6.6.1 Beispiel für Summe über eine Spalte

Hierfür benötigen wir ein Kalkulations-Widget (ohne Inhalt) vom Typ Label mit Variablen für die Ausgabe. Der Ausgabewert wird vom Skript berechnet und eingetragen.

```
var Summe;
```

Hiermit definieren wir die JavaScript Variable **Summe**

```
Summe = getAfeTableColumnFunction('.jsID_E_0_3', 'sum');
```

Hiermit befüllen wir die Variable **Summe** mit der Berechnung der Summe

```
getAfeTableColumnFunction('.jsID_E_0_3', 'sum') über alle sichtbaren Zeilen des Widgets 'jsID_E_0_3'
```

```
setAfeWidgetNumValue('.jsID_C_0_0', Summe);
```

Hiermit schreiben wir **setAfeWidgetNumValue('.jsID_C_0_0', Summe);** die, in der Variable **Summe** gespeicherten Berechnung in das Kalkulations-Widget **'jsID_C_0_0'**

6.7 Einsatz von Variablen

Wie in Kapitel 3.1 bereits erwähnt wurde, ist der Einsatz von Variablen nur eingeschränkt möglich. Sollen diese in Berechnungen einfließen, müssen diese zunächst in einem Widget vom Typ Label mit Variablen ausgegeben und anschließend von dort mit der get Methode ausgelesen werden.

Widget-Einstellungen

Widget Typ Zuordnung & Datenwerte Widget-Verhalten Visuelles Hilfstexte Datenausgabeformat

- Eingabefeld
- Textareal
- Checkbox
- Einfaches Auswahlfeld (nur für die Zieltabelle)
- Lookup Auswahlfeld (für alle Tabellen)
- Mehrfachauswahl
- Label
- Label mit Variablen
- Platzhalter & Titel
- Business Case Link
- Datei Upload/Download

Label mit Variablen: Zum Darstellen von Text und für die Ausgabe von Variablen. Html ist erlaubt.

Interne Beschreibung

JavaScript Selektor ID
jsID_E_0_20

OK ABBRECHEN

Benötigt wird die JavaScript Selektor ID des Widgets

Widget-Einstellungen

Widget Typ Zuordnung & Datenwerte Widget-Verhalten Visuelles Hilfstexte Datenausgabeformat

Label Wert

<%offsetMay%>

Verstecke das Widget wenn der Wert einer verwendeten Variable leer ist

OK ABBRECHEN

Die Variable wird als Label Wert ausgegeben.

Widget-Einstellungen

Widget Typ Zuordnung & Datenwerte Widget-Verhalten Visuelles Hilfstexte Datenausgabeformat

Versteckt

Verstecke das Widget im Editbereich

Verstecke das Widget im Eingabebereich

Verstecke das Widget im Edit- und Eingabebereich für alle Anwender

OK ABBRECHEN

Optional kann das Widget auch als ‚versteckt‘ ausgeblendet werden

Beispiel:

```
setAfeTableWidgetNumValue(this,'jsID_E_0_7', calcValueNum+
getAfeTableWidgetNumValue(this,'jsID_E_0_20')
```

Mit **setAfe** wird hier in das Widget **'jsID_E_0_7'** der Wert der JavaScript Variablen **calcValueNum** geschrieben, addiert mit dem Wert der Apparo Variablen **<%offsetMay%>**, ausgegeben in Widget **'jsID_E_0_20'**

7 Einsatz von größeren JavaScript-Programmen

Mit zunehmender Komplexität empfiehlt es sich, das JavaScript in eine externe Datei auszulagern.

Mit der Variablen:

```
<%FILE_CONTENT(path+file)%>
```

können Sie den Inhalt der Datei wieder importieren.

Beispielaufruf der Datei myJsFunctions.txt:

```
<%FILE_CONTENT(D:\My Data\script\myJsFunctions.txt)%>
```

Nachfolgend wird die Variable mit dem Inhalt der Datei ersetzt:

```
myAlert(); // <- Beispielinhalt der Datei
```

8 Enter-Key für den Aufruf der JavaScript-Routine

Die Eingabetaste wird normalerweise verwendet, um einen Klick auf die OK-Taste zu simulieren.

Soll der Anwender aber mit der Eingabetaste die Berechnungen starten können, so muss die Funktion „ready“ erweitert werden:

```
$(document).ready(function){  
// Der Business Case wurde gestartet, das Calc-Widget ist auf 0 gesetzt  
setAfeWidgetNumValue('.jsID_C_0_0', 0);  
  
    disableFormSubmitOnEnter();  
}}
```

Mit **disableFormSubmitOnEnter();** wird die Eingabetaste umfunktioniert und ruft nur die JavaScript-Routine auf.

9 Beispiel eines Table Business Cases für Planung

In diesem Beispiel wird eine kleine Planungsanwendung entwickelt.

1. Die eingegebene Jahressumme wird auf die Jahresmonate verteilt, dabei werden die vergangenen Monate ignoriert (diese sind „eingefroren“)
2. Wenn der Anwender einen Monatsplanwert eingegeben hat, dann wird die Jahressumme automatisch aktualisiert
3. Für den Mai wird zusätzlich automatisch ein Wert dazugefügt aus einem „Label with variables“ Widget, das versteckt ist
4. Die Monatssummen über alle ausgewählten Produkte und die Jahressumme werden automatisch berechnet

Demo: Verteilung des Jahresbudgets auf 12 Monate

Product id
 Bags New York
 Gilbert
 Lueneburg
 Luxor

SUCHE FILTER ZURÜCKSETZEN

Produkt	Summe Jahr	Januar	Februar	März	April	Mai	Juni	Juli	August	September	Oktober	November	Dezember	Arbeitsschritt	Kommentar	Letzte An- zu
T-Shirt Vienna	3.000,00	0,00	0,00	200,00	300,00	2.000,00	300,00	120,00	200,00	-30,00	-30,00	-30,00	-30,00	offen		sales
Lueneburg	50.000,00	0,00	0,00	200,00	300,00	3.450,00	300,00	120,00	200,00	11.357,50	11.357,50	11.357,50	11.357,50	Bereit für Controlling		administ
Bags New York	-39.857,50	0,00	0,00	200,00	300,00	3.450,00	300,00	120,00	200,00	1.857,50	1.857,50	-50.000,00	1.857,50	offen		administ
New Yorker	20.000,00	0,00	0,00	400,00	500,00	500,00	5.005,00	500,00	500,00	3.148,75	3.148,75	3.148,75	3.148,75	Bereit für Controlling		administ
Nightblue	300,00	0,00	0,00	800,00	500,00	500,00	5.005,00	500,00	500,00	-1.876,25	-1.876,25	-1.876,25	-1.876,25	offen		administ
Gilbert	5.000,00	0,00	0,00	1.200,00	500,00	500,00	5.005,00	500,00	500,00	-801,25	-801,25	-801,25	-801,25	offen		administ
Luxor	8.000,00	0,00	0,00	800,00	500,00	500,00	5.005,00	500,00	500,00	48,75	48,75	48,75	48,75	Bereit für Controlling		administ
Madox	10.000,00	0,00	0,00	800,00	500,00	500,00	5.005,00	500,00	500,00	548,75	548,75	548,75	548,75	offen		administ
Summe		56.442,50	0,00	0,00	4.800,00	3.400,00	11.400,00	25.925,00	2.860,00	3.100,00	14.253,75	14.253,75	-37.603,75	14.253,75		

OK SCHLIESSEN

Diesem Beispiel Business Case finden Sie in der öffentlichen Demo

<https://demo.apparo.services>

Demonstration Apparo Fast Edit Business Case Liste administrator Demonstration Öffne Portal

+ Neu + Neu x Löschen Kopieren/Verschieben Import Export Filter

Business Case Ordner	Business Cases von Ordner Planning	Start ID	Name	Typ	Verbindungsname	Zieltabelle /-view	Geändert von	Geändert am
Demonstration	Planning year		Planning year with auto-distributing month values + sums	Table	SAMPLES	SAMPLE_FORECAST5	administrator	23.08.21 11:29
	SAMPL APP SALES		SAMPLES - Sales	Set			Administrator	28.11.17 14:49
	SAMPL APP SALES MAN		SAMPLES - Sales manager	Table	SAMPLES	SAMPLE_SALES	Administrator	28.11.17 14:48
	SAMPL MASTER PLAN DETAILS		SAMPLES - product details	Single	SAMPLES	SAMPLE_PRODUCT	Administrator	28.11.17 14:48
	SAMPL PLAN SALES PLAN		SAMPLES - sales planning	Table	SAMPLES	SAMPLE_SALES_PLANNING	Administrator	28.11.17 14:48

Beschreibungen einblenden

Hinweis

Im Menu 'Ausbildung' des Apparo Designers finden Sie die Making-of-Videos zu diesem Business Case.

Das Skript:

```
// sum of a product was changed:
$(document).on('change', '.jsID_E_0_1', function(){

// get current value of the sum widget:
var myValue = getAfeTableWidgetNumValue(this, '.jsID_E_0_1');

// Date calculations...
var currentDate = new Date();
var currentMonth = currentDate.getMonth() +2;
var months = 13 - currentMonth;

//calculate sum of values in the past
var sumOfPast = 0;
if (currentMonth >= 2) { sumOfPast = getAfeTableWidgetNumValue(this, '.jsID_E_0_3'); };
if (currentMonth >= 3) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_4'); };
if (currentMonth >= 4) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_5'); };
if (currentMonth >= 5) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_6'); };
if (currentMonth >= 6) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_7'); };
if (currentMonth >= 7) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_8'); };
if (currentMonth >= 8) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_9'); };
if (currentMonth >= 9) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_10'); };
if (currentMonth >= 10) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_11'); };
if (currentMonth >= 11) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_12'); };
if (currentMonth >= 12) { sumOfPast = sumOfPast + getAfeTableWidgetNumValue(this, '.jsID_E_0_13'); };

// change values of future months and current only
// toFixed(2) rounds the value to 2 decimal places, but the result is string. The '+' in the begin convert this string into English number
var calcValueNum = + ((myValue-sumOfPast) / months).toFixed(2));

if (currentMonth == 1) { setAfeTableWidgetNumValue(this, '.jsID_E_0_3', calcValueNum); };
if (currentMonth <= 2) { setAfeTableWidgetNumValue(this, '.jsID_E_0_4', calcValueNum); };
if (currentMonth <= 3) { setAfeTableWidgetNumValue(this, '.jsID_E_0_5', calcValueNum); };
if (currentMonth <= 4) { setAfeTableWidgetNumValue(this, '.jsID_E_0_6', calcValueNum); };

// for May additional the calculated value of E_0_17 must be added.
// this is an example how to use calculated values from the database, e.g. SQL. The value is stored in a hidden widget of type "Label with variables"
if (currentMonth <= 5) { setAfeTableWidgetNumValue(this, '.jsID_E_0_7', calcValueNum+ getAfeTableWidgetNumValue(this, '.jsID_E_0_17') ); };

if (currentMonth <= 6) { setAfeTableWidgetNumValue(this, '.jsID_E_0_8', calcValueNum); };
if (currentMonth <= 7) { setAfeTableWidgetNumValue(this, '.jsID_E_0_9', calcValueNum); };
if (currentMonth <= 8) { setAfeTableWidgetNumValue(this, '.jsID_E_0_10', calcValueNum); };
if (currentMonth <= 9) { setAfeTableWidgetNumValue(this, '.jsID_E_0_11', calcValueNum); };
if (currentMonth <= 10) { setAfeTableWidgetNumValue(this, '.jsID_E_0_12', calcValueNum); };
if (currentMonth <= 11) { setAfeTableWidgetNumValue(this, '.jsID_E_0_13', calcValueNum); };
if (currentMonth <= 12) { setAfeTableWidgetNumValue(this, '.jsID_E_0_14', calcValueNum); };

// because the particular values may be rounded, recalculate the SUM in order to reflect the sum of rounded values
calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_3', function(){
// january value was changed
calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_4', function(){
// february value was changed
calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_5', function(){
calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_6', function(){
calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_7', function(){
calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_8', function(){
calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_9', function(){
```

```

    calculateYearSum(this);
  })

$(document).on('change', '.jsID_E_0_10', function(){
  calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_11', function(){
  calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_12', function(){
  calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_13', function(){
  calculateYearSum(this);
})

$(document).on('change', '.jsID_E_0_14', function(){
  calculateYearSum(this);
})

function calculateYearSum(elem) {
  // make a sum of all months of the current product
  var yearSum = getAfeTableWidgetNumValue(elem, '.jsID_E_0_3');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_4');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_5');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_6');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_7');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_8');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_9');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_10');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_11');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_12');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_13');
  yearSum = yearSum + getAfeTableWidgetNumValue(elem, '.jsID_E_0_14');

  // recalculate all month sums of all products
  calculateColumnSums();

  // set sum of year of current product
  setAfeTableWidgetNumValue(elem, '.jsID_E_0_1', yearSum);

}

function calculateColumnSums() {

  // recalc all month sums

  var m;
  m = getAfeTableColumnFunction('.jsID_E_0_3', 'sum');
  setAfeWidgetNumValue('.jsID_C_0_3', m);

  m = getAfeTableColumnFunction('.jsID_E_0_4', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_4', m);

  m = getAfeTableColumnFunction('.jsID_E_0_5', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_5', m);

  m = getAfeTableColumnFunction('.jsID_E_0_6', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_6', m);

  m = getAfeTableColumnFunction('.jsID_E_0_7', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_7', m);

  m = getAfeTableColumnFunction('.jsID_E_0_8', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_8', m);

  m = getAfeTableColumnFunction('.jsID_E_0_9', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_9', m);

  m = getAfeTableColumnFunction('.jsID_E_0_10', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_10', m);

  m = getAfeTableColumnFunction('.jsID_E_0_11', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_11', m);

  m = getAfeTableColumnFunction('.jsID_E_0_12', 'sum');
  setAfeWidgetNumValue( '.jsID_C_0_12', m);
}

```

```

m = getAfeTableColumnFunction('.jsID_E_0_13', 'sum');
setAfeWidgetNumValue( '.jsID_C_0_13', m);

m = getAfeTableColumnFunction('.jsID_E_0_14', 'sum');
setAfeWidgetNumValue( '.jsID_C_0_14', m);

// calc total sum and display it, value is the sum of all products
setAfeWidgetNumValue( '.jsID_C_0_1',
  getAfeTableColumnFunction('.jsID_E_0_3', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_4', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_5', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_6', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_7', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_8', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_9', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_10', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_11', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_12', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_13', 'sum') +
  getAfeTableColumnFunction('.jsID_E_0_14', 'sum') );
}

$(document).ready(function(){
// Business Case was started, this function will be called automatically, the calc widget are updated

// pressing enter key means new event and not making submit
disableFormSubmitOnEnter();

// calc month sums:
calculateColumnSums();

})

function onAfeFormReload() {
  $(document).ready(function(){
// Business Case after submit (e.g. pressing OK button) is calling this function automatically

// enter key means new event
disableFormSubmitOnEnter();

// calc month sums:
calculateColumnSums();
  })
}

$(document).on('focus', '.jsID_E_0_1', function(){

// the user has clicked into the sum widget. Now this function is called automatically.
// This is helpful if you want to make calculations directly after user clicked into a widget

// ... place for activities

})

```