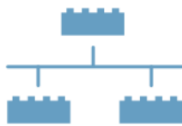


Benutzerhandbuch WebForm-Designer

Next Generation Business Process Management



Geschäftsprozesse flexibel
automatisieren



Anwendungen prozessorientiert
orchestrieren



Human Workflows ohne
Programmierung realisieren



Prozess-Steuerung nahtlos
integrieren



Der AristaFlow WebForm-Designer



Vorbemerkungen

Mit dem AristaFlow WebForm-Designer können professionelle Formulare erstellt werden, die mehr Flexibilität beim Layout und eine höhere Funktionalität bieten.

Die vom WebForm-Designer erzeugten Formulare sind HTML-basiert, daher betriebssystemunabhängig und auf allen gängigen Browsern einsetzbar.

Die wesentlichen Funktionen dieses Editors werden im Folgenden meist anhand einfacher Prozessbeispiele erläutert. Es bietet sich an, diese Beispiele mit Hilfe des AristaFlow *Process Template Editors* unter Verwendung der *WebForm*-Aktivitätenvorlagen nachzuvollziehen.

Die Beispiele basieren meist auf den Tabellen der Übungsdatenbank *LegoTrailer AG*, die im Anhang nachgeschlagen werden kann. Die SQL-Skripte zur automatisierten Erstellung und Befüllung der SQL-Tabellen finden Sie im internen Bereich des AristaFlow-Forums.

Vorkenntnisse

Die folgenden Kenntnisse werden in dieser Beschreibung als vorhanden vorausgesetzt:

- Modellierung von Prozessen mit dem AristaFlow *Process Template Editor*
- Bedienung des AristaFlow *Process Template Editors*
- Umgang mit Aktivitätenvorlagen

Diese Kenntnisse werden auf unsere Online Dokumentation der BPM Suite vermittelt. Die Tutorials und mehr finden Sie unter: <https://pubweb.aristaflow.com/temp/doc/master/>

1	STARTEN DES WEBFORM-DESIGNERS	7
2	FORMULARMODELLIERUNG	10
2.1	Anwendungsoberfläche	10
3	BEISPIELPROZESS AUFTRAGSBEARBEITUNG	11
4	ENTWURF DER FORMULARE.....	14
4.1	Elemente einfügen und konfigurieren.....	14
4.2	Control-Eigenschaften	17
4.3	Eingabe- und Ausgabeparameter zuweisen	18
4.4	Mehr-Blatt-Formulare.....	20
4.5	Zusammenspiel von Controls und Events (Ereignissen)	21
5	VERKNÜPFUNG VON CONTROLS MIT DATENBANKINHALTEN	22
5.1	Einrichten von Datenbankinhalten	22
5.2	Formulierung und Hinterlegung von Datenbankabfragen	24
5.2.1	Allgemeines	24
5.2.2	Arten von Datenbankabfragen.....	25
5.2.3	Abfragen für Beispiel-Formulare	26
5.3	Zuordnung von Datenbankabfragen zu Controls.....	30
5.4	Ereignisgesteuerte DB-Anfragen-Ausführung zur Aktualisierung von Controls	32
5.5	Erstellen von Tabellen (Subtables)	35
5.5.1	Realisierung interner Subtabellen.....	35
5.5.2	Realisierung von Output-Subtabellen	41
6	IMPLEMENTIERUNG VON FORMULARFUNKTIONEN UND ZUSÄTZLICHER LOGIK.....	46
6.1	Events und Aktionen für ein Control definieren.....	46
6.1	Erstellung von JavaScript-Funktionen	47
6.2	Verknüpfung von JavaScript-Funktionen mit Events und Controls	50
6.3	Handhabung der „Formular-Ereignisse“ onload, submit und onsuspend	52
6.4	Arbeiten mit Subformularen.....	54
7	UMGANG MIT PARAMETERN VOM TYP <i>USERDEFINED SUBTABLE</i>	57
7.1	Allgemeines zu erweiterten AristaFlow-Datentypen.....	57
7.2	Erzeugung von Parameter-Objekten vom Typ <i>USERDEFINED subtable</i>	57

7.3	USERDEFINED subtable als Input-Parametertyp einer WebForm-Aktivität	59
7.4	Weiterverarbeitung von USERDEFINED subtable Ausgabeparametern	62
8	ANWENDUNGSBEISPIELE	62
8.1	Validieren von Werten von Controls	62
8.2	Ein- und Ausblenden von Controls	64
8.3	Zeilenweise Verarbeitung von Subtabellen	65
8.4	Realisierung von Auswahl-Tabellen	66
8.4.1	Realisierung von „Multi-Selection-Auswahltabellen“	66
8.4.2	Realisierung von „Single-Selection“-Auswahltabellen	68
8.5	Komplettbeispiel Auftragserfassung und –speicherung in Datenbank	70
8.5.1	Schritt 1: Auftrag erfassen.....	71
8.5.2	Schritt 2: Auftrag -> DB mit der Aktivität Scripting	77
9	TROUBLE SHOOTING	80
6.1	Testen der Datenbankverbindung	80
9.1	Erkennung von SQL-Ausführungsfehlern	80
9.2	JavaScript-Debugging mit Webentwickler Tools (von Mozilla Firefox)	82
10	SCHLUSSBEMERKUNGEN	84
11	ANHANG: BEISPIELTABELLEN DER LEGOTRAILERDB	85

Abbildungsverzeichnis

Abbildung 1: Aktivitätenvorlage des WebForm-Designers.....	7
Abbildung 2: Aktivitätenvorlage auf einen Prozessschritt ziehen.....	8
Abbildung 3: Starten des WebForm-Designers.....	9
Abbildung 4: WebForm-Designer mit Controls, Formular und Properties.....	10
Abbildung 5: Beispielprozess.....	11
Abbildung 6: Stammdaten abfragen: Kunden – mit Dropdown-Liste für Kunden auswählen realisiert	11
Abbildung 7: Stammdaten abfragen - mit Dropdown-Liste für Kunden auswählen realisiert.....	12
Abbildung 8: Auftragsanzeige als Subtabelle realisiert.....	12
Abbildung 9: Stammdaten abfragen – mit Auswahltabelle für Kunde auswählen realisiert (1).....	13
Abbildung 10: Stammdaten abfragen - mit Auswahltabelle für Kunden auswählen realisiert (2).....	13
Abbildung 11: Subformulare.....	14
Abbildung 12: Einfügen und Konfigurieren von Controls (1).....	15
Abbildung 13: Einfügen und Konfigurieren von Controls (2).....	16
Abbildung 14: Properties eines Controls.....	18
Abbildung 15: Verknüpfen eines Parameters mit einem Control (Alternative 1).....	19
Abbildung 16: Verknüpfen eines Parameters mit einem Control (Alternative 2).....	19
Abbildung 17: Resultierende Verknüpfung des Ausgabeparameters mit dem gewählten Control.....	19
Abbildung 18: Benennung der Formular-Reiter.....	20
Abbildung 19: Einfügen eines Formularblatt.....	20
Abbildung 20: Einrichten einer neuen Datenbankverbindung.....	22
Abbildung 21: Konfigurationsdialog.....	22
Abbildung 22: Konfigurationsbeispiel.....	23
Abbildung 23: Erstellen einer neuen Datenbankabfrage.....	24
Abbildung 24: Formular Stammdaten abfragen.....	26
Abbildung 25: SQL-Anweisung Erstellen Kundenliste.....	27
Abbildung 26: SQL-Anweisung Ausgabe ausgewählter Kunde.....	27
Abbildung 27: SQL-Anweisung Erstellen Auftragsliste.....	28
Abbildung 28: Für Formular Stammdaten abfragen definierte Abfragen.....	28
Abbildung 29: Formular Auftrag anzeigen.....	29
Abbildung 30: Formular Auftrag anzeigen.....	30
Abbildung 31: Datenbankabfrage Bestimmen Kundenname für Formular Auftrag anzeigen.....	30
Abbildung 32: Datenbankabfrage Bestimmen Auftragspositionen für Formular Auftrag anzeigen.....	30
Abbildung 33: Properties des Controls kunde_combobox.....	31
Abbildung 34: Properties des Controls kdnme_textbox.....	31
Abbildung 35: Properties des Controls kdstadt_textbox.....	31
Abbildung 36: Properties des Controls kdbonitaet_textbox.....	31
Abbildung 37: Properties des Controls auftrnr_combobox.....	32
Abbildung 38: Properties des Controls kdnme_textbox.....	32
Abbildung 39: Properties des Controls subtable.....	32
Abbildung 40: Laufzeitansicht des Formulars Stammdaten abfragen.....	33
Abbildung 41: Festlegen eines EventHandlers (1).....	33
Abbildung 42: Festlegen eines EventHandlers (2).....	34
Abbildung 43: Festlegen eines EventHandlers (3).....	34
Abbildung 44: Laufzeitansicht von Stammdaten abfragen nach Einfügen des EventHandlers.....	34
Abbildung 45: Properties der SubtableView subtable mit Columns-Property.....	35
Abbildung 46: Column-Editor.....	36
Abbildung 47: Properties des Spaltentyps Text Column.....	37
Abbildung 48: Tabelle mit auswählbaren Zeilen mittels Check Column.....	38
Abbildung 49: Data-Properties der Spalten TeileNr, Anzahl, Preis und Farbe.....	38
Abbildung 50: Beispiel - Mögliche Ausgabe der Formularaktivität Auftrag anzeigen.....	39
Abbildung 51: Festlegung von readonly- und addremove-Eigenschaften für Subtables.....	40
Abbildung 52: Subtable mit addremove = true Eigenschaft.....	40
Abbildung 53: Erweiterte Auftragsbearbeitungsprozess.....	41
Abbildung 54: Erzeugen der Subtable-Datenstruktur (1).....	41
Abbildung 55: Erzeugen der Subtable-Datenstruktur (2).....	42
Abbildung 56: Erzeugen der Subtable-Datenstruktur (3).....	42
Abbildung 57: Erzeugen der Subtable-Datenstruktur (4).....	42
Abbildung 58: Verknüpfung der Formular-Tabelle mit der Subtable-Datenstruktur (1).....	43
Abbildung 59: Verknüpfung der Formular-Tabelle mit der Subtable-Datenstruktur (2).....	43
Abbildung 60: Verknüpfung der Formular-Tabelle mit der Subtable-Datenstruktur (3).....	44

Abbildung 61: Verknüpfung der Tabelle mit einem Ausgabeparameter (1)	44
Abbildung 62: Verknüpfung der Tabelle mit einem Ausgabeparameter (2)	45
Abbildung 63: Formular für Stammdaten abfragen mit Namen des Controls	46
Abbildung 64: Eventhandler für Control kunde_combobox.....	47
Abbildung 65: Zentrales JavaScript-Fenster	48
Abbildung 66: WebForm API Dokumentation (1)	49
Abbildung 67: WebForm API Dokumentation (2)	49
Abbildung 68: Formular Auftrag anzeigen mit zusätzlichem Summenfeld.....	50
Abbildung 69: Hinterlegung von JavaScript-Funktionen im JavaScript-Fenster	51
Abbildung 70: Aufruf der JavaScript-Funktion berechnePreis() im EventHandler des Controls Preis..	52
Abbildung 71: Formular mit berechnetem Wert	53
Abbildung 72: Realisierung einer Initialisierungsfunktion	53
Abbildung 73: Aufruf der Funktion init() im EventHandler des Formulars	54
Abbildung 74: Schritt Erfassen Auftrag- Subformular Anlegen Auftrag	54
Abbildung 75: Schritt Erfassen Auftrag –Subformular Erfassen Auftragspositionen	55
Abbildung 76: Erfassen Auftrag- erstes Subformular aktiv	55
Abbildung 77: Erfassen Auftrag – zweites Subformular aktiv	55
Abbildung 78: Erfassen Auftrag – zweites Subformular aktiv	56
Abbildung 79: Auftrag anlegen button freischalten oder sperren	56
Abbildung 80: Umschalten von einem Subformular auf ein anderes	56
Abbildung 81: Erzeugung eines UDT:subtable-Objektes per Scripting-Aktivität.....	58
Abbildung 82: SubtableView und subtable-Datenstruktur des Schrittes Anzeigen Tabelle	58
Abbildung 83: Scripting-Aktivität des Schrittes Erzeugen Tabelle	58
Abbildung 84: Script zum Erzeugen und Befüllen des Subtable-Ausgabeobjekts.....	59
Abbildung 85: Resultierende Tabelle	59
Abbildung 86: Formular zur Berechnung einer Entfernung.....	62
Abbildung 87: JavaScript: function checkIntegerDifference() – Version 1	63
Abbildung 88: JavaScript: Formular mit gültigen Eingaben	63
Abbildung 89: Formular mit ungültigen Eingaben	63
Abbildung 90: Formular mit falscher nicht-numerischer Eingabe.....	64
Abbildung 91: Formular Stammdatenabfrage mit zusätzlicher Groupbox	64
Abbildung 92: Init() Funktion.....	64
Abbildung 93: einAusblenden() Funktion	65
Abbildung 94: Auftragserfassung mit Gesamtwert-Berechnung	65
Abbildung 95: JavaScript-Funktion: Ermitteln Spaltensumme	65
Abbildung 96: Auswahl-Tabelle	66
Abbildung 97: Anlegen einer Checkbox-Spalte.....	66
Abbildung 98: Auswahlformular nach Start	67
Abbildung 99: Auswahlformular nach erfolgter Auswahl	67
Abbildung 100: JavaScript-Funktion zum Löschen der abgewählten Zeilen	68
Abbildung 101: Beispiel für „Single-Selection“ -Auswahltablelle.....	68
Abbildung 102: Hinterlegen eines onclick-Ereignisses mit Selbst-Referenz.....	69
Abbildung 103: JavaScript-Funktion zur Realisierung der Single-Selection Auswahltablelle.....	70
Abbildung 104: Prozess „Auftragerfassung und -speicherung in Datenbank“	70
Abbildung 105: Subformular Anlegen Auftrag des Prozessschrittes Auftrag erfassen	71
Abbildung 106: Subformular Erfassen Auftragspositionen des Prozessschrittes Auftrag erfassen.....	71
Abbildung 107: Webformular Anlegen Auftrag	72
Abbildung 108: Datenstruktur der Subtable Kunde_SubtableView	72
Abbildung 109: SQL Anfrage Erstellen Kundenliste für Kunde_SubtableView	73
Abbildung 110: Webformular Erfassen Auftragspositionen.....	73
Abbildung 111: Datenstrukturen von TeileKatalog_SubtableView und AuftragsPos_SubtableView	74
Abbildung 112: SQL Anfrage Ausgabe ausgewählter Kunde für Kunden Textboxen	74
Abbildung 113: SQL Anfrage Teilekatalog für TeileKatalog_SubtableView.....	74
Abbildung 114: JavaScript-Funktionen des Formulars Erfassen Auftrag (1)	75
Abbildung 115: JavaScript-Funktionen des Formulars Erfassen Auftrag (2)	76
Abbildung 116: Konfiguration für Datenbankverbindung.....	77
Abbildung 117: Script für den Eintrag in die Datenbank (1)	78
Abbildung 118: Script für den Eintrag in die Datenbank (2)	79
Abbildung 119: Testen der Datenbankverbindung mit der executeStatements-Aktivität.....	80
Abbildung 120: AristaFlow-Logger View – Liste der Nachrichten	81
Abbildung 121: AristaFlow-Logger-View Detaildarstellung	82
Abbildung 122: Webformular mit JavaScript Funktion	82

Abbildung 123: Laufzeit-Fehlermeldung.....	83
Abbildung 124: Eigenschaften-Fenster mit URL	84
Abbildung 125: Debugger Fenster von Firefox.....	84

Tabellenverzeichnis

Tabelle 1: Graphische Elemente <i>Controls</i> des WebForm-Designers	14
Tabelle 2: Änderbare Eigenschaften des Controls.....	17
Tabelle 3: Typen von Spalten für Control <i>Subtable</i>	36
Tabelle 4: Typen von Ereignissen (Events).....	47
Tabelle 5: Erläuterungen zum Prozess <i>Auftragserfassung und -speicherung in Datenbank</i>	72
Tabelle 6: Im Formular <i>Auftrag erfassen</i> verwendete JavaScript-Funktionen und Ereignisse	73
Tabelle 7: <i>kunden</i> -Tabelle	85
Tabelle 8: <i>auftraege</i> -Tabelle.....	85
Tabelle 9: <i>auftragspos</i> -Tabelle	85
Tabelle 10: <i>preisliste</i> -Tabelle.....	85
Tabelle 11: <i>teiletypen</i> -Tabelle	85

1 Starten des WebForm-Designers

Der WebForm-Designer wird nicht als separate Anwendung gestartet, sondern ist als eine Aktivitätenvorlage realisiert. Das Erstellen von Formularen mit dem WebForm-Designer findet während der Prozessmodellierung mit dem AristaFlow *Process Template Editor* in Form einer Konfiguration der Aktivitätenvorlage für den Prozessschritt statt, dem sie zugeordnet wurde.

Die Aktivitätenvorlage für ein Webformular steht im *Activity Repository Browser* unter *de.aristaflow.form.WebForm* → *WebForm* → zur Verfügung.

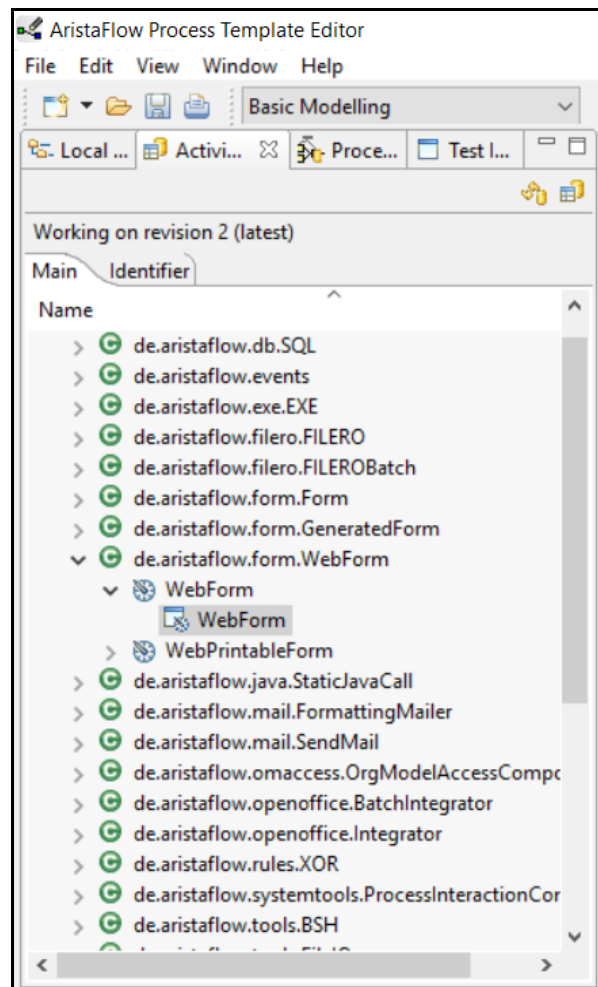


Abbildung 1: Aktivitätenvorlage des WebForm-Designers

Um den WebForm-Designer verwenden zu können, muss eine WebForm-Aktivitätenvorlage auf einen gewünschten Prozessschritt per Drag'n'Drop gezogen werden (siehe [Abbildung 2](#)).

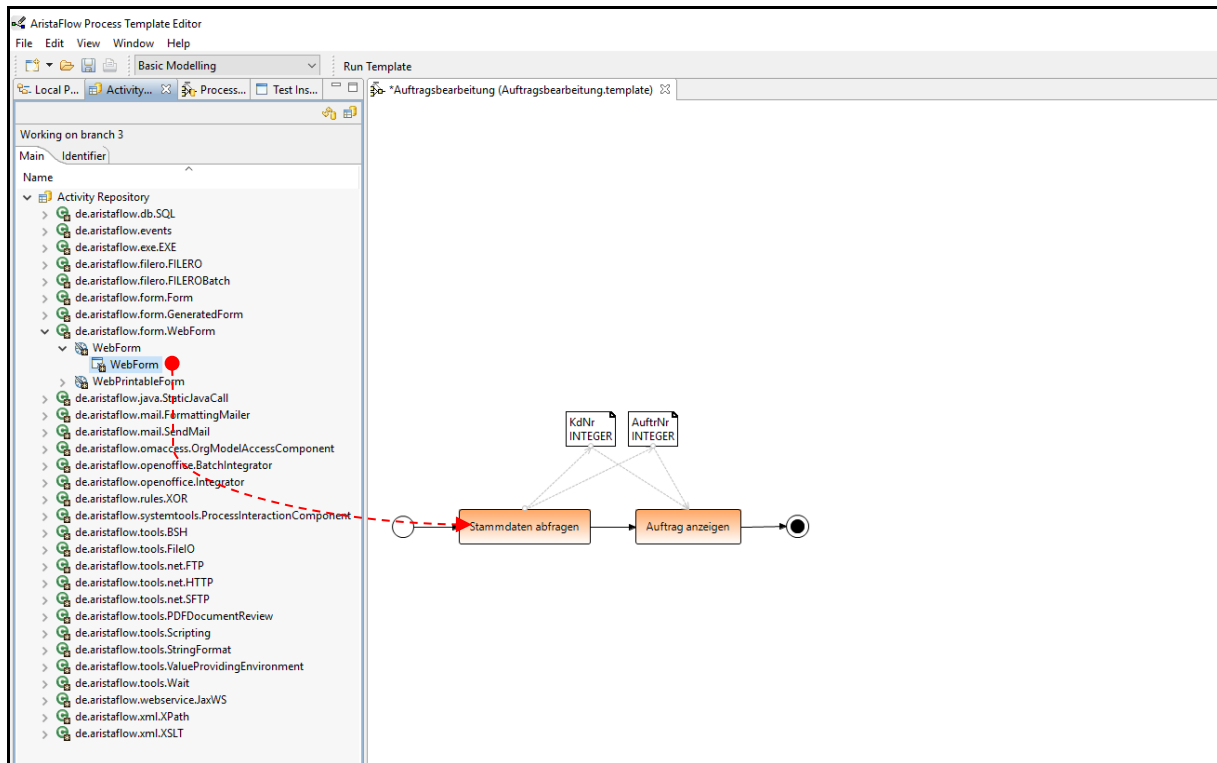


Abbildung 2: Aktivitätenvorlage auf einen Prozessschritt ziehen

Nach Zuordnung der WebForm-Aktivitätenvorlage zum Prozessschritt, öffnet sich ein Konfigurationsdialog. Mit *Finish* wird der Konfigurationsdialog beendet. Um den WebForm-Designer öffnen zu können, muss der Prozessschritt selektiert und im Rechtsklickmenü auf *Open in Form Designer...* geklickt werden. (siehe [Abbildung 3](#))

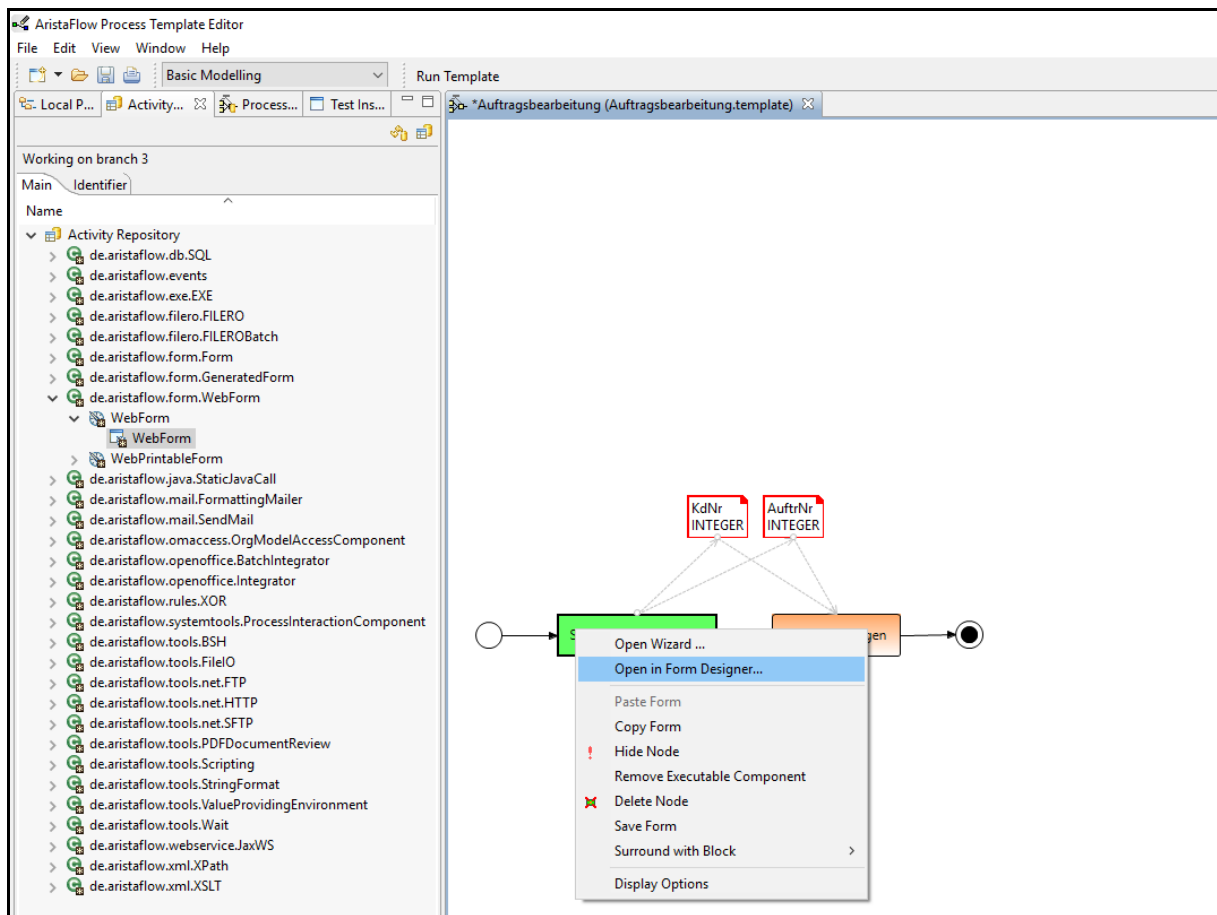


Abbildung 3: Starten des WebForm-Designers

Durch Bestätigung wird der WebForm-Designer gestartet und in einem Entwurfswindow, in dem man ein neues Formular für diesen Prozessschritt entwickeln oder ein mit diesem Schritt evtl. assoziiertes Formular verändern kann.

2 Formulardesign

2.1 Anwendungsoberfläche

Die Oberfläche des Editors (*Abbildung 4*) weist drei Bereiche auf und zwar

- eine Leiste, der zur Verfügung stehenden Formularelemente (*Controls*)
- dem eigentlichen Formularbereich
- dem Konfigurationsbereich mit den drei Registerreibern
 - *Properties* (Eigenschaften eines Controls)
 - *Data Sets* (Datenbankquellen)
 - *Process Data* (Ein- und Ausgabeparameter dieses Prozessschrittes)

Die Eigenschaften der einzelnen *Controls* und deren Konfigurationsmöglichkeiten werden im Folgenden näher erläutert.

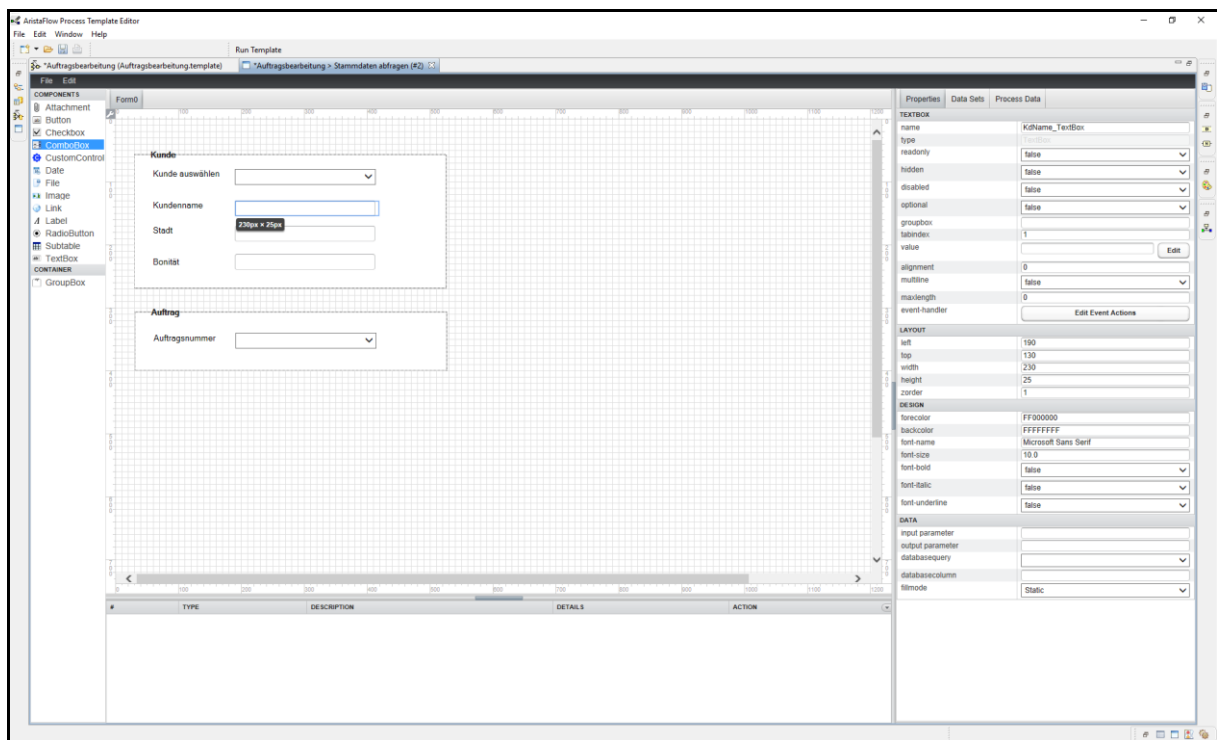


Abbildung 4: WebForm-Designer mit Controls, Formular und Properties

3 Beispielprozess Auftragsbearbeitung

Im Folgenden wird der in *Abbildung 5* dargestellte einfache Beispielprozess verwendet, um die Funktionen des WebForm-Designers zu erläutern. Wie man sieht, hat die Aktivität *Stammdaten abfragen*, die beiden Ausgabeparameter *KdNr* und *AuftrNr*, die gleichzeitig Eingabeparameter für die nachfolgende Aktivität *Auftrag anzeigen* sind.

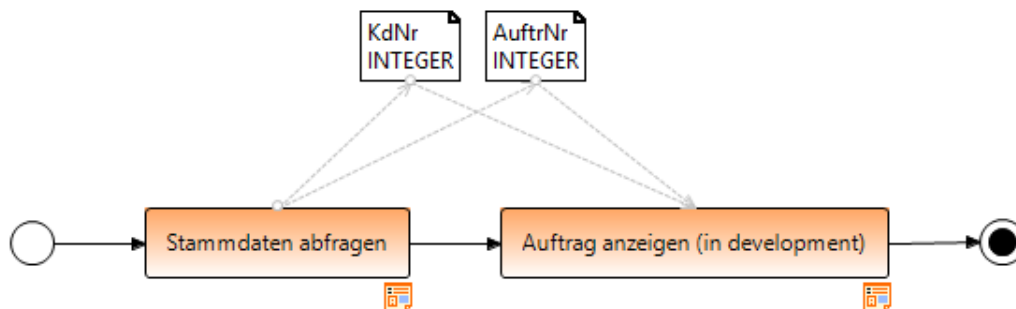


Abbildung 5: Beispielprozess

Mit diesem Prozess soll die Auswahl und Anzeige eines Auftrags realisiert werden. Hierfür werden zwei Formulare erstellt, und zwar für die *Stammdaten abfragen* (*Abbildung 6* und *Abbildung 7*, wo der Kunde und der gewünschte Auftrag (d.h. dessen *Auftragsnummer*) ausgewählt wird und die *Auftragsanzeige* (*Abbildung 8*) welche die einzelnen Auftragspositionen und deren Gesamtwert anzeigt.

Die beiden Formulare sollen zur Laufzeit, das in *Abbildung 6*, *Abbildung 7* und *Abbildung 8* dargestellte Aussehen haben. Nach Klick auf den Aufklapp-Button (*Abbildung 6 a*) wird eine Liste von Kundennamen angezeigt (*Abbildung 6 b*), aus welcher der Anwender einen auswählen kann.

The image shows two versions of a web form. The top part of the form is labeled 'Kunde' and contains a dropdown menu 'Kunde auswählen' with a downward arrow, and three text input fields for 'Kundenname', 'Stadt', and 'Bonität'. The bottom part is labeled 'Auftrag' and contains a dropdown menu 'Auftragsnummer' with a downward arrow. In screenshot a), the dropdowns are closed. In screenshot b), the 'Kunde auswählen' dropdown is open, showing a list of customer names: Aberer, Babel, Becker, Beyer AG, Breu, Dehling, Dreher, Fischer GmbH, Fobke GmbH, Graf KG, Hermann KG, Kahlert, Ketzer, Koch GmbH & Co. KG, Lummer, Manger, Merz, Moehne, Schmidt, Voigt & Co., Walter, Womser, Zander, Zimmer, Zwack.

Abbildung 6: Stammdaten abfragen: Kunden – mit Dropdown-Liste für Kunden auswählen realisiert

Nach Auswahl eines Kunden mittels der Dropdown-Liste werden *Kundenname*, *Stadt* und *Bonität* in den zugeordneten *TextBoxen* angezeigt und unter *Aufträge des Kunden*, dessen *Auftragsnummern* in

der Auswahlliste angezeigt. Aus dieser kann der Anwender dem in nächsten Prozessschritt anzuzeigenden Auftrag auswählen.

Abbildung 7: Stammdaten abfragen - mit Dropdown-Liste für Kunden auswählen realisiert

TeileNr	Anzahl	Farbe	Preis	+
				-
Number of new Rows:				+

Abbildung 8: Auftragsanzeige als Subtabelle realisiert

Alternativ zur oben dargestellten Auswahl des Kunden mittels Dropdown-Liste hätten wir uns auch für eine Auswahl via Subtabelle entscheiden können, wie in [Abbildung 9](#) und [Abbildung 10](#) dargestellt. In [Abbildung 9](#) ist die Entwurfsansicht dargestellt, während [Abbildung 10](#) die Laufzeitansicht zeigt.

Damit in der Subtabelle jeweils immer nur max. eine Zeile ausgewählt sein kann und damit die *TextBoxen* bei Auswahl einer Zeile mit den korrespondierenden Werten gefüllt werden, muss man geeignete JavaScript Funktionen implementieren. Hierauf werden wir in Kapitel 6 [Implementierung von Formularfunktionen und zusätzlicher Logik](#) eingehen.

Kunde

Kundenliste

CHECK COLUMN	NUMMER	NAME	STADT

Auftrag

Auftragsnummer

Abbildung 9: Stammdaten abfragen – mit Auswahltabelle für Kunde auswählen realisiert (1)

Kunde

Kundenliste

	Nummer	Name	Stadt	+
<input type="checkbox"/>	100	Walter	Siegen	-
<input checked="" type="checkbox"/>	105	Aberer	Stuttgart	-
<input type="checkbox"/>	112	Zander	Ulm	-
Number of new Rows:				+

Auftrag

Auftragsnummer

Abbildung 10: Stammdaten abfragen - mit Auswahltabelle für Kunden auswählen realisiert (2)

4 Entwurf der Formulare

4.1 Elemente einfügen und konfigurieren

Für die Gestaltung von Formularen stehen die in [Tabelle 1](#) dargestellten graphischen Elemente *Controls* zur Verfügung. Außerdem besteht noch die Möglichkeit *Subformulare* zu verwenden ([Abbildung 11](#)).

Attachment	Ein Attachment bietet die Möglichkeit, eine Datei hochzuladen
Button	Ein Button kann eine festgelegte JavaScript-Funktion ausführen
CheckBox	Standard-Checkbutton
ComboBox	Dropdown-Menü, entweder mit vordefinierten Werten innerhalb des Formulars oder automatisch befüllbar mit Ergebnissen einer SQL-Abfrage
CustomControl	Für spätere Erweiterung reserviertes Element, wird in diesem Dokument nicht behandelt
Date	Datum (Tag, Monat und Jahr sowie Uhrzeit, auch in Kombination). Das Format lässt sich in den Eigenschaften einstellen. Kann bei Aktivierung eines Kalenders angezeigt werden, mit deren Hilfe sich das Element komfortabel befüllen lässt.
File	Mit File kann ein Benutzer eine verknüpfte Datei herunterladen
Image	Bild im Formular anzeigen, z.B. für ein Logo
Label	Text
Link	URL
RadioButton	Radioknöpfe, die zu einer Gruppe zusammengefasst werden können (indem man ihnen denselben internen Namen gibt), um eine Auswahl über mehrere Möglichkeiten zu geben.
Subtable	Tabelle (siehe 5.5 Erstellen von Tabellen (Subtables))
TextBox	Simple Texteingabe. Das akzeptierte Format hängt von den verknüpften Prozessein- und Ausgabeparametern ab. Ist in <i>AristaFlow Process Template Editor</i> ein Datenelement vom Typ <i>Date</i> zu befüllen, so akzeptiert die <i>TextBox</i> auch nur ein solches im richtigen Format (dd.mm.yy). Unterstützt sowohl einen Einzeilen- als auch Mehrzeilenmodus.
GroupBox	Mit einer GroupBox können mehrere Elemente logisch zusammengefasst werden, etwa um sie mit einem Rahmen zu versehen oder um sie zur Laufzeit gemeinsam ein- oder auszublenden.

Tabelle 1: Graphische Elemente *Controls* des WebForm-Designers

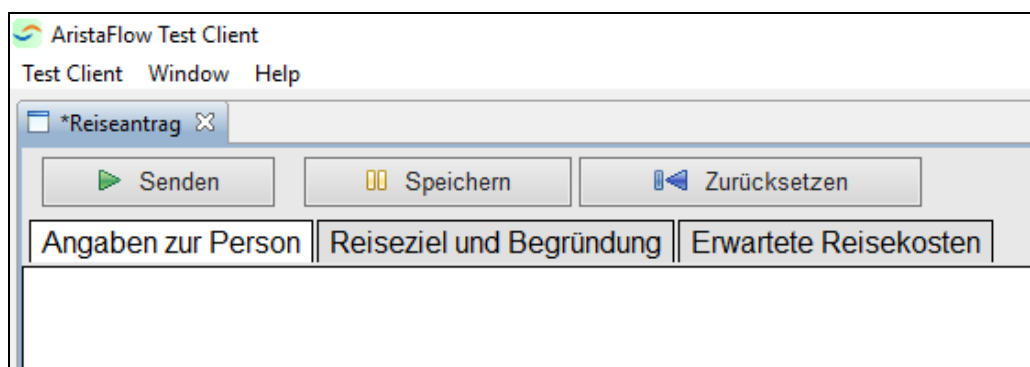


Abbildung 11: Subformulare

Die graphischen Elemente werden mittels gedrückter linker Maustaste vom Auswahlfenster (linkes Teilfenster in [Abbildung 12](#)) auf die Zeichenebene (mittleres Teilfenster) per Drag´n´Drop gezogen und durch Einträge in der *Properties*-Ansicht (rechtes Teilfenster) konfiguriert.

Zur Erzeugung des Formulars *Stammdaten abfragen* könnte man etwa wie folgt vorgehen.

1. Man fügt zwei übereinander stehende *GroupBoxen* (für *Kunde* und *Aufträge*) in das Formular ein.
2. Mit einem Klick auf die *Controls* werden bei Bedarf ihre Eigenschaften im Registerreiter *Properties* verändert, etwa mit einer Beschriftung versehen, wie in [Abbildung 12](#) für die *GroupBox Kunde* illustriert.
3. Anschließend fügt man in *GroupBox Kunde* und in *GroupBox Auftrag* jeweils einen *DropDown-Box* und in *GroupBox Kunde* zusätzlich drei *Textboxen* ein, wie in [Abbildung 13](#) dargestellt.
4. Sollen die *Controls* mit Ein-/Ausgabeparametern verknüpft werden oder auf diese in JavaScript-Routinen Bezug genommen werden, so bietet es sich an, anstelle der vom System vergebenen Bezeichnern, selbstgewählte Bezeichner zu verwenden, also z.B. den internen Bezeichner *GroupBox1* in [Abbildung 13](#) durch *kunde_combobox* ersetzen.

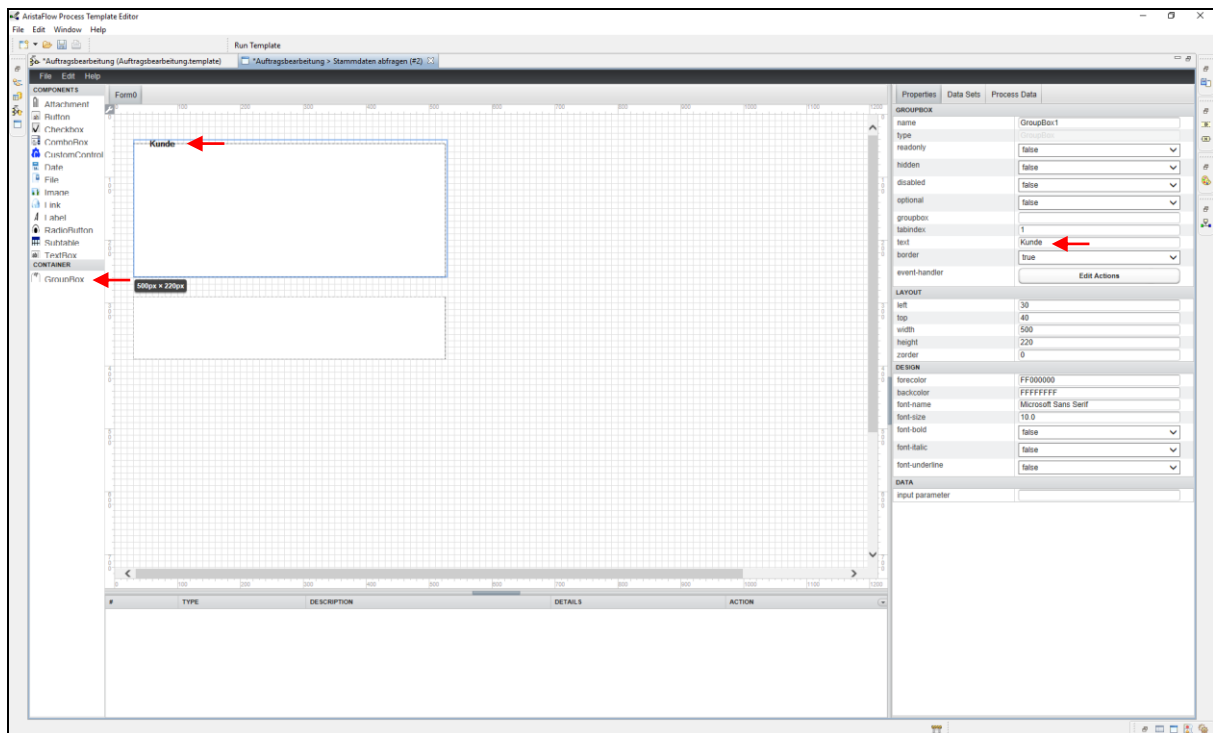


Abbildung 12: Einfügen und Konfigurieren von Controls (1)

Kunde

Kunde auswählen	<input type="text" value="-"/>
Kundenname	<input type="text"/>
Stadt	<input type="text"/>
Bonität	<input type="text"/>

Auftrag

Auftragsnummer	<input type="text" value="-"/>
----------------	--------------------------------

Abbildung 13: Einfügen und Konfigurieren von Controls (2)



Hinweis:

Manchmal möchte man zunächst die Formulare grob inhaltlich gestalten, bevor man sich an die „Feinarbeit“ macht. Der *Process Template Editor* gestattet, an mehreren Prozessschritten gleichzeitig zu arbeiten. Hierzu kann für jeden Prozessschritt, die eine WebForm-Aktivitätenvorlage zugewiesen hat, ein eigenständiger WebForm-Designer geöffnet werden.

4.2 Control-Eigenschaften

Das Aussehen und Verhalten eines Controls lässt sich über den Registerreiter *Properties* beeinflussen (siehe [Tabelle 2](#) sowie [Abbildung 14](#)). Die Bedeutung und Verwendung der *Daten-Controls* werden wir im Kapitel 5 [Verknüpfung von Controls mit Datenbankinhalten](#) behandeln. Einige *Controls-Eigenschaften*, wie z.B. *Hidden*, *ReadOnly*, *Vorder- und Hintergrundfarbe* lassen sich mittels API-Funktionen dynamisch zur Laufzeit verändern. Hierauf werden wir im Kapitel 6 [Implementierung von Formularfunktionen und zusätzlicher Logik](#) näher eingehen.

TextBox/ComboBox/...	
name	Intern vergebener Name des <i>Controls</i> . Mit diesem Bezeichner lässt sich das <i>Controls</i> in JavaScript referenzieren
type	Zeigt den Typ des <i>Controls</i> an
readonly	Legt fest, dass nur lesender Zugriff auf das <i>Control</i> möglich ist
hidden	Versteckt, aber deaktiviert das gewählte <i>Control</i> nicht
disabled	Deaktiviert das <i>Control</i> , blendet es jedoch nicht aus („ausgrauen“)
optional	Gibt an, ob das <i>Control</i> vom Benutzer ausgefüllt werden muss oder nicht
groupbox	Die in der dazugehörigen <i>GroupBox</i>
tabindex	Legt die „Sprungreihenfolge“ der Tabulatortaste fest.
value	Die im Formular angezeigte Beschriftung des <i>Controls</i>
alignment	Horizontale Textausrichtung (Left, Center, Right)
multiline	Entscheidung zwischen einer einzeiligen oder mehrzeiligen <i>TextBox</i>
maxlength	Die Anzahl der Zeichen in der <i>TextBox</i>
event-handler	JavaScript-EventHandler – führen JavaScript-Funktionen nach Triggern eines bestimmten <i>Events</i> aus (<i>onclick</i> , <i>onchange</i> , <i>onblur</i> , usw.)
Layout	
left	Abstand des <i>Controls</i> vom linken Formularrand in Pixel
top	Abstand des <i>Controls</i> vom oberen Formularrand in Pixel
width	Breite des <i>Controls</i> in Pixel
height	Höhe des <i>Controls</i> in Pixeln
zorder	Reihenfolge von <i>Controls</i> entlang der Z-Achse
Design	
forecolor	Farbe des <i>Controls</i>
backcolor	Hintergrundfarbe des <i>Controls</i>
font-name	Verwendete Schriftart
font-size	Verwendete Schriftgröße
font-bold	Schriftstil <i>fett</i>
font-italic	Schriftstil <i>kursiv</i>
font-underline	Schriftstil <i>unterstrichen</i>
Data	
input parameter	Eingabeparameter des Prozessmodells, der beim Aufruf des Prozessschritts als Wert für das <i>Control</i> gesetzt werden soll
output parameter	Ausgabeparameter des Prozessmodells, der beim Verlassen des Prozessschritts den Wert des <i>Controls</i> übergeben bekommt
databasequery	Auswahl der unter <i>Data Sets</i> definierten Datenbankabfrage
databasecolumn	Auswahl der Datenbankspalte, die den Input für das Füllen des <i>Controls</i> liefert
fillmode	<i>Static</i> bei fest definierten Werten / <i>Dynamic</i> bei Datenbankabfragen

Tabelle 2: Änderbare Eigenschaften des Controls

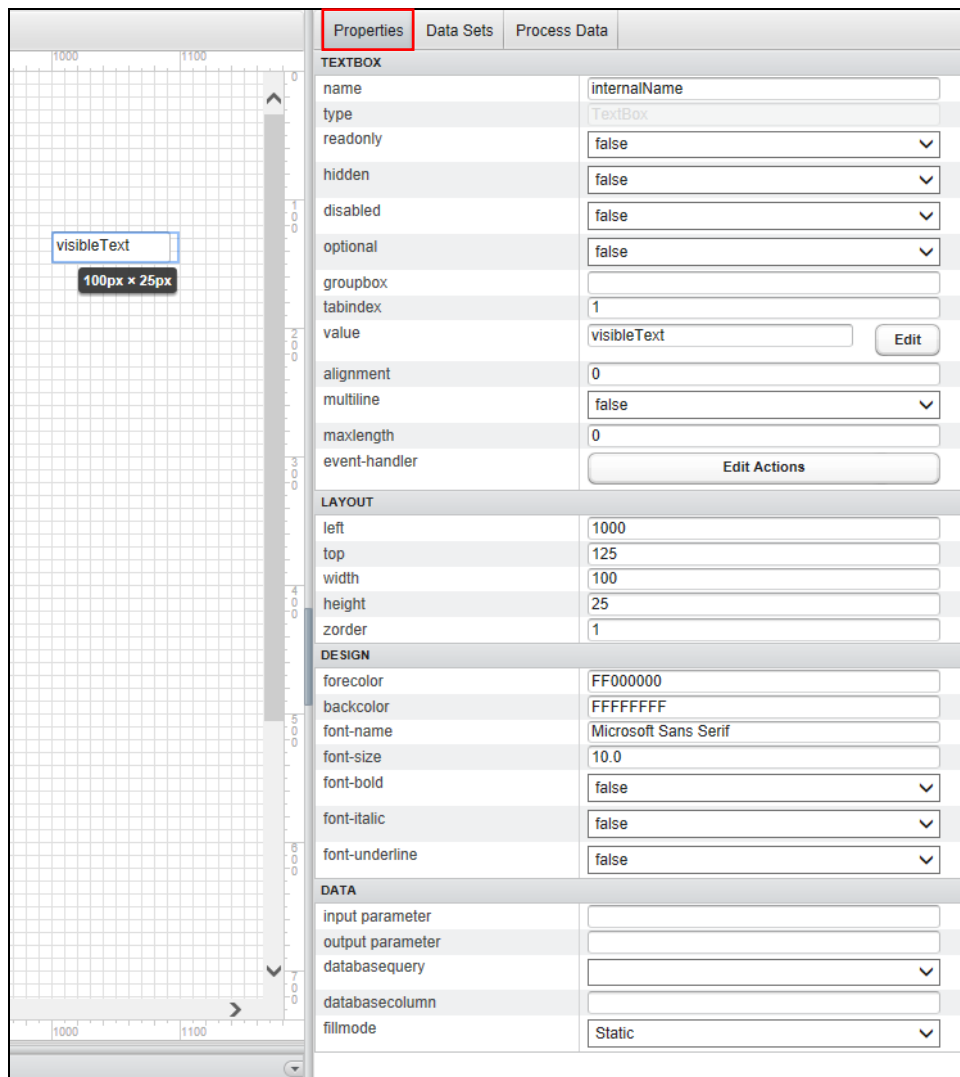


Abbildung 14: Properties eines Controls

4.3 Eingabe- und Ausgabeparameter zuweisen

Sofern im entsprechenden Prozessschritt des Prozessmodells Ein- und Ausgabeparameter festgelegt wurden, lassen sich diese über zwei Vorgehensweisen einem Control zuweisen:

- **Alternative 1:** Der gewünschte Parameter wird aus dem Bereich *Process Data* per Drag'n'Drop auf das entsprechende *Control* gezogen
- **Alternative 2:** Man wählt das *Control* aus und trägt in dessen *Properties* unter *input parameter* oder *output parameter* den Namen des Parameters ein. Hierbei ist zu beachten, dass der **FillMode** auf **Static** gesetzt wird (siehe [Abbildung 14](#) unterster Eintrag).

Im Formular *Stammdaten abfragen* sind die fünf Output-Parameter (siehe [Abbildung 5](#)) zu verknüpfen.

[Abbildung 15](#) und [Abbildung 16](#) zeigen die beiden Alternativen am Beispiel der Verknüpfung des *output parameters AuftrNr* mit dem *Control auftrnr_combobox*. Die korrekte Eingabe des Parameternamens bei Alternative 2 vorausgesetzt, führt dies in beiden Fällen zu dem in [Abbildung 17](#) dargestellten Resultat.

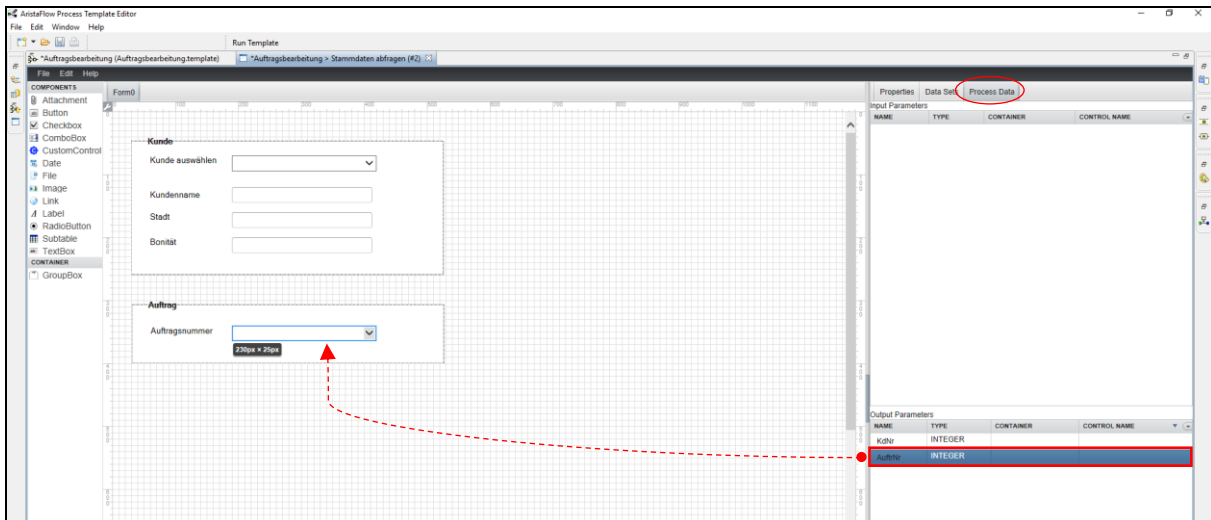


Abbildung 15: Verknüpfen eines Parameters mit einem Control (Alternative 1)

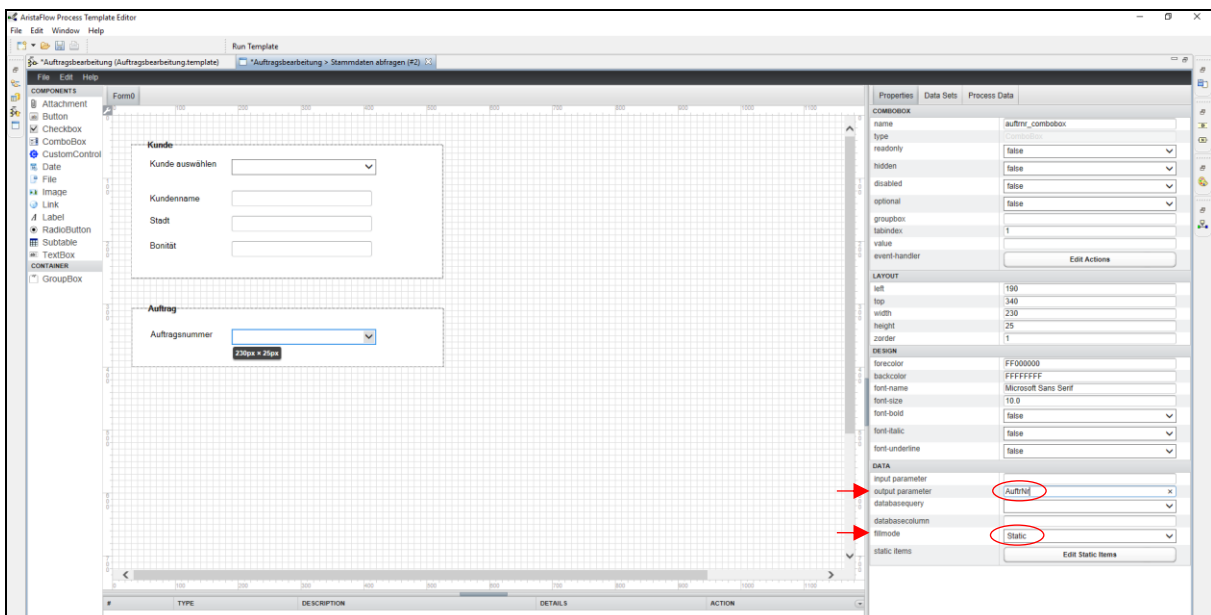


Abbildung 16: Verknüpfen eines Parameters mit einem Control (Alternative 2)

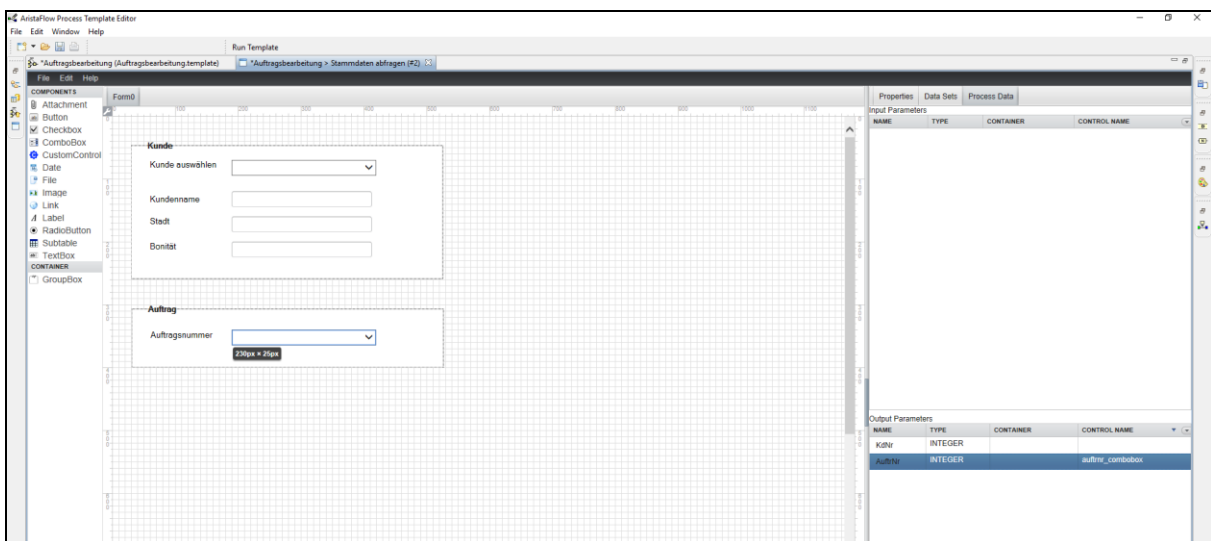


Abbildung 17: Resultierende Verknüpfung des Ausgabeparameters mit dem gewählten Control

In analoger Weise verknüpfen wir den *output parameter KdNr* mit dem *Control kunden_combobox*. In diesem *Control* werden zur Laufzeit die Liste der Kundennamen zur Auswahl angeboten werden (*Abbildung 17*). Wie die *Abbildung Kundename → KdNr* realisiert wird, um das Feld *Kundename* zu füllen, werden wir in Kapitel 5 *Verknüpfung von Controls mit Datenbankinhalten* kennen lernen.

4.4 Mehr-Blatt-Formulare

Der WebForm-Designer unterstützt auch die Realisierung von Mehr-Blatt-Formularen, wobei jedes Blatt durch einen eigenen Reiter am linken oberen Rand angesprochen wird. Die Beschriftung dieser Reiter können wir in den *Properties* dieses Blattes festlegen (*Abbildung 18*).

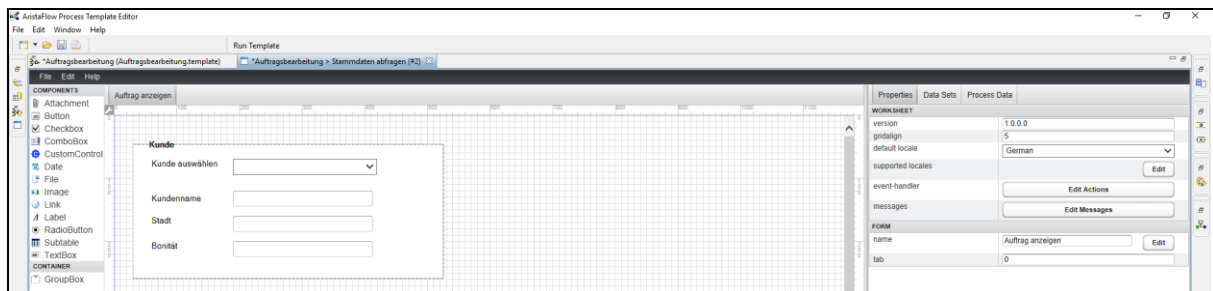


Abbildung 18: Benennung der Formular-Reiter

Ein neues Formularblatt wird mittels *Edit → Add Subform* (*Abbildung 19*) eingefügt.

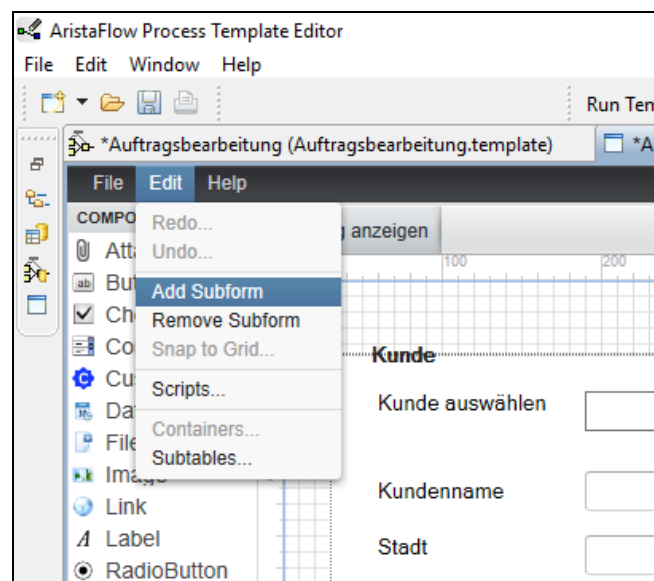


Abbildung 19: Einfügen eines Formularblatt

Auf den weiteren Formularblättern stehen ebenfalls die Ein-/Ausgabeparameter zur Verknüpfung zur Verfügung. D.h. *Controls* auf den verschiedenen Formularblättern „teilen“ sich ggf. denselben Parameter. (Nicht davon irritieren lassen, dass in der *Process Data*-Ansicht nur eine Verknüpfung angezeigt wird; ausschlaggebend sind die entsprechenden *Property*-Einträge bei den *Controls*!). Weitere Einzelheiten zum Arbeiten mit Mehrblatt-Formularen finden sich im Abschnitt 6.4 *Arbeiten mit Subformularen*.

4.5 Zusammenspiel von Controls und Events (Ereignissen)

Die interne Steuerung der Formularverarbeitung bzw. seines „Verhaltens“ erfolgt (wie üblich) ereignisgesteuert. So löst z.B. das Laden des Formulars ein *Load*-, ein Mausklick auf einem *Control*, ein *Click*- und ein Eintrag in einem Eingabefeld ein *Change*-Ereignis aus. Jedes *Control* hat in seinen *Properties* einen *EventHandler*, bei dem man hinterlegen kann, auf welche Ereignisse dieses *Control* reagieren soll und welche Aktionen dies ggf. auslösen soll.

Das *Load*-Ereignis führt z.B. dazu, dass alle im Formular mit irgendwelchen *Controls* assoziierten SQL Anfragen zur Ausführung gebracht werden. Einige davon werden bereits Ergebnisse liefern können, während andere (nur intern) sichtbar fehlschlagen, weil z.B. in der Anfrage referenzierte *Controls* noch nicht mit Werten versorgt wurden.

Wenn beispielsweise ein *Control A* mit der Ausgabe einer SQL-Anfrage „gefüttert“ werden soll, die in ihrer *WHERE*-Bedingung ein anderes *Control B* referenziert, das vom Anwender mit einem Wert versorgt werden muss, dann wird die SQL-Anfrage zunächst (intern und für den Anwender nicht sichtbar) beim Laden des Formulars fehlschlagen.

Damit zu gegebener Zeit die Anfrage nochmals ausgeführt wird, wird man in der Regel bei *Control A* ein *Change*-Ereignis hinterlegen, welches bei *Control B* ein *Reload*-Ereignis auslöst, das wiederum die erneute Ausführung der mit B verknüpften SQL-Anfrage (und damit ggf. das Befüllen von B) auslöst.

Im folgenden Kapitel (Kapitel 5 *Verknüpfung von Controls mit Datenbankinhalten*) gehen wir auf die Hinterlegung von Datenbankabfragen, deren Verknüpfung mit *Controls* sowie die hierzu relevanten *Events* ein. In Kapitel 6 *Implementierung von Formularfunktionen und zusätzlicher Logik* behandeln wir dann weitere Eventtypen und beschreiben, wie man mittels JavaScript weitere Formularlogik implementieren kann. In Kapitel 7 Umgang mit Parametern vom Typ *USERDEFINED subtable* gehen wir auf sog. „Subtabellen“ bzw. den Umgang mit Parametern vom Typ *USERDEFINED subtable* ein und in Kapitel 8 *Anwendungsbeispiele* zeigen wir das Zusammenwirken dieser Konzepte anhand ausgewählter Anwendungsbeispiele.

5 Verknüpfung von Controls mit Datenbankinhalten

Die Verknüpfung von *Controls* mit Datenbankinhalten besteht aus mehreren Teilen und zwar der

- Einrichtung von Datenbankverbindungen
- Formulierung von Hinterlegung von Datenbankabfragen
- Verknüpfung von *Controls* mit Datenbankabfragen
- Automatischen oder ereignisgesteuerten Ausführung von Datenbankabfragen zur Aktualisierung von *Controls*

5.1 Einrichten von Datenbankinhalten

Eine Datenbankabfrage ist immer an eine hinterlegte Verbindung zur Datenbank gebunden. Bevor man also Datenbankabfragen hinterlegen kann, muss man zuvor eine oder mehrere Datenbankverbindungen hinterlegen. In den entsprechenden Konfigurationsdialog gelangt man in der Ansicht *Data Sets* mittels Rechtsklick *Add Connection*, wie in [Abbildung 20](#) dargestellt.

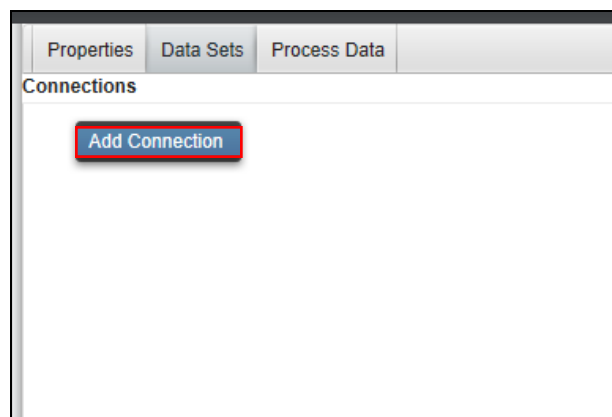


Abbildung 20: Einrichten einer neuen Datenbankverbindung

Klickt man auf den angezeigten Verbindungsnamen wie in [Abbildung 21 a\)](#) dargestellt, öffnet sich das Konfigurationsfenster ([Abbildung 21 b\)](#)), in dem man die Verbindungsdaten ändern bzw. neu hinterlegen kann.

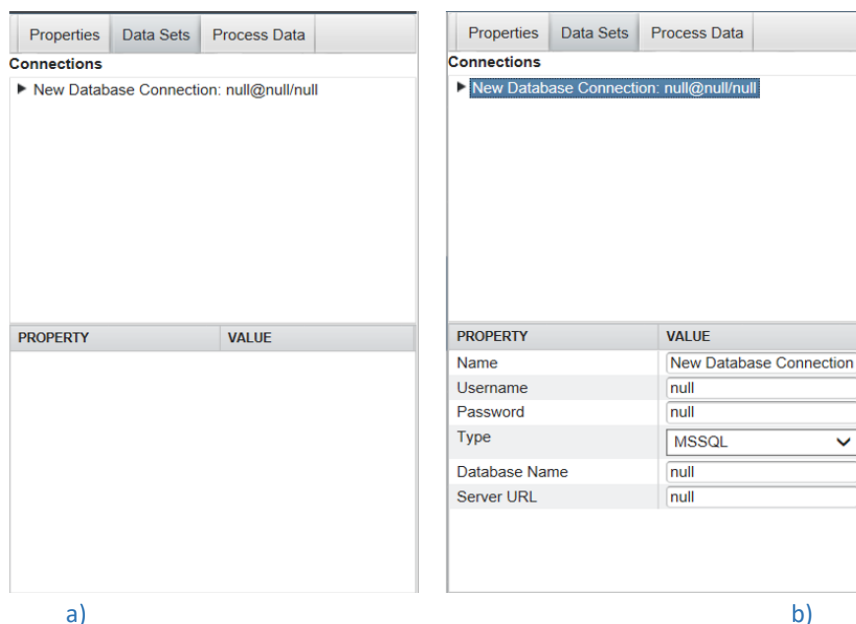


Abbildung 21: Konfigurationsdialog

Der WebForm-Designer unterstützt derzeit die folgenden Datenbanksysteme:

- MMSQL
- Oracle
- MySQL
- Derby
- DB2
- DataSource
- PostgreSQL
- H2

Wir vergeben zuerst einen Namen für die Verbindung und passen weitere Eigenschaften entsprechend an. Für eine PostgreSQL-Datenbank namens *legotrailerdb* mit Username *postgres* und Passwort *postgres* könnte dieser Eintrag wie in [Abbildung 22](#) dargestellt aussehen.

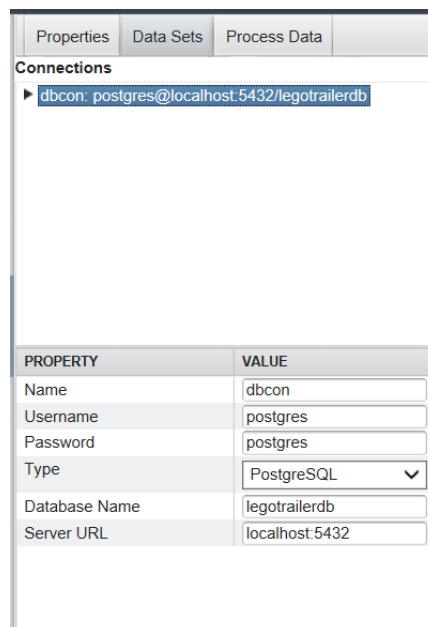


Abbildung 22: Konfigurationsbeispiel

Für die folgenden Beispiele wurde das Datenbanksystem PostgreSQL, wie in [Abbildung 22](#) konfiguriert und mit den Beispiel-Tabellen der *LegoTrailer AG* (siehe [Vorbemerkungen](#) sowie [Anhang: Beispieltabellen](#)) befüllt.

5.2 Formulierung und Hinterlegung von Datenbankabfragen

5.2.1 Allgemeines

Mittels Rechtsklick auf der gewählten Datenbankverbindung erhalten wir die Möglichkeit Abfragen zu erstellen, zu löschen oder zu verändern (*Abbildung 23 a*). Im Feld *Name* (*Abbildung 23 b*) können wir den Namen der Abfrage festlegen.

Ein Klick auf *Command* und ein weiterer Klick auf den Button *Open editor* (*Abbildung 23 b*) öffnet ein Fenster für die Eingabe der SQL-Anweisung. Nach Abschluss steht die Abfrage in der Auswahlliste unter der Datenbankverbindung.

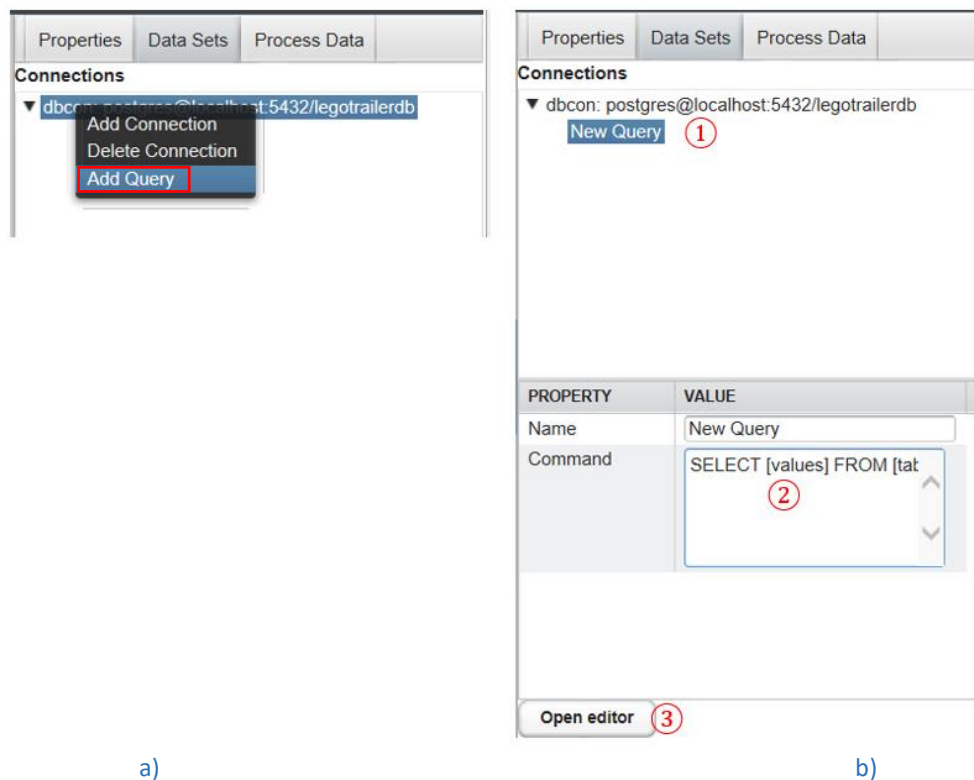


Abbildung 23: Erstellen einer neuen Datenbankabfrage

Die Abfragen sind normale SQL-Abfragen. Bezüglich der SQL-Syntax gibt es im Prinzip keine Einschränkungen. Hier ist man nur durch die Fähigkeiten des verwendeten Datenbanksystems beschränkt.

Achtung:

Trotz SQL-Standardisierung gibt es subtile syntaktische Unterschiede zwischen den verschiedenen Datenbanksystemen. Daher im Zweifelsfall immer die Ausführbarkeit der SQL-Anfragen vorher separat testen, um etwa mit Hilfe des SQL-Explorers.

In den *WHERE*-Klauseln kann man Bezug auf *Controls* nehmen.

Angenommen, wir haben eine *TextBox kdnr_textbox*, dann könnten wir diese in der *WHERE*-Klausel in der Form

`WHERE attributname = $kdnr_textbox$`

referenzieren. Man beachte die *\$-Zeichen*, zwischen die der Name des *Controls* eingeschlossen wird.

5.2.2 Arten von Datenbankabfragen

Bei den Datenbankabfragen muss man hinsichtlich des *ResultSet*-Typs unterscheiden zwischen

- Ein-Tupel-Abfragen (liefern stets genau (oder max.) ein Tupel zurück)
- Tupel-Menge-Abfragen
 - Allgemeine Mehrspalten-Ergebnismenge
 - Einspalten-Ergebnismenge
 - Key-Value-Paar-Ergebnismenge

Eigenschaften:

- Eine **Ein-Tupel-Abfrage** liefert (max.) ein Resultat-Tupel zurück. Diese Art von Abfragen wird mit den *Controls* verknüpft, die nur einzelne Werte aufnehmen können, wie z.B. die *TextBox*. Hierbei können sich mehrere *TextBoxen* eine Abfrage „teilen“, da in den Properties der *TextBox* spezifiziert werden kann, welches Attribut dieser *TextBox* zugeordnet ist (siehe Beispiele im folgenden Abschnitt).
- Ein **Tupel-Menge-Abfrage** liefert eine Tupelmenge zurück, die auch leer sein kann.
 - Die **allgemeine Mehrspalten-Menge** (aMM) ist die „klassische“ Ergebnismenge, die auch wieder als Tabelle im Formular angezeigt werden soll. Dies wäre in diesem Fall das *Control Subtable*, das im zweiten Formular *Auftrag anzeigen* Verwendung findet.
 - Die **Einspalten-Ergebnismenge** ist einerseits ein Spezialfall der aMM und deshalb gilt für Sie auch das vorstehend Gesagte. Sie eignet sich darüber hinaus aber auch für das Control *Combobox*. Dieses zeigt in diesem Fall eine Liste von Einträgen an, von denen einer ausgewählt werden kann und dann den „Wert“ des *Controls* repräsentiert. (Auf diesen kann dann z.B. in Datenbankabfragen in der oben beschriebenen Form Bezug genommen werden.)
 - Die **Key-Value-Paar-Ergebnismenge** ist eine spezielle Form einer zweispaltigen Ergebnismenge (und damit natürlich auch wieder eine aMM), und zwar mit fest vorgegebenen Spaltennamen *key* und *value* (Kleinschreibung beachten!).
Wenn sie einem *Combobox-Control* zugewiesen wird, dann wird in dessen Auswahlliste der Wert der Spalte *key* angezeigt und bei Auswahl eines Elements dessen *value* ausgegeben (siehe SQL-Anweisung im folgenden Abschnitt bzw. in *Abbildung 24*).

Tipp:

Wenn wir in der *Groupbox*-Liste pro Zeile mehr als nur einen Namen oder eine Zahl anzeigen lassen möchten, dann können wir das in begrenztem Umfang tun, indem wir mehrere Attribute in der *SELECT*-Klausel zu einem Ausgabe-Attribut zusammenfassen.

Angenommen, wir möchten in der Auswahlliste Kundenname und Ort, durch Komma getrennt, anzeigen. Dann könnten wir dies mittels `SELECT KdName || „ , „ || KdStadt FROM Kunden ...` erreichen.

Wenn wir z.B. *KdNr* gefolgt von *KdName* anzeigen lassen möchten, so müssen wir zunächst das *INTEGER*-Attribut *KdNr* in ein *Character*-Attribut umwandeln, damit der Konkatenationsoperator anwendbar ist. Dies könnte z.B. in der folgenden Form geschehen:

```
SELECT CAST(KdNr AS CHAR(4)) || „ , „ || KdName FROM Kunden ...
```

Leider bieten die im SQL-Standard vorgesehenen Stringfunktionen nur begrenzte Möglichkeiten, die Ausgabe nach eigenen Wünschen „schön“ formatiert zu gestalten. Hier müsste man bei Bedarf dann auf proprietäre Erweiterungen des DBMS zurückgreifen (sofern vorhanden) oder eine geeignete *Userdefined Function* implementieren.

5.2.3 Abfragen für Beispiel-Formulare

Das Formular *Stammdaten abfragen* soll zur Laufzeit das in *Abbildung 24* dargestellte Aussehen und Funktionalität haben: In der *ComboBox Kunde auswählen* (interner Name: *kunde_combobox*) soll eine Liste von Kundennamen zur Auswahl angeboten werden. Für den ausgewählten Kunden sollen in nachfolgenden *TextBoxen* der *Name* des Kunden, die *Stadt* und die *Bonität* sowie in der *ComboBox* die Liste der *Auftragsnummer* zur Auswahl angezeigt werden. Hierfür benötigen wir insgesamt 3 Datenbankabfragen:

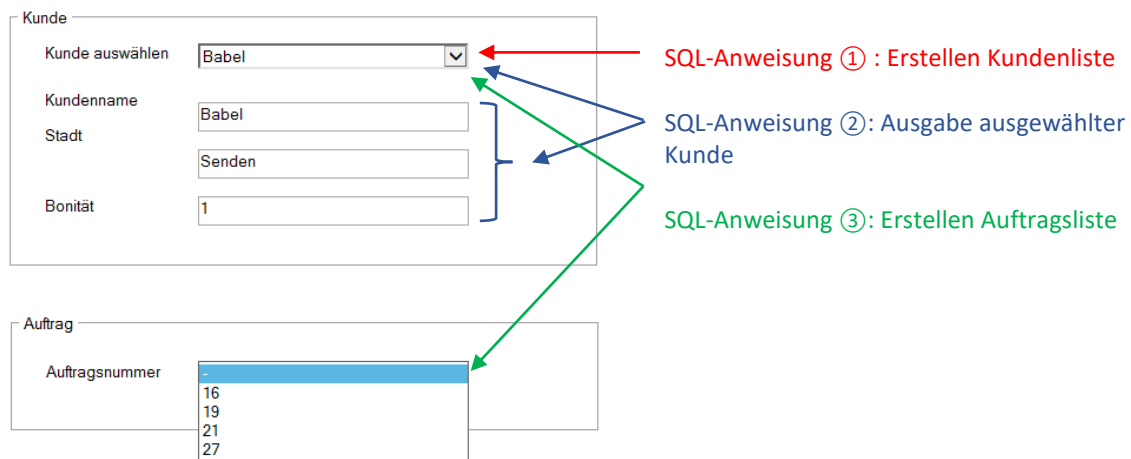


Abbildung 24: Formular Stammdaten abfragen

- Die SQL-Anweisung ①: **Erstellen Kundenliste** dient zum Befüllen der *Dropdown*-Liste *kunde_combobox* mit dem Namen der Kunden. Sie liefert *KdNr* und *KdName* aus der Tabelle *Kunden* in Form einer „Key-Value-Paar-Ergebnismenge“ wie im vorherigen Abschnitt beschrieben, indem sie *KdName* als *key* und *KdNr* als *value* zurückliefert (*Abbildung 25*).

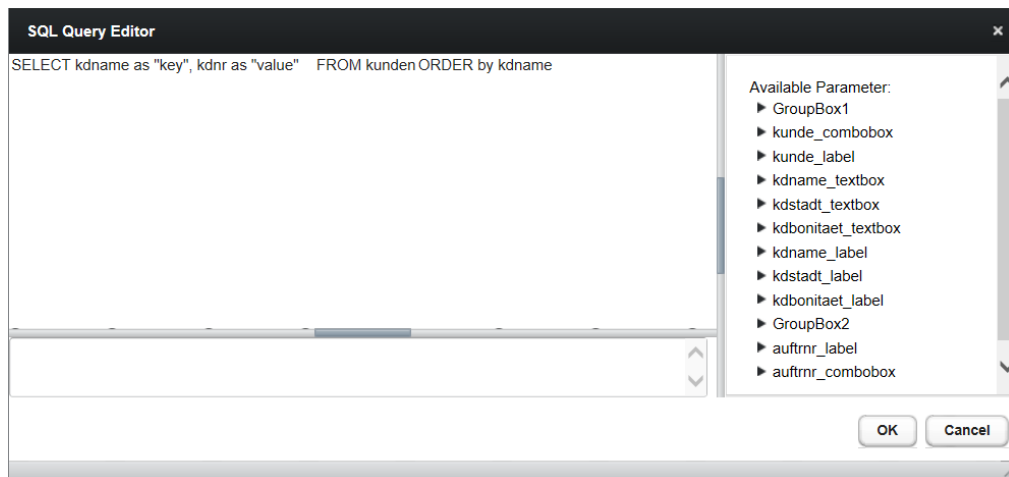


Abbildung 25: SQL-Anweisung Erstellen Kundenliste

- Die SQL-Anweisung ②: **Ausgabe ausgewählter Kunde** dient zum Befüllen der *TextBoxen* mit den internen Namen *kdname_textbox*, *kdstadt_textbox* und *kdbonitaet_textbox*. Sie liefert als Result *KdNr*, *KdName* und *Bonitaet* entsprechend der in *ComboBox kunde_combobox* gewählten Kundennummer (*Abbildung 26*).

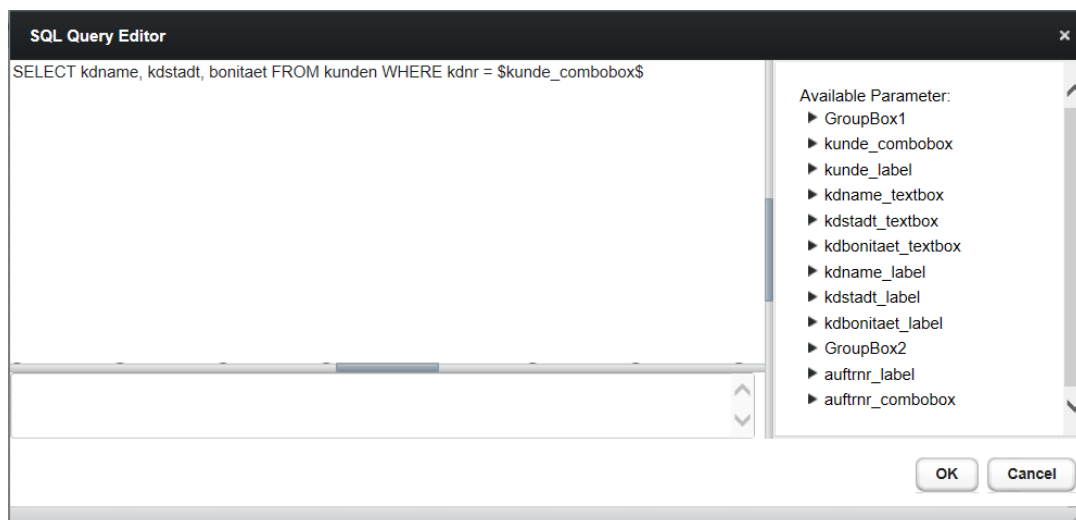


Abbildung 26: SQL-Anweisung Ausgabe ausgewählter Kunde

- Die SQL-Anweisung ③: Erstellen Auftragsliste dient dazu, die Groupbox *auftrnr_groupbox* mit der Liste der Auftragsnummern entsprechend der in ComboBox *kunde_combobox* gewählten Kundennummer zu befüllen. Hierzu erstellen wir eine Einspalten-Ergebnistabelle (wie im vorherigen Abschnitt beschrieben) für *AuftrNr* (Abbildung 27).

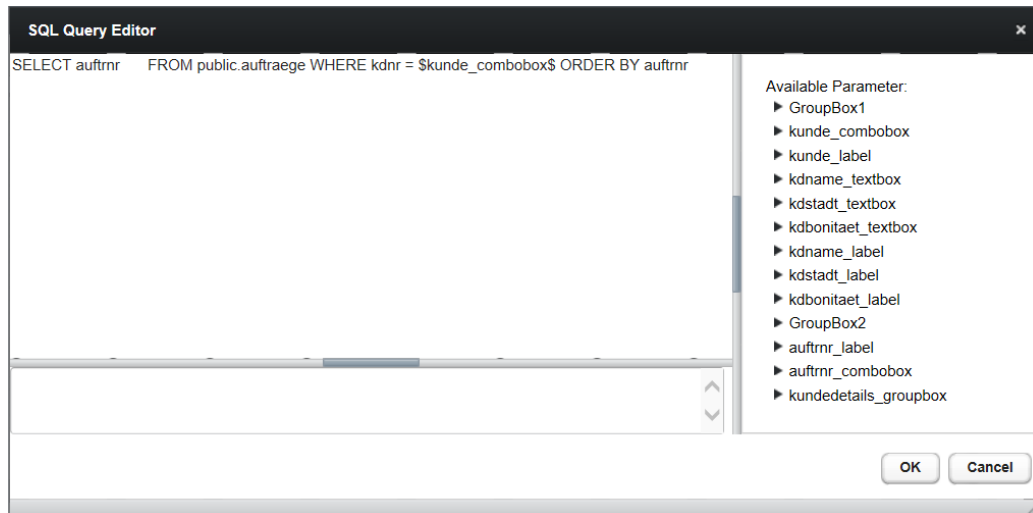


Abbildung 27: SQL-Anweisung Erstellen Auftragsliste

Insgesamt haben wir damit nun die in [Abbildung 28](#) dargestellten Abfragen definiert. In Abschnitt 5.3 [Zuordnung von Datenbankabfragen zu Controls](#) wird beschrieben, wie diese mit den *Controls* verbunden werden.

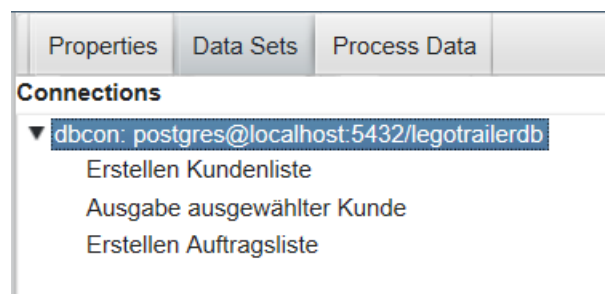


Abbildung 28: Für Formular Stammdaten abfragen definierte Abfragen

Im Formular *Auftrag anzeigen* (Abbildung 29) soll für den im vorherigen Formularschritt *Stammdaten abfragen* (Abbildung 24) ausgewählten Kunden dessen Name sowie für die dort ausgewählte Auftragsnummer die Liste der Auftragspositionen sowie der Auftragswert als Summe der Auftragspositionen berechnet werden (auf die Berechnung dieser Summe und die hierfür benötigte JavaScript-Funktion werden wir erst in Abschnitt 8.3

Zeilenweise Verarbeitung von Subtabellen eingehen.)

Auftrag

Kundenname

Auftragspositionen

TEILENR	ANZAHL	FARBE	PREIS

Gesamtpreis

Abbildung 29: Formular Auftrag anzeigen

Im Prinzip könnten wir den Kundennamen diesem Prozessschritt direkt als Aufrufparameter übergeben, da wir diesen im Prozessschritt *Stammdaten abfragen* bereits bestimmt haben. Wir wollen jedoch annehmen, dass wir anstatt dessen die Kundennummer *KdNr* übergeben, so dass wir den Kundennamen mittels einer Datenbankabfrage bestimmen müssen. Ebenso soll die gewählte Auftragsnummer *AuftrNr* als Aufrufparameter an unseren Prozessschritt übergeben werden, für den wir nun per Datenbankabfrage die zugehörigen Auftragspositionen bestimmen müssen.

Bei SQL-Anfragen müssen sich somit in ihren *WHERE*-Klauseln auf Werte von Aufrufparametern beziehen, und zwar *KdNr* bzw. *AuftrNr*. Dies geht nicht direkt, d.h. eine SQL-Anfrage kann in WebForm nur *Controls*, nicht aber Aufrufparameter referenzieren. Wir ergänzen deshalb unser Formular um zwei weitere *TextBoxen* *kdnr_textbox* und *auftrnr_textbox* (Abbildung 30) und verknüpfen diese jeweils mit den entsprechenden Aufrufparametern. Damit stehen sie uns nun im Formular als Bezugswerte zur Verfügung. – Die beiden *TextBoxen* deklarieren wir als „versteckt“ (d.h. sie werden zur Laufzeit nicht angezeigt), in dem wir in ihren Properties *Hidden* auf *true* setzen.

Zum Befüllen des *Controls* *kdnr_textbox* hinterlegen wir eine Datenbankabfrage *Bestimmen Kundennamen* (Abbildung 31) und für das Befüllen unserer Subtable eine Abfrage *Bestimmen Auftragspositionen* (Abbildung 32) – Wie die dort dargestellte Tabellenstruktur erzeugt wird, werden wir in Abschnitt 5.3 *Zuordnung von Datenbankabfragen zu Controls* kennen lernen.

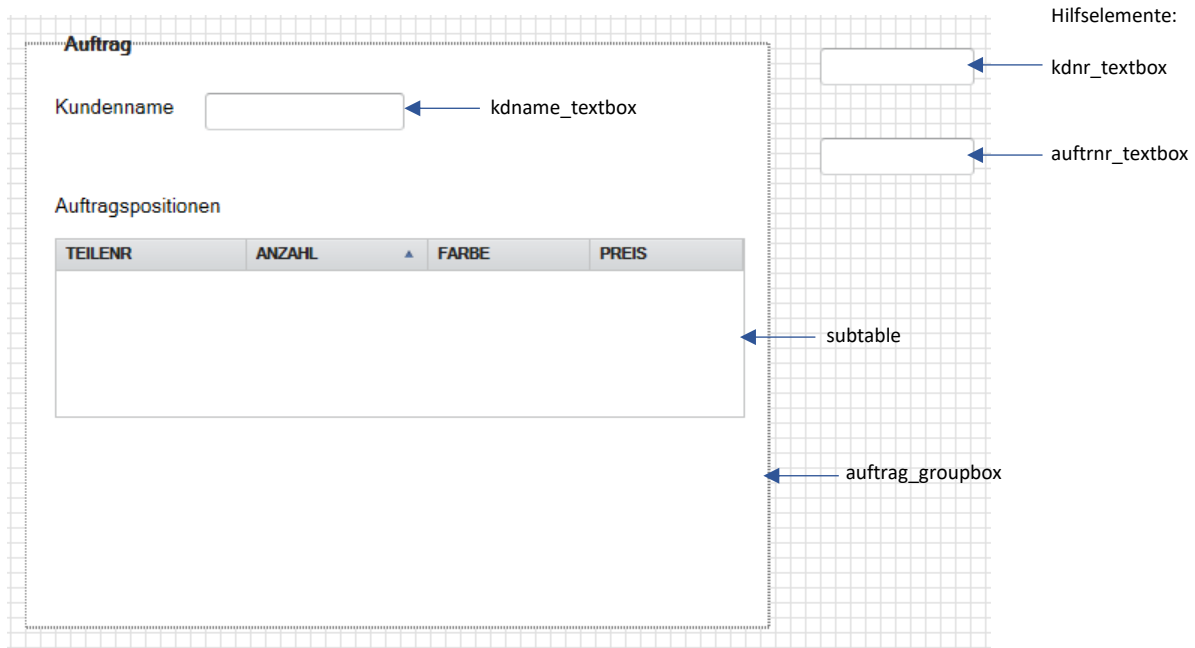


Abbildung 30: Formular Auftrag anzeigen

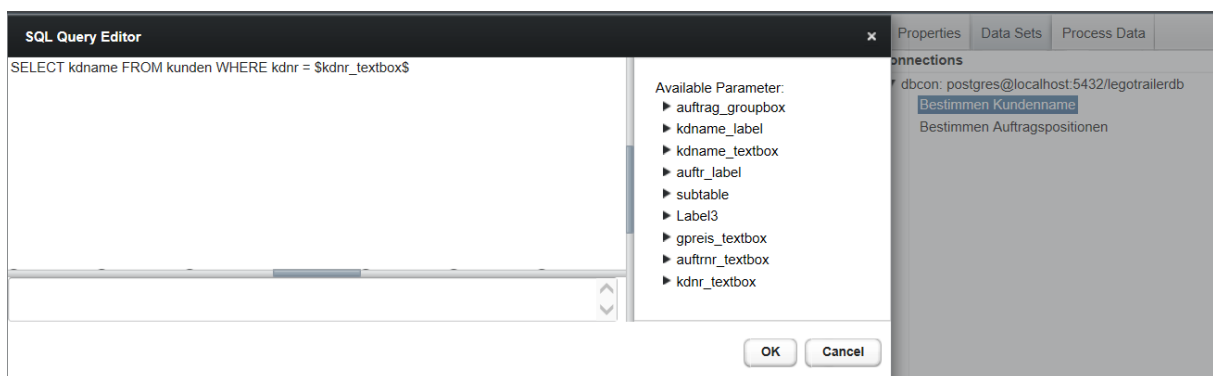


Abbildung 31: Datenbankabfrage Bestimmen Kundenname für Formular Auftrag anzeigen

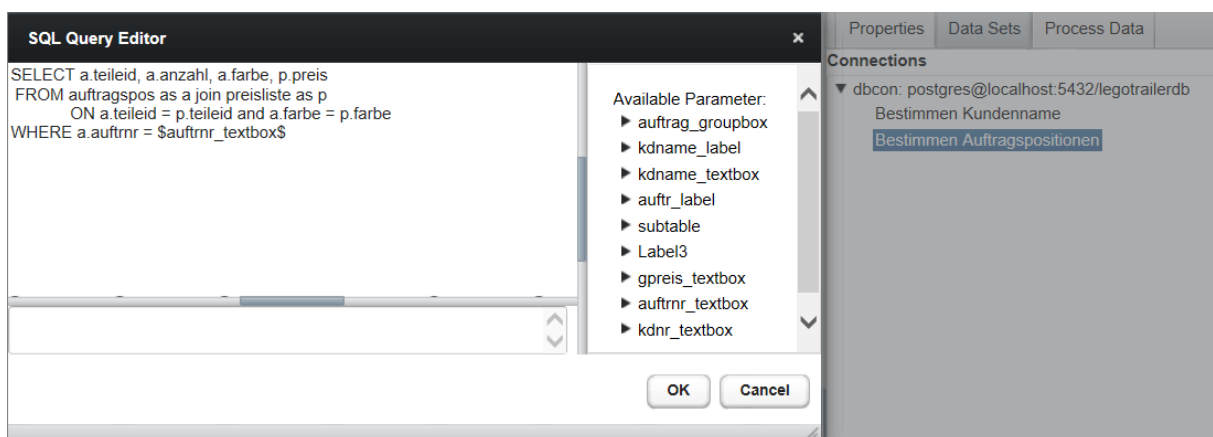


Abbildung 32: Datenbankabfrage Bestimmen Auftragspositionen für Formular Auftrag anzeigen

5.3 Zuordnung von Datenbankabfragen zu Controls

Die Zuordnung von Datenbankabfragen zu *Controls* erfolgt in den *Properties* der *Controls*. Hierzu wählen wir im Feld *DatabaseQuery* den Namen der gewünschten Abfrage aus der angebotenen Liste

aus und wählen bei *Fillmode Database*. Falls die Abfrage mehr als eine Spalte zurückliefert, geben wir im Feld *DatabaseColumn* die gewünschte Spalte an.

Für unser Formular **Stammdaten abfragen** sehen die *Properties* der *Controls*, wie in den Abbildungen *Abbildung 33* bis *Abbildung 37* dargestellt, aus.

DATA	
input parameter	<input type="text"/>
output parameter	KdNr
databasequery	Erstellen Kundenliste ▼
databasecolumn	<input type="text"/>
fillmode	Database ▼
static items	<input type="button" value="Edit Static Items"/>

Abbildung 33: Properties des Controls kunde_combobox

DATA	
input parameter	<input type="text"/>
output parameter	<input type="text"/>
databasequery	Ausgabe ausgewählter Kunde ▼
databasecolumn	kdname
fillmode	Database ▼

Abbildung 34: Properties des Controls kdname_textbox

DATA	
input parameter	<input type="text"/>
output parameter	<input type="text"/>
databasequery	Ausgabe ausgewählter Kunde ▼
databasecolumn	kdstadt
fillmode	Database ▼

Abbildung 35: Properties des Controls kdstadt_textbox

DATA	
input parameter	<input type="text"/>
output parameter	<input type="text"/>
databasequery	Ausgabe ausgewählter Kunde ▼
databasecolumn	bonitaet
fillmode	Database ▼

Abbildung 36: Properties des Controls kdbonitaet_textbox

DATA	
input parameter	<input type="text"/>
output parameter	AuftrNr
databasequery	Erstellen Auftragsliste ▼
databasecolumn	<input type="text"/>
fillmode	Database ▼
static items	Edit Static Items

Abbildung 37: Properties des Controls auftrnr_combobox

Für unser Formular **Auftrag anzeigen** sehen die *Properties* für das Control *kdname_textbox* wie in [Abbildung 38](#) dargestellt aus. Dem Control *subtable* ordnen wir im Feld „Daten“ der *Properties* die Datenbankabfrage *Bestimmen Auftragspositionen* zu ([Abbildung 39](#)). – Auf die im *subtable*-Control zu realisierende Tabellenstruktur und deren Verknüpfung mit dieser Datenbankabfrage gehen im Abschnitt 5.5 [Erstellen von Tabellen \(Subtables\)](#) ein.

DATA	
input parameter	<input type="text"/>
output parameter	<input type="text"/>
databasequery	Bestimmen Kundename ▼
databasecolumn	kdname
fillmode	Database ▼

Abbildung 38: Properties des Controls kdname_textbox

DATA	
input parameter	<input type="text"/>
output parameter	<input type="text"/>
databasequery	Bestimmen Auftragspositionen ▼
subtable	<input type="text"/> ▼
columns	Edit Columns

Abbildung 39: Properties des Controls subtable

5.4 Ereignisgesteuerte DB-Anfragen-Ausführung zur Aktualisierung von Controls

Beim Starten einer WebForm-Aktivität (genauer: beim Laden des Formulars) werden alle diesem Formular zugeordneten Datenbankabfragen **automatisch** ausgeführt. Alle *Controls*, die mit Datenbankabfragen verbunden sind, die hierbei erfolgreich ausgeführt werden können, werden mit den entsprechenden Werten aus dem Abfrageergebnis versorgt. Evtl. können nicht alle diese Abfragen erfolgreich ausgeführt werden, weil sie z.B. *Controls* referenzieren, die zu diesem Zeitpunkt noch keinen Wert aufweisen. (Anmerkung: Der „Fehlschlag“ solcher Datenbankabfragen wird formularintern abgefangen und ist für den Anwender nicht sichtbar.) Für diese Controls muss die (erneute) Ausführung der Datenbankabfrage zum geeigneten Zeitpunkt explizit angestoßen werden, indem wir ein *Reload*-Ereignis für das betroffene Control erzeugen (siehe unten).

Wenn wir die Datenbankabfragen für beide Formulare wie gezeigt im *Data Sets*-Fenster hinterlegt haben, diese in den *Properties* der *Controls* wie angegeben referenziert werden und den Prozess im *TestClient* auf Basis der *LegoTrailer*-Datenbank ausführen, dann bekommen wir bei *Kunde auswählen* eine Auswahlliste angeboten (*Abbildung 40 a*), aus der wir auch einen Eintrag auswählen können. Allerdings hat diese Auswahl noch keine Auswirkung auf die anderen *Controls*; d.h. sie bleiben leer, wie z.B. *Kundenname*, *Stadt*, *Bonität* und *Auftragsnummer* in *Abbildung 40 b*.

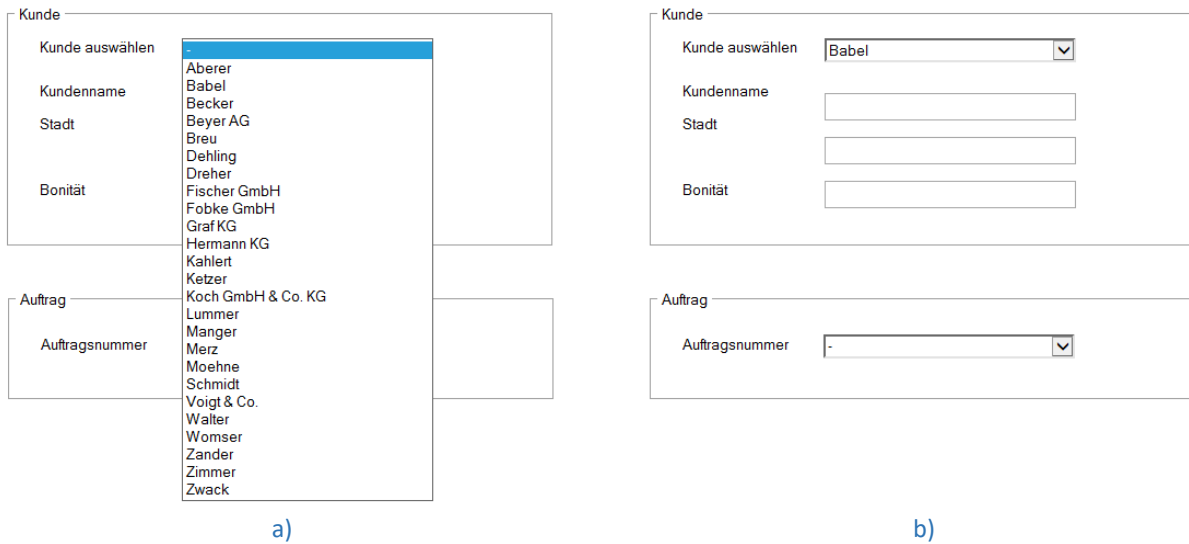


Abbildung 40: Laufzeitansicht des Formulars Stammdaten abfragen

Damit diese *Controls* automatisch „befüllt“ werden, sobald wir in dem „übergeordneten“ Control den benötigten Wert hinterlegt haben, müssen wir von diesem aus die Aktualisierung der „abhängigen“ Controls und damit die (erneute) Ausführung der mit ihnen assoziierten Datenbankabfragen anstoßen. In unserem Beispiel müssen wir beim *Control kunde_combobox* hinterlegen, dass bei Wertänderung die Controls *kdname_textbox*, *kdstadt_textbox*, *kdbonitaet_textbox* und *auftrnr_combobox* aktualisiert werden sollen.

In den *Properties* des *Controls kunde_combobox* klicken wir im Feld *event-handler* auf den Button *Edit Actions* ②. In dem sich öffnenden Dialog des *EventHandlers* klicken wir auf den Button *Add event action* ③ wie in *Abbildung 41* dargestellt.

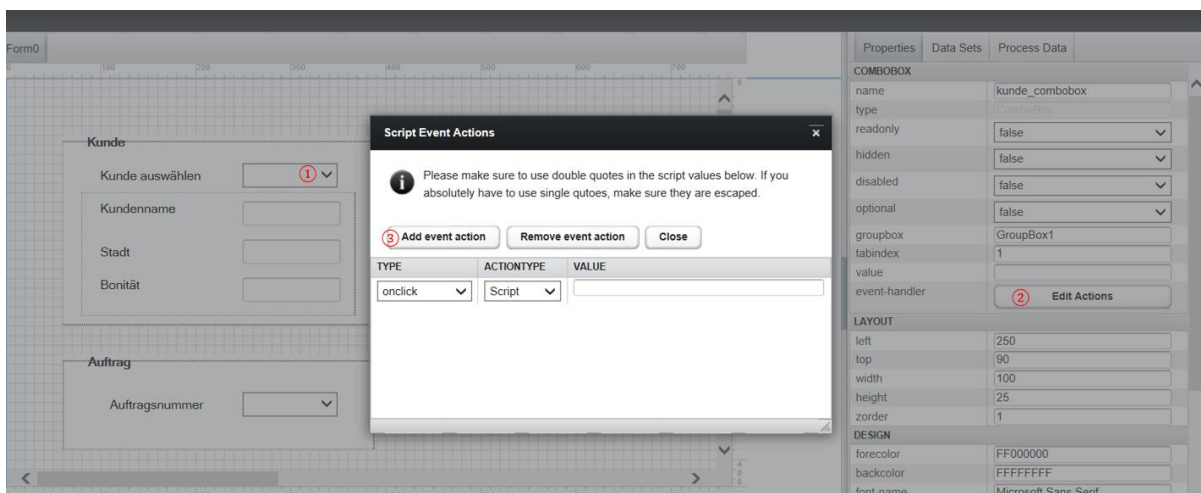


Abbildung 41: Festlegen eines EventHandlers (1)

Nun muss unter *Type* → *onclick* sowie unter *Actiontype* → *Reload* ausgewählt werden (*Abbildung 42*).

Im Feld *Value* tragen wir die zu aktualisierenden *Controls* durch Kommas getrennt ein (also *kdname_textbox, kdstadt_textbox, kdbonitaet_textbox, auftrnr_combobox* in [Abbildung 43](#)).

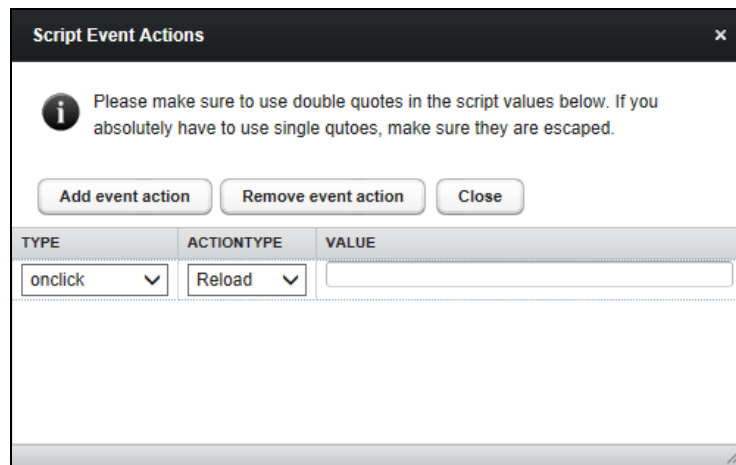


Abbildung 42: Festlegen eines EventHandlers (2)

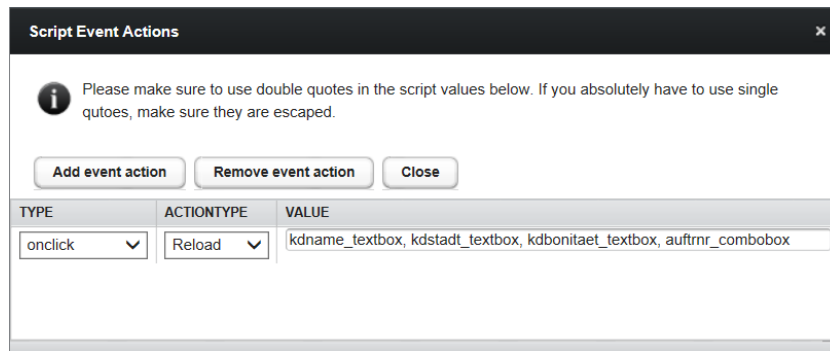


Abbildung 43: Festlegen eines EventHandlers (3)

Nach Sichern aller Änderungen erhalten wir bei Ausführung unseres Prozesses das Formular *Stammdaten abfragen* nun mit den gewünschten Inhalten angezeigt ([Abbildung 44](#)).

Kunde

Kunde auswählen

Kundenname

Stadt

Bonität

Auftrag

Auftragsnummer

- 16
- 19
- 21
- 27

Abbildung 44: Laufzeitansicht von Stammdaten abfragen nach Einfügen des EventHandlers

Für das Formular *Auftrag anzeigen* benötigen wir keinen Eventhandler, da der für die Datenbankabfrage erforderliche Wert per Aufrufparameter an das *Control auftrnr_textbox* übergeben wird und somit zum Ladezeitpunkt des Formulars bereits zur Verfügung steht.

5.5 Erstellen von Tabellen (Subtables)

Beim Anlegen von Tabellen *Subtables* muss man unterscheiden zwischen Tabellen, die nur formularintern (interne Subtabellen) verwendet werden und solchen, die via Ausgabeparameter (Output-Subtabellen) an nachfolgende Prozessschritte weitergereicht werden.

Interne Subtabellen finden vor allem zur Darstellung von Datenbank-Anfrageergebnissen Verwendung. Im Prinzip können diese im Formular auch verändert werden, jedoch ist die Subtabelle als Ganzes kein Ausgabeparameter der Formularaktivität. **Output-Subtabellen** werden im Gegensatz dazu auf einen Ausgabeparameter der Formularaktivität abgebildet. Aus diesem Grund muss – zusätzlich zur graphischen Repräsentation im Formular – noch eine *subtable*-Datenstruktur angelegt und mit der graphischen Repräsentation verknüpft werden, welche die Subtabelle als „Ausgabeobjekt“ kapselt und über den assoziierten Ausgabeparameter nach außen gibt.

5.5.1 Realisierung interner Subtabellen

Um in einer Tabelle Informationen anzeigen zu können, müssen zuerst passende Spaltendefinitionen erstellt werden. Hierzu werden in den *Properties* des *Controls SubtableView subtable* (*Abbildung 45*) die gewünschten Spalten definiert. Nach einem Klick auf den Button *Edit Columns* öffnet sich der *Column-Editor*, in dem man die gewünschten Spalten anlegen kann (*Abbildung 46*).

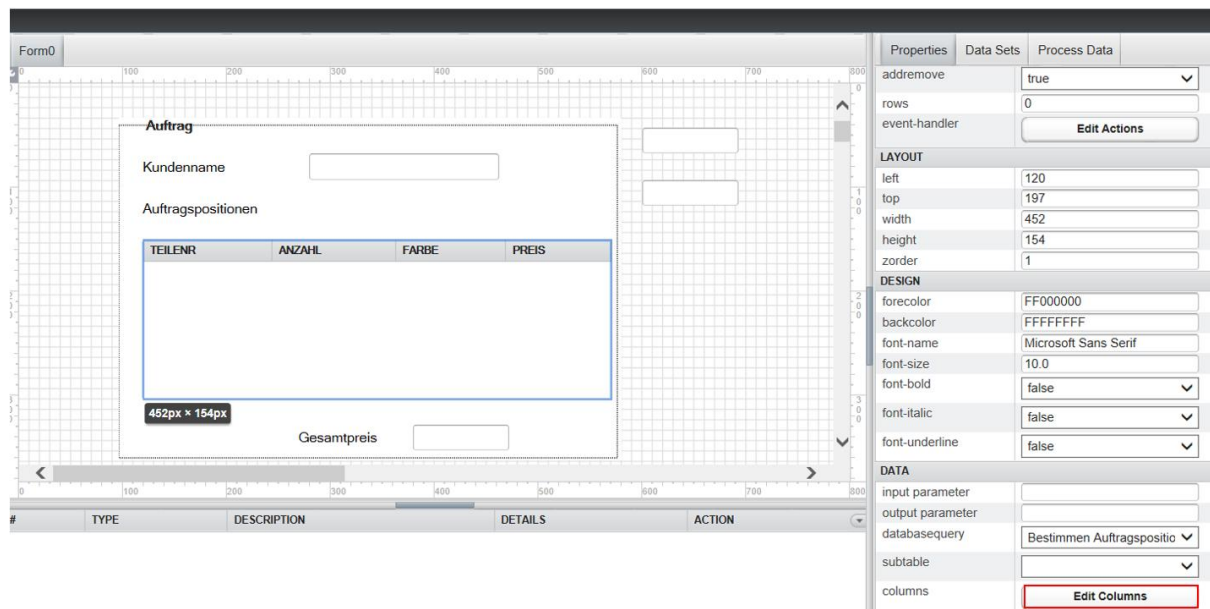


Abbildung 45: Properties der SubtableView subtable mit Columns-Property

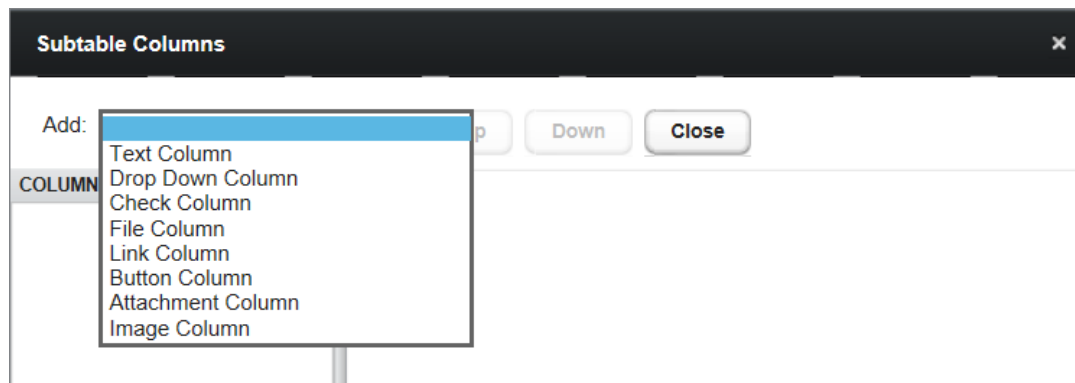


Abbildung 46: Column-Editor

In [Tabelle 3](#) ist beschrieben, was die verschiedenen Typen von Spalten bedeuten. Wählt man den Spaltentyp *Text Column* erhält man das in [Abbildung 47](#) dargestellte *Properties*-Formular, in dem man den Namen dieser Spalte sowie weitere Eigenschaften festlegen kann. Die unter *TextBox*, *Layout*, *Design* und *Data* beschriebenen Eigenschaften haben hierbei dieselbe Bedeutung wie in [Tabelle 2](#) (Abschnitt 4.1 *Elemente einfügen und konfigurieren*) beschrieben.

Text Column	Textspalte, die auch beschrieben werden kann (<i>ReadOnly = false</i>)
Drop Down Column	Spalte mit Dropdown-Menüs, die Werte können hierbei in den Eigenschaften unter <i>Data</i> → <i>Static Items</i> bearbeitet werden
Check Column	Checkboxen als Spalte
File Column	Button mit der Möglichkeit Dateien hochzuladen
Link Column	Link auf eine Web-Ressource
Button Column	Aktionsloser <i>Button</i> , dessen Funktionalität per JavaScript programmiert werden kann
Attachment Column	Heruntergeladene Datei
Image Column	Bettet Bilder in die jeweilige Spalte ein

Tabelle 3: Typen von Spalten für Control *Subtable*

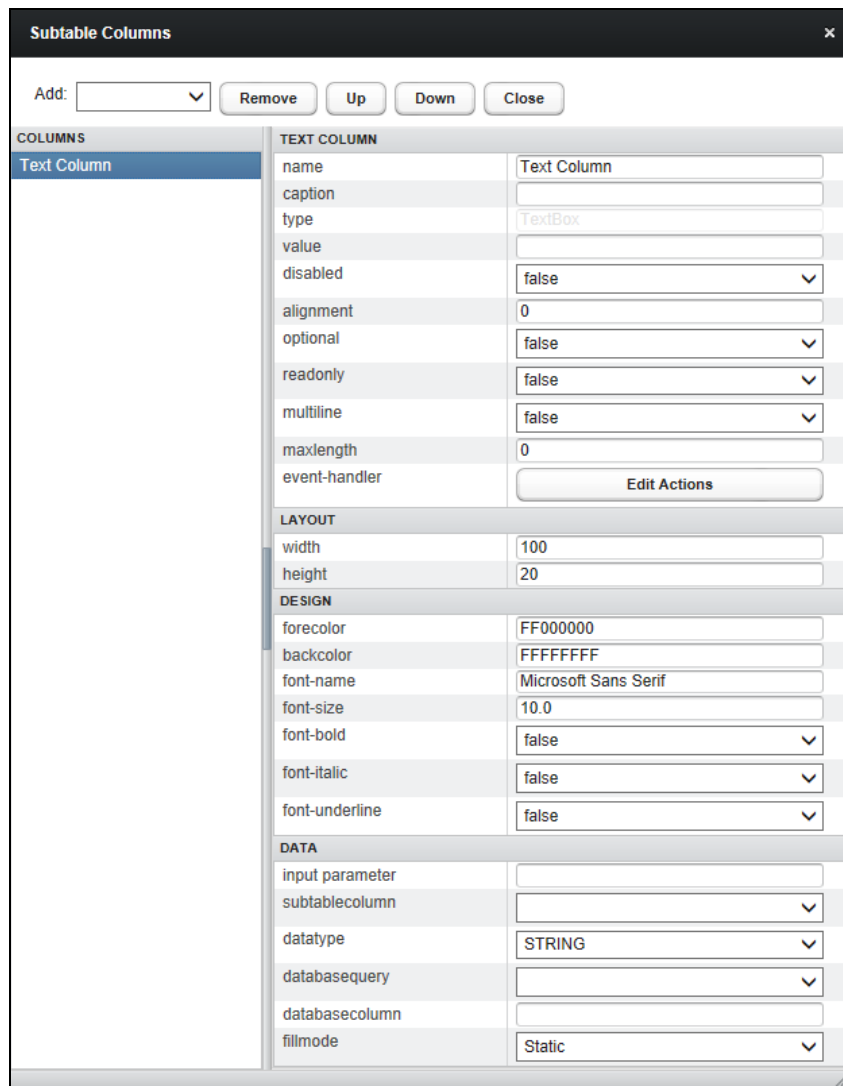


Abbildung 47: Properties des Spaltentyps Text Column

Wird die *Subtable* zur Anzeige einer Resultat-Tabelle einer Datenbankabfrage verwendet, dann verwendet man zur Darstellung der Attributwerte der Ergebnistupel den Spaltentyp *Text Column*. Für alle anzuzeigenden Spalten, der dem *Control* zugeordneten Datenbankabfrage, muss mittels entsprechendem Eintrag im Feld *datatype* eine typgerechte *Text Column* (*INTEGER*, *FLOAT*, *STRING*, ...) angelegt werden. *Name* ist hier (wieder) der (interne) Name des *Controls* und *Caption* dient zur Festlegung der Spaltenüberschrift im Formular (siehe Beispiele in [Abbildung 49](#)).

Möchte man die Zeilen der Tabelle auswählbar machen, so legt man eine zusätzliche *Check Column* an ([Abbildung 48](#)). Auf die hierzu erforderlichen Events und JavaScript-Funktionen gehen wir in Abschnitt 8.4

[Realisierung](#) von Auswahl-Tabellen ein.

Kunde

Kundenliste

	Nummer	Name	Stadt	+
<input type="checkbox"/>	100	Walter	Siegen	-
<input checked="" type="checkbox"/>	105	Aberer	Stuttgart	-
<input type="checkbox"/>	112	Zander	Ulm	-
Number of new Rows:				+

Abbildung 48: Tabelle mit auswählbaren Zeilen mittels Check Column



Wichtige Hinweise:

- Der *FillMode* für jede Spalte muss auf *Database* gesetzt werden, das Feld *DatabaseQuery* der Spalten muss jedoch leer bleiben (sonst wird für jede Zeile der Tabelle die Abfrage neu gestartet und jede Zeile zeigt dasselbe Ergebnis und zwar das erste Ergebnistupel an)
- Die Aktualisierung des Inhalts der *SubtableView* wird mittels assoziierter Datenbankabfrage nur einmalig beim Start des Prozessschritts durchgeführt. Eine „Parametrisierung“ der Datenbankabfrage ist deshalb nur mittels Input-Parameter der Aktivität oder mittels *Controls*, die vorgelegte Werte aufweisen, möglich. – Ein nochmaliges *Reload* der Subtable-View führt zu einem Fehlschlag der WebForm-Aktivität!

DATA	
input parameter	<input type="text"/>
subtablecolumn	<input type="text"/> ▼
datatype	STRING ▼
databasequery	<input type="text"/> ▼
databasecolumn	teileid
fillmode	Database ▼

DATA	
input parameter	<input type="text"/>
subtablecolumn	<input type="text"/> ▼
datatype	INTEGER ▼
databasequery	<input type="text"/> ▼
databasecolumn	anzahl
fillmode	Database ▼

DATA	
input parameter	<input type="text"/>
subtablecolumn	<input type="text"/> ▼
datatype	STRING ▼
databasequery	<input type="text"/> ▼
databasecolumn	preis
fillmode	Database ▼

DATA	
input parameter	<input type="text"/>
subtablecolumn	<input type="text"/> ▼
datatype	INTEGER ▼
databasequery	<input type="text"/> ▼
databasecolumn	farbe
fillmode	Database ▼

Abbildung 49: Data-Properties der Spalten TeileNr, Anzahl, Preis und Farbe

Abbildung 50 zeigt die Ausgabe der Formularaktivität *Auftrag anzeigen* für Auftrag Nr. 15 des Kunden *Zwack* der *LegoTrailer AG*. Sollte die Tabelle nicht ganz angezeigt werden, dann einfach den Wert *Width* in der Kategorie *Layout* des Controls *subtable* etwas größer wählen.

Auftrag

Kundenname

Auftragspositionen

TeileNr	Anzahl	Farbe	Preis
AAG	5	1	479,08
AAG	2	2	526,96
CAH	1	1	311,36

Abbildung 50: Beispiel - Mögliche Ausgabe der Formularaktivität Auftrag anzeigen

Die Einträge in der Subtabelle kann man, falls gewünscht, als änderbar gestalten.

- Änderung von einzelnen Tabellenwerten**
 Mit dem Eintrag *readonly* (*true/false*) auf Ebene des Controls *Subtable* (Abbildung 51 ①) kann man global festlegen, ob Änderungen am Tabelleninhalt prinzipiell möglich sein soll oder nicht. Welche Einträge ggf. tatsächlich änderbar sind, wird auf Spaltenebene entschieden, indem dort das *readonly*-Attribut der Spalte auf *true* oder *false* gesetzt wird.
- Einfügen und Löschen von Zeilen**
 Mit dem Eintrag *addremove* auf Ebene des Controls *Subtable* (Abbildung 51 ②) kann man festlegen, ob zur Laufzeit Zeilen zur Tabelle hinzugefügt oder gelöscht werden können. (Wenn man neue Zeilen einfügen lassen möchte, dann sollten (zumindest einige) Spaltenwerte mit *readonly = false* ausgestattet sein, sonst kann man nur leere Zeilen einfügen).
 Mit diesem Feature kann z.B. eine Datenerfassung in tabellarischer Form realisieren, indem solange neue Zeilen angehängt werden, wie noch Positionen zu erfassen sind.

In [Abbildung 52](#) ist für die Subtable dargestellt, wie die Anzeige der Auftragspositionen aussieht, wenn die *addremove*-Eigenschaft des Controls *subtable* auf *true* gesetzt ist.

Durch Betätigung des -Buttons wird eine neue leere Zeile gefügt (oder auch gleich einen ganzen Block leerer Zeilen). Welche Attribute einer neuen Zeile mit Werten befüllt werden kann, regelt die *Read Only*-Eigenschaft der Spalte, wie vorstehend beschrieben. Durch Betätigung des -Buttons neben einer Zeile wird diese gelöscht.

Tipp:

Da die zusätzliche Spalte zusätzlichen Platz benötigt, sollte der vorgeschlagene Wert im *Width* Parameter des *Subtable*-Controls etwas höher gesetzt werden



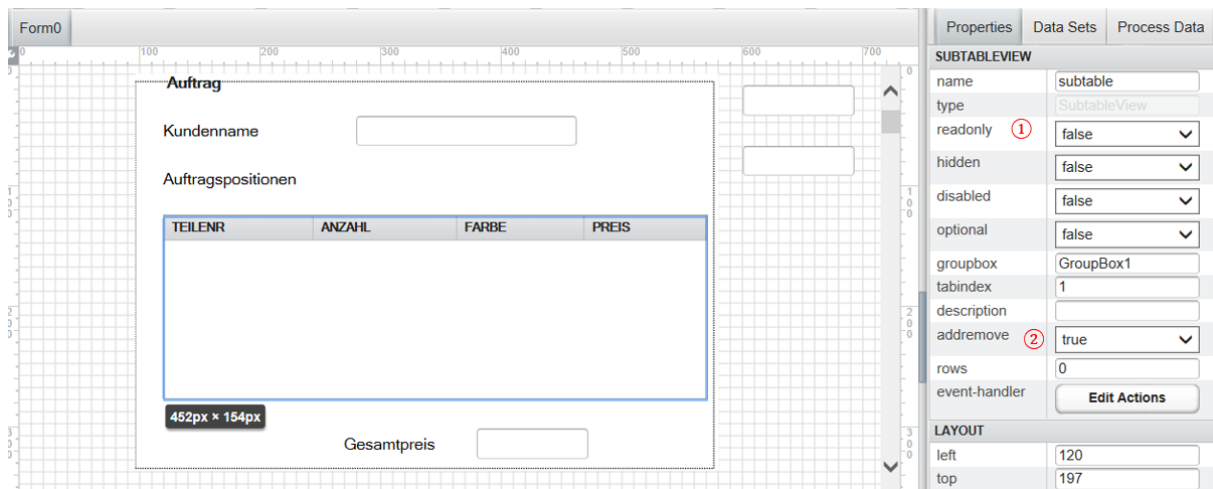


Abbildung 51: Festlegung von readonly- und addremove-Eigenschaften für Subtables

Auftrag

Kundenname

Auftragspositionen

Teilenr	Anzahl	Farbe	Preis	+
AAG	5	1	479.08	-
AAG	2	2	526.96	-
CAH	1	1	311.36	-
				-
Number of new Rows:				1
				+

Abbildung 52: Subtable mit addremove = true Eigenschaft

5.5.2 Realisierung von Output-Subtabellen

Angenommen, wir wollen in unserem Prozessschritt die Auftragspositionen nicht nur anzeigen, sondern auch die Möglichkeit schaffen, die Einträge bei Bedarf ändern zu können, bevor im nächsten Prozessschritt der Auftrag dann bearbeitet wird (*Abbildung 53*).

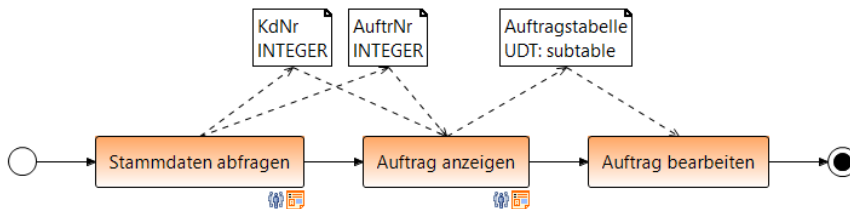


Abbildung 53: Erweiterte Auftragsbearbeitungsprozess

Hinweis:

- Nach der Einbindung einer Aktivität an einen Prozessschritt kann das Datenelement nicht mehr über Änderungsoperationen hinzugefügt werden. Dafür müssen Sie das Datenelement im Assistenten der Aktivität zuordnen. In unserem Beispiel tragen wir die Subtable *Auftragstabelle* als Output Parameter im Assistenten des Prozessschrittes *Auftrag anzeigen* ein.

Hierzu müssen wir eine Datenstruktur bereitstellen, welche ganz (oder in Auszügen) die angezeigte Subtabelle auf eine Datenstruktur abbildet, die bei Abschluss der Aktivität als (XML-Strukturierer) Ausgabeparameter vom Typ **UDT:subtable** zurückgegeben wird.

Zur Erzeugung unserer „Subtable-Datenstruktur“ wählen wir im *Edit*-Menü → *Subtables ...* (*Abbildung 54*) aus, worauf sich das „Subtables-Fenster“ öffnet (*Abbildung 55*) und uns das Anlegen einer Subtable-Datenstruktur ermöglicht, welcher wir im Beispiel den Namen *auftragspos_subtable* gegeben haben.

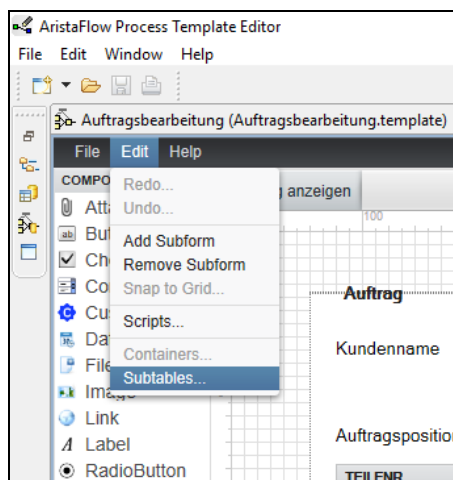


Abbildung 54: Erzeugen der Subtable-Datenstruktur (1)

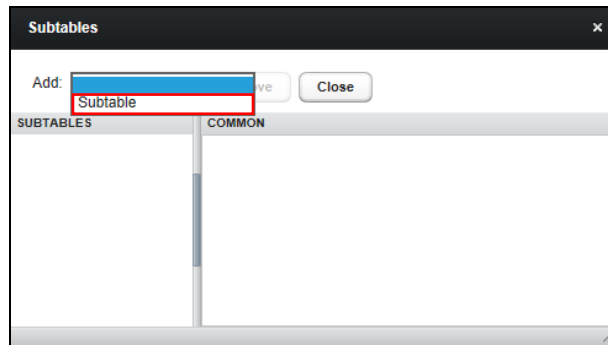


Abbildung 55: Erzeugen der Subtable-Datenstruktur (2)

Als nächstes legen wir für alle Spalten unser Formular-Subtabelle, die wir mittels unserer Subtable Datenstruktur später nach außen geben wollen, ebenfalls Spalten an ([Abbildung 56](#)), und zwar entsprechend deren gewünschtem Ausgabedatentyp (*STRING*, *INTEGER*, *FLOAT*, ...) in der XML-Struktur.

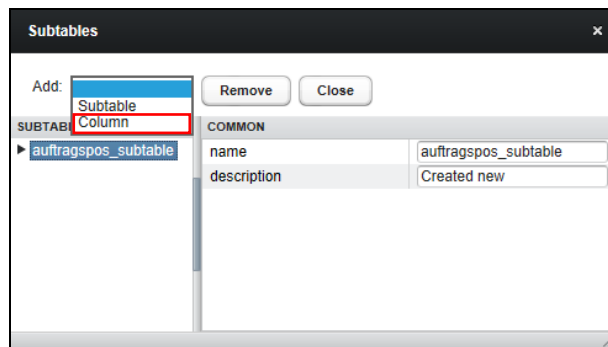


Abbildung 56: Erzeugen der Subtable-Datenstruktur (3)

Wenn wir in unserem Beispiel in diesem Sinne alle Spalten „spiegeln“ möchten, könnten die Einträge wie in [Abbildung 57](#) dargestellt aussehen. Die Spaltennamen in der Subtable-Datenstruktur müssen hierbei nicht identisch mit den Spaltennamen in der Formulartabelle sein. Diese Abbildung wird explizit festgelegt (siehe unten) und nicht etwa „by name“.

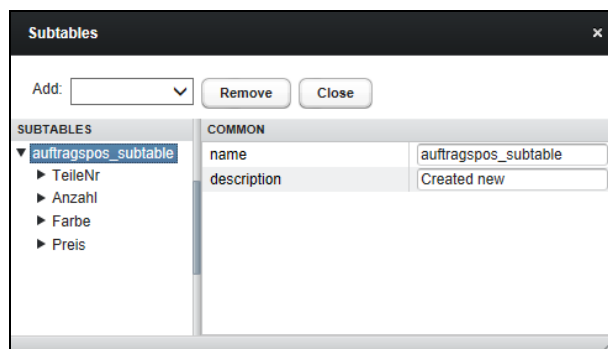


Abbildung 57: Erzeugen der Subtable-Datenstruktur (4)

Nachdem wir die Subtable-Datenstruktur angelegt haben, verknüpfen wir sie mit der Formular-Tabelle, indem wir in den Daten-*Properties* der Formular-Tabelle im Feld *Subtable* die dort nunmehr zur Auswahl angebotene Subtable-Datenstruktur auswählen ([Abbildung 58](#)).

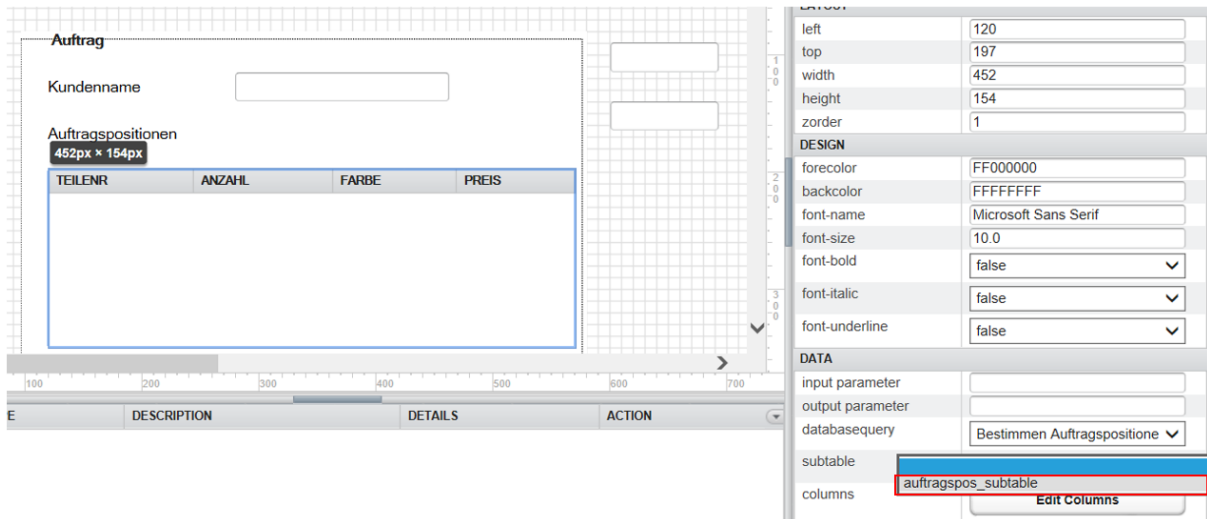


Abbildung 58: Verknüpfung der Formular-Tabelle mit der Subtable-Datenstruktur (1)

Anschließend öffnen wir in der *SubtableView* der Formular-Tabelle durch Anklicken des Buttons *Edit Columns* die *Columns*-Auflistung (Abbildung 59) und ordnen dort nun jeder „gespiegelten“ Formulareispekte die entsprechende Spalte der Subtable-Datenstruktur zu (Abbildung 60).

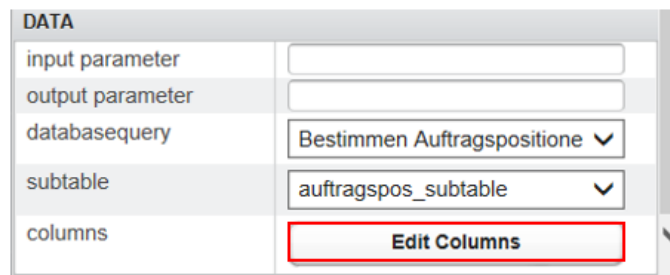


Abbildung 59: Verknüpfung der Formular-Tabelle mit der Subtable-Datenstruktur (2)

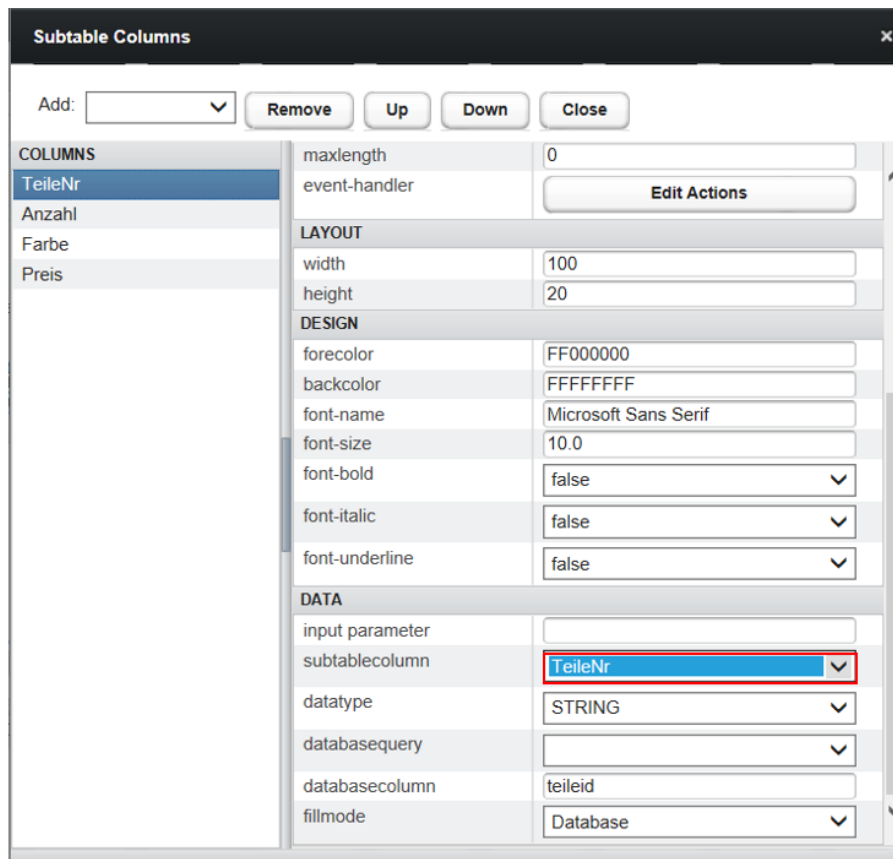


Abbildung 60: Verknüpfung der Formular-Tabelle mit der Subtable-Datenstruktur (3)

Die Verknüpfung unserer Subtable-Datenstruktur mit dem Subtable-Ausgabeparameter stellen wir dadurch her, dass wir entweder (in gewohnter Weise) die Process Data View öffnen und den Ausgabeparameter auf die Formular-Tabelle ziehen (mit der unsere Subtable-Datenstruktur ja intern verknüpft ist) ([Abbildung 61](#)) oder in dem wir in der Properties-Ansicht im Feld Output-Parameter den Namen des Ausgabeparameters eintragen ([Abbildung 62](#)). Der Ausgabeparameter muss vom Typ *UDT:subtable* sein.

Auf die Verarbeitung des Inhalts dieses Ausgabe-Parameters in einem nachfolgenden Prozessschritt gehen wir in Kapitel 7 [Umgang mit Parametern vom Typ USERDEFINED subtable](#) ein.

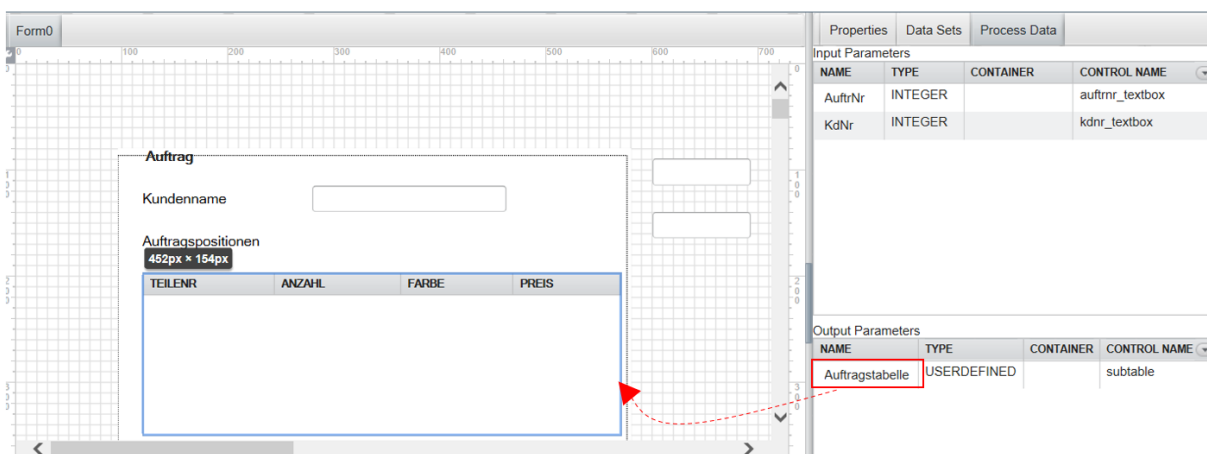


Abbildung 61: Verknüpfung der Tabelle mit einem Ausgabeparameter (1)

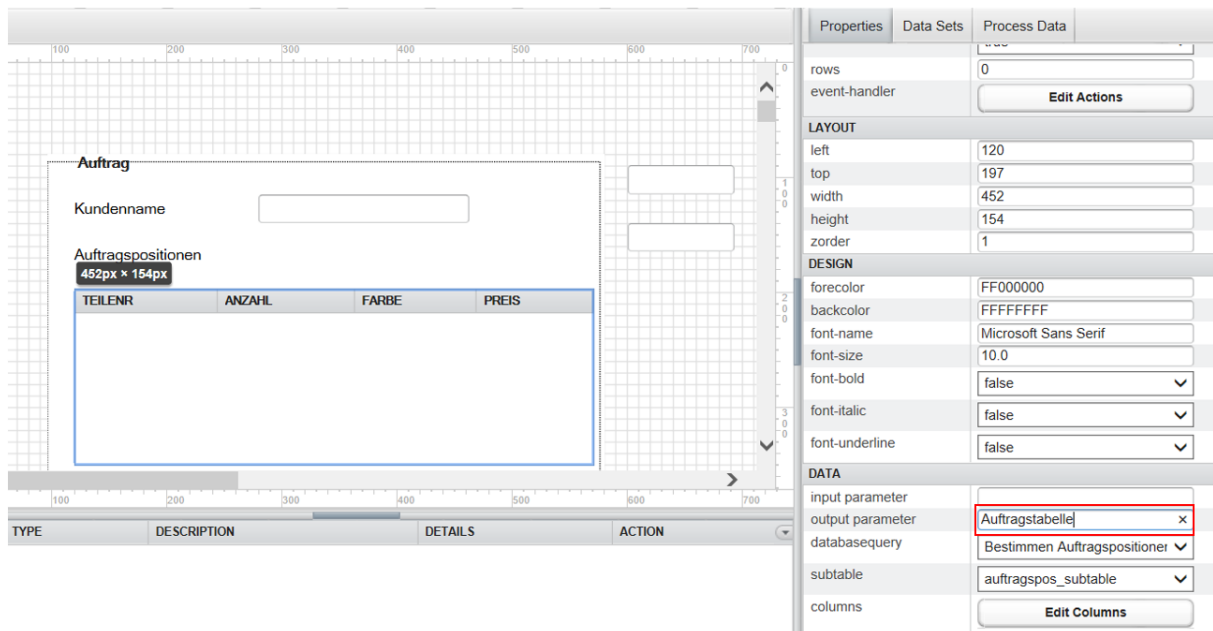


Abbildung 62: Verknüpfung der Tabelle mit einem Ausgabeparameter (2)

6 Implementierung von Formularfunktionen und zusätzlicher Logik

Dieses Kapitel beschreibt die Möglichkeiten, Formulare mit Hilfe von *Events* und *JavaScript* mit erweiterten Funktionen auszustatten, etwa um Eingaben zu überprüfen oder Berechnungen auszuführen.

1.1 Events und Aktionen für ein Control definieren

In Abschnitt 5.4 *Ereignisgesteuerte DB-Anfragen-Ausführung zur Aktualisierung von Controls* haben wir am Beispiel von *Controls*, die mit Datenbankabfragen verbunden sind, bereits kennen gelernt, wie man eine (erneute) Ausführung der Abfrage veranlassen kann. Man hinterlegt beim auslösenden Control unter *Event Handlers* den gewünschten *Eventtyp* und trägt bei dessen *ActionType* und *Script* ein, welche Aktion auf welchen Objekten ausgeführt werden soll.

Im Formular für *Stammdaten abfragen* (Abbildung 63) sollte z.B. die Auswahl eines Kunden in der Combobox *kunde_combobox* eine Aktualisierung der von dem gewählten Wert abhängigen Controls auslösen. Wir haben deshalb das *onchange*-Ereignis gewählt (Abbildung 64 ①). Die „abhängigen“ Controls sind in diesem Fall *kname_textbox*, *kdstadt_textbox*, *kdbonitaet_textbox* und *auftrnr_combobox* (Abbildung 64 ②).

„Aktualisierung“ bedeutet bei allen diesen Controls, dass die mit ihnen assoziierte Datenbankabfrage erneut ausgeführt werden soll. Da (wie bereits in Abschnitt 5.4 *Ereignisgesteuerte DB-Anfragen-Ausführung zur Aktualisierung von Controls* erwähnt), Datenbankabfragen (eigentlich) beim Laden (*Load*-Ereignis) des Formulars ausgeführt werden, wird ein *Reload*-Event für diese Controls erzeugt (Abbildung 64 ③).

The image shows a web form with two main sections: 'Kunde' and 'Auftrag'. The 'Kunde' section is enclosed in a dashed box and contains a dropdown menu labeled 'Kunde auswählen' and three text input boxes labeled 'Kundenname', 'Stadt', and 'Bonität'. The 'Auftrag' section is also enclosed in a dashed box and contains a dropdown menu labeled 'Auftragsnummer'. Red arrows on the right side of the form point to each of these controls, with labels: 'kunde_groupbox' points to the 'Kunde' section, 'kunde_combobox' points to the 'Kunde auswählen' dropdown, 'kname_textbox' points to the 'Kundenname' text box, 'kdstadt_textbox' points to the 'Stadt' text box, 'kdbonitaet_textbox' points to the 'Bonität' text box, 'auftrag_groupbox' points to the 'Auftrag' section, and 'auftrnr_combobox' points to the 'Auftragsnummer' dropdown.

Abbildung 63: Formular für Stammdaten abfragen mit Namen des Controls

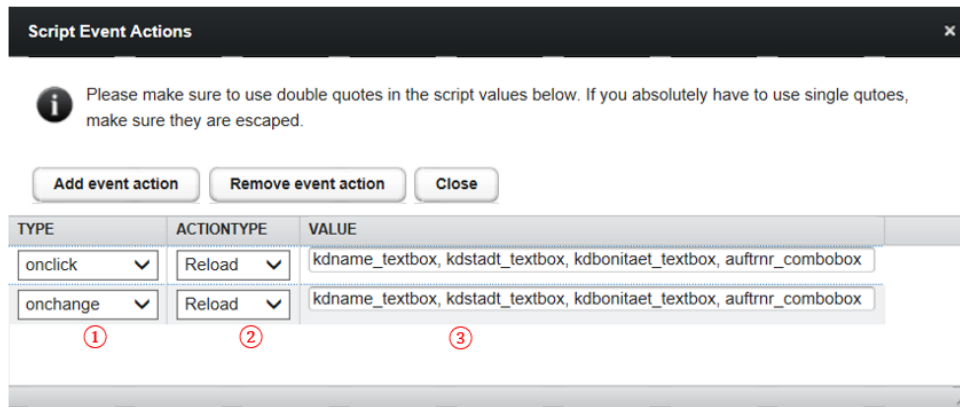


Abbildung 64: Eventhandler für Control kunde_combobox

Der WebForm-Designer kennt folgende Ereignistypen:

onclick	Event nach Anklicken des Controls
ondblclick	Event nach zweimaligem Klicken in das entsprechende Control
onchange	Event nach Änderung des Inhalts
onfocus	Event nach Auswählen des Controls (z.B. per Tabulator oder Maus)
onblur	Event nach Verlassen des Controls

Tabelle 4: Typen von Ereignissen (Events)

Wenn ein Event geworfen wird, lässt sich mittels Action Type *Reload* entweder ein anderes Control erneut laden (und dann werden ggf. dessen registrierte Events ausgelöst) oder es kann mittels Action Type *Script* ein Script zur Ausführung gebracht werden. Soll ein *Eventtyp* (z.B. *onchange*) sowohl *Reload*- als auch *Script*-Events auslösen, dann fügt man das entsprechende Ereignis im Eventhandler mittels Menüpunkt *Add* einfach mehrfach ein, um die unterschiedlichen Aktionen hinterlegen zu können.

6.1 Erstellung von JavaScript-Funktionen

Die empfohlene Vorgehensweise eine JavaScript-Funktion zu erstellen und es einem Control-Ereignis zuzuordnen ist wie folgt: Man spezifiziert im zentralen JavaScript-Fenster des Formulars ([Abbildung 65](#); erreichbar via Menü *Edit* → *Scripts...*) die gewünschte Funktion und ruft diese dann im *Eventhandler* des Controls auf. Hierdurch stehen alle JavaScript-Routinen des Formulars an einer zentralen Stelle und man vermeidet zudem redundanten Code, wenn eine Funktion von mehr als einem Control gerufen werden muss (weitere Details hierzu in Abschnitt 6.2 [Verknüpfung von JavaScript-Funktionen mit Events und Controls](#))

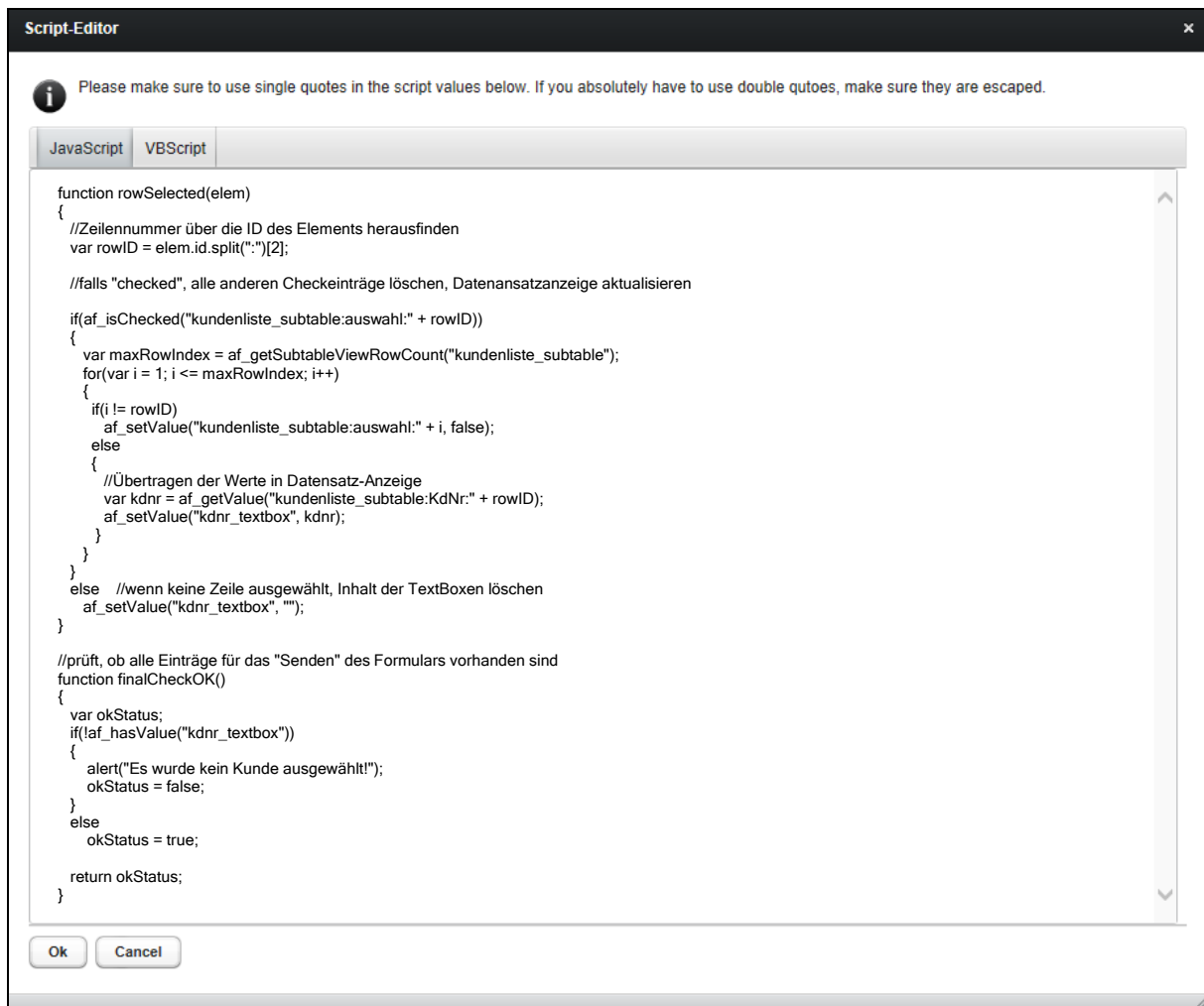


Abbildung 65: Zentrales JavaScript-Fenster

JavaScript ist eine an Java angelehnte Programmiersprache, die in HTML-Seiten eingebettet werden kann. Eine übersichtliche Darstellung der JavaScript-Sprachelemente finden Sie in zahlreichen Tutorials im Internet. (Eine gute Online-Quelle für JavaScript ist <http://de.selfhtml.org/javascript/>.)

Die Controls der WebForm-Formulare bieten eine Reihe von Methoden an, die man in JavaScript-Programmen ansprechen kann. So kann man z.B.

- Den Wert eines Controls abfragen oder setzen
- Ein Control sichtbar machen oder verstecken
- Ein Control auf *read only* setzen oder dies wieder aufheben
- ...

Die Beschreibung dieser Methoden ([Abbildung 66](#) und [Abbildung 67](#)) befindet sich im Menü des WebForm Designers unter *Help* → *Show API documentation*.

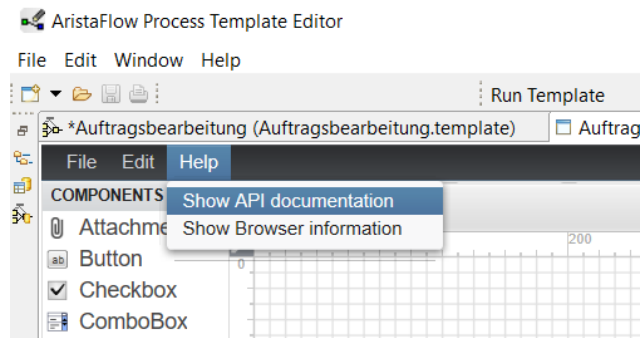


Abbildung 66: WebForm API Dokumentation (1)

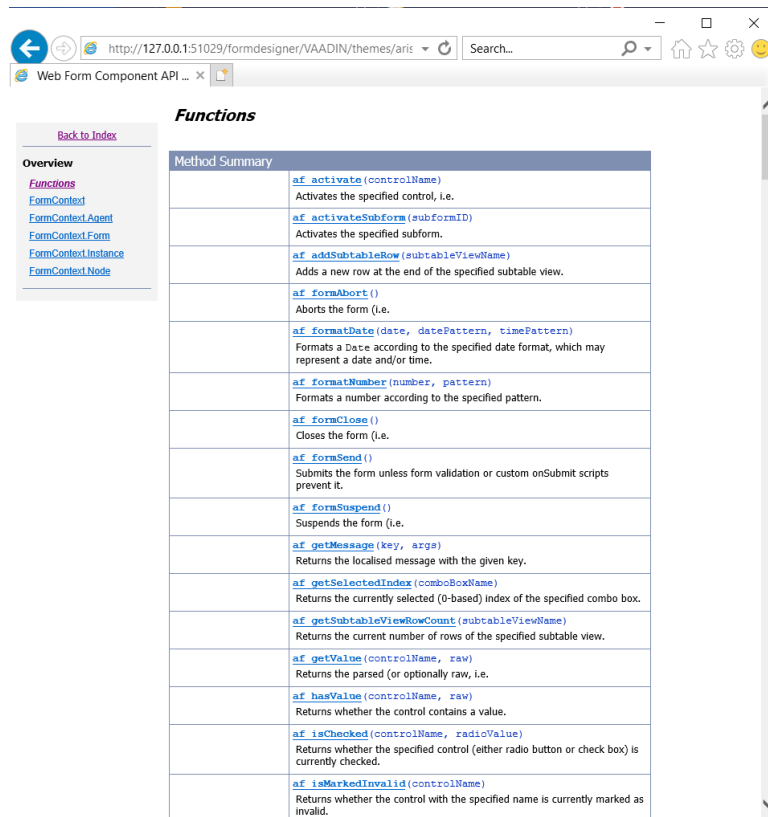


Abbildung 67: WebForm API Dokumentation (2)

Bevor wir in Kapitel 8 *Anwendungsbeispiele* einige Anwendungsbeispiele vorstellen, erläutern wir in Abschnitt 6.2 *Verknüpfung von JavaScript-Funktionen mit Events und Controls* zunächst den Zusammenhang von Controls, Events und JavaScript-Programmen und wie dieser hergestellt wird.

6.2 Verknüpfung von JavaScript-Funktionen mit Events und Controls

Mittels in JavaScript-Routinen eingebettete WebForm-API-Funktionen kann man den Wert von Controls ermitteln oder setzen, kann diese ein- oder ausblenden, aktivieren und deaktivieren, auf *read only* setzen und vieles mehr. Der Anstoß zum Ausführen solcher Routinen erfolgt stets durch ein „Ereignis“ (*Event*). D.h. wir definieren ein entsprechendes Ereignis und hinterlegen dort, welche unserer JavaScript-Funktionen gerufen werden soll.

Wir illustrieren die Vorgehensweise anhand eines Beispiels. Wir ergänzen hierzu das Formular *Auftrag anzeigen* um ein weiteres Control mit dem internen Namen *gpreis_textbox*, welches zur Ausführungszeit den Gesamtwert der Auftragspositionen anzeigt.

Auftrag

Kundenname

Auftragspositionen

Teilenr	Anzahl	Farbe	Preis	+
AAG	5	1	479.08	-
AAG	2	2	526.96	-
CAH	1	1	311.36	-
Number of new Rows: <input type="text"/>				+

Gesamtpreis

Abbildung 68: Formular Auftrag anzeigen mit zusätzlichem Summenfeld

Zur Berechnung der Summe in der Spalte *Preis* definieren wir im JavaScript-Fenster unseres Formulars eine Funktion *berechnePreis()* (Abbildung 69 ①). Diese soll immer dann aufgerufen werden, wenn sich ein Eintrag in der Spalte *Preis* ändert, um den Wert des *gpreis_textbox*-Controls neu zu berechnen. (Die inhaltlichen Details dieser Funktion sind im Moment nicht von Belang; auf diese gehen wir in Kapitel 8 *Anwendungsbeispiele* ein.)

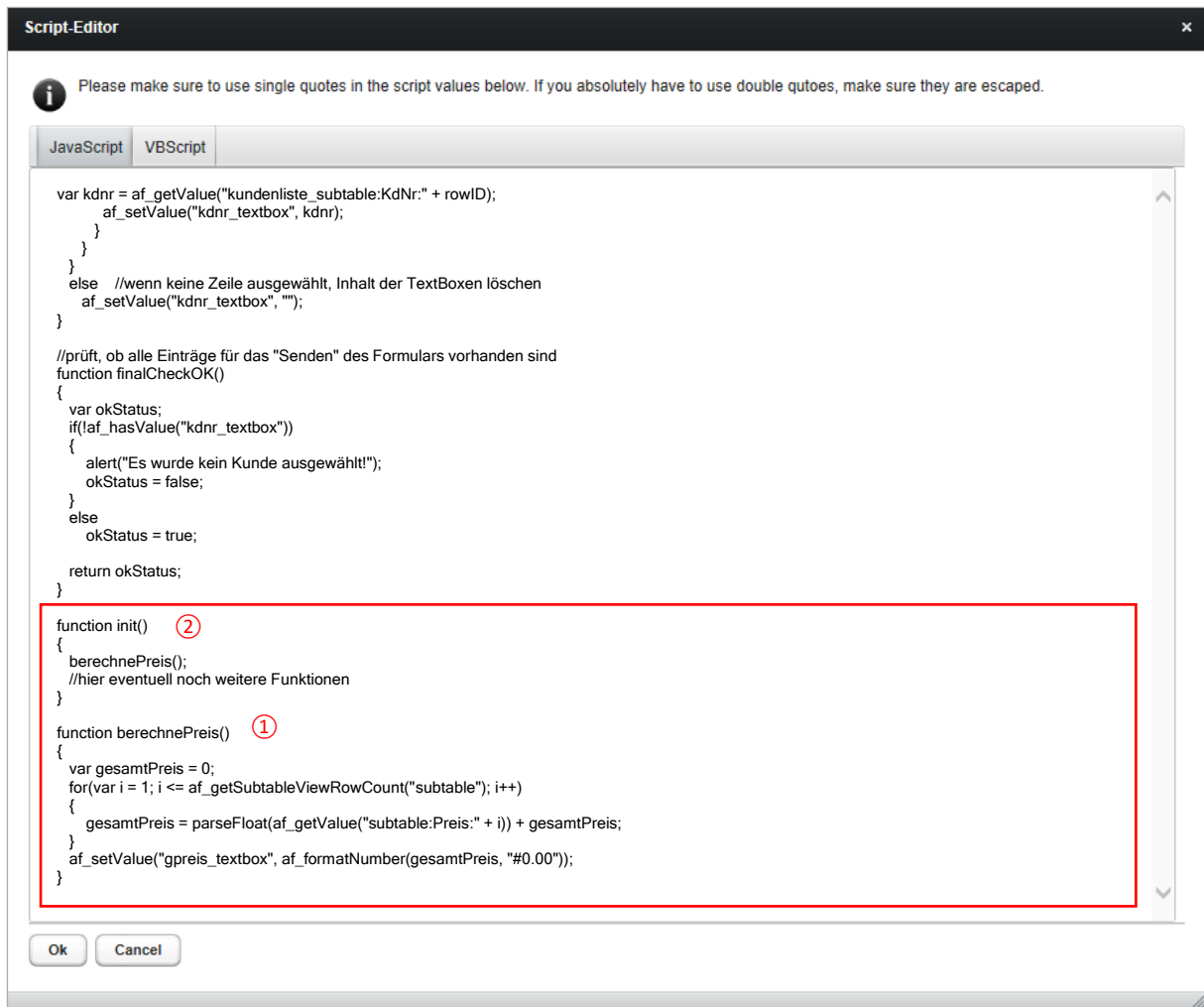


Abbildung 69: Hinterlegung von JavaScript-Funktionen im JavaScript-Fenster

Um die **fortlaufende Neuberechnung** auszulösen, definieren wir in der Subtabelle im Control *Preis* ein *onchange*-Ereignis, welches bei jeder Wertänderung (z.B. wegen einer neuen Zeile) einen Aufruf der Funktion *berechnePreis()* Funktion ausführt. [Abbildung 70](#) illustriert den Ablauf:

1. Man markiert die *subtable* ①
2. Wählt in deren *Properties* ②
3. Mittels *Columns* das Control *Preis* ③
4. Und in dessen *Properties* dessen *event-handlers* aus. ④
5. Dort fügt man ein *onchange* Ereignis ein ⑤
6. Wählt *ActionType Script* ⑥ und
7. Hinterlegt im Script-Feld ⑦ den Funktionsaufruf (mit Klammer und Semikolon!)

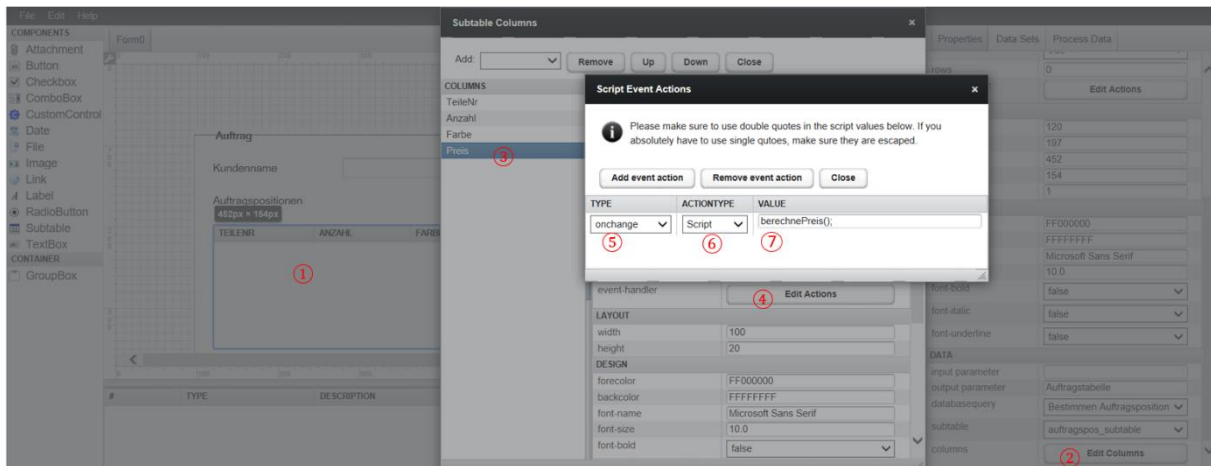


Abbildung 70: Aufruf der JavaScript-Funktion `berechnePreis()` im EventHandler des Controls `Preis`

6.3 Handhabung der „Formular-Ereignisse“ `onload`, `submit` und `onsuspend`

Gelegentlich will man beim Laden einer Formularaktivität gewisse Dinge ausführen, wie z.B. das Ausblenden von Controls und Subformularen, das Berechnen von Werten oder das Vorbelegen von Controls mit initialen Werten. Zwar kann man dies meist auch mittels der Properties der Controls tun, doch dann sind diese Aktionen jedoch quasi im ganzen Formular verstreut. Aus Dokumentations- und Wartungsgründen ist es deshalb besser, diese Dinge in eine Initialisierungsfunktion im zentralen JavaScript-Fenster zu packen und diese beim Laden des Formulars ausführen zu lassen.

Die Bearbeitung von vielen interaktiven Prozessschritten (so auch WebForm-Formularschritte) kann unterbrochen *Suspend Activity* und zu einem späteren Zeitpunkt wieder fortgesetzt werden *Resume Activity*. Bei einem solchen *Suspend* werden zwar eventuelle Formular-Eingaben gespeichert, nicht aber die Werte berechneter Felder. Wenn man nach einem *Resume* das Formular automatisch wieder in dem Zustand sehen soll, in dem verlassen wurde, muss man beim Laden des Formulars etwaige Berechnungen zum Befüllen von Controls (nochmals) auslösen. Hierzu spezifiziert man beim EventHandler des Formulars (auf der „Hauptebene“) ein *onload*-Ereignis und hinterlegt dort den Aufruf einer entsprechenden Funktion, z.B. *init()*, in der dann auszuführenden Einzelfunktionen gerufen werden.

In dem in [Abbildung 71](#) dargestellten Formular soll nach dessen in der dargestellten TextBox mit dem internen Namen `gpreis_textbox` die Spaltensumme von `Preis` angezeigt werden. Für die Berechnung des Wertes existiere eine JavaScript-Funktion `berechnePreis` (siehe [Abbildung 72](#) ①). Diese Funktion soll einerseits bei einer Wertänderung in der Spalte `Preis` andererseits aber auch beim Laden des Formulars aufgerufen werden.

Auftrag

Kundenname

Auftragspositionen

TeileNr	Anzahl	Farbe	Preis
AAG	5	1	479,08
AAG	2	2	526,96
CAH	1	1	311,36

Gesamtpreis

Abbildung 71: Formular mit berechnetem Wert

```

function init() ②
{
    berechnePreis();
    //hier eventuell noch weitere Funktionen
}

function berechnePreis() ①
{
    var gesamtPreis = 0;
    for(var i = 1; i <= af_getSubtableViewRowCount("subtable"); i++)
    {
        gesamtPreis = parseFloat(af_getValue("subtable:Preis" + i)) + gesamtPreis;
    }
    af_setValue("gpreis_textbox", af_formatNumber(gesamtPreis, "#0.00"));
}

```

Ok Cancel

Abbildung 72: Realisierung einer Initialisierungsfunktion

Für die Neuberechnung nach Wertänderung hinterlegen wir im Eventhandler der Spalte *Preis* ein *onchange*-Ereignis, welches die Funktion *berechnePreis()* aufruft.

Für die Neuberechnung nach Laden definieren wir auf Formularebene je ein *onload*-Ereignis und rufen dort die in [Abbildung 72](#) ② dargestellte Funktion *init()*.

[Abbildung 73](#) illustriert den Ablauf, um diese Events auf Formularebene zu definieren und ihnen *die init()*-Funktion zuzuordnen.:

1. Man klickt außerhalb eines Controls in das Formular ① worauf in der Properties-Ansicht *Worksheet* angezeigt wird.
2. Dort klickt man auf *Edit Actions* ② und öffnet *Script Event Actions*
3. Im *Script Event Actions* -Fenster fügt man ein *Event Action* ③ hinzu,
4. Indem man zuerst den Typ *onload*-Event ④
5. Und *ActionType Script* ⑤ wählt.
6. Unter *Value* hinterlegt man den *init()* ⑥-Aufruf

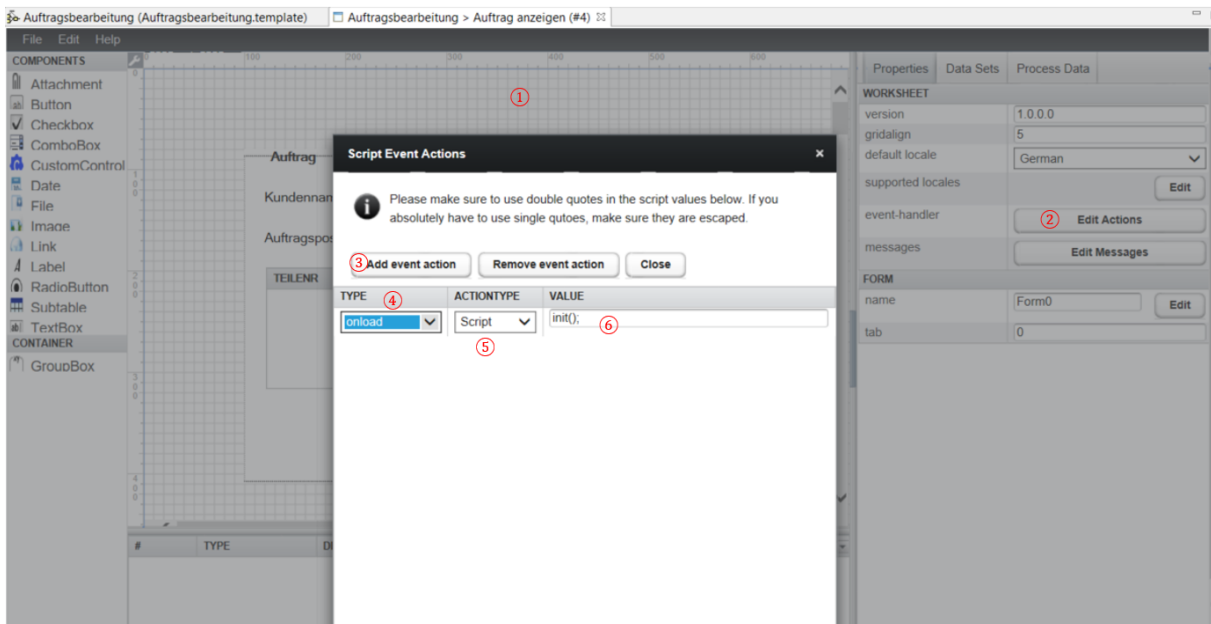


Abbildung 73: Aufruf der Funktion init() im EventHandlerer des Formulars

6.4 Arbeiten mit Subformularen

Wie bereits in Abschnitt 4.4 *Mehr-Blatt-Formulare* erwähnt, besteht die Möglichkeit, einen Formularschritt als „Mehrblatt-Formular“ zu realisieren, indem man im Edit-Menü *Add* → *Subform* wählt und in dessen Properties die Beschriftung des Reiters festlegt. Auf diese Weise könnte man z.B. in einem Subformular *Anlegen Auftrag* einen Auftrag anlegen, in dem man dort den Kunden und das Auftragsdatum festlegt (Abbildung 74) und in einem anderen Subformular *Erfassen Auftragspositionen* die Auftragspositionen erfassen (Abbildung 75).

Per Default werden alle Subformulare angezeigt und man kann diese durch Anklicken des entsprechenden Reiters aktivieren, wie in *Abbildung 74* und *Abbildung 75* angedeutet.

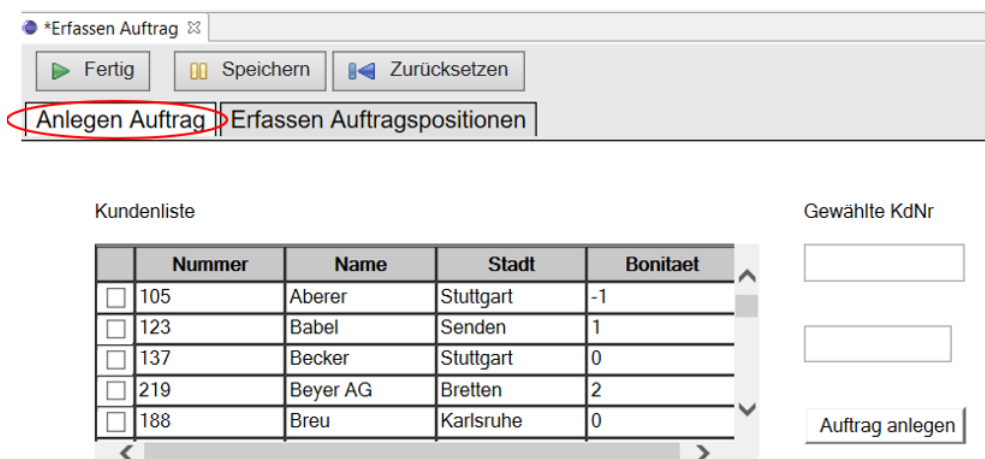


Abbildung 74: Schritt Erfassen Auftrag- Subformular Anlegen Auftrag

*Erfassen Auftrag

Fertig Speichern Zurücksetzen

Anlegen Auftrag Erfassen Auftragspositionen

Kunde: Auftragsdatum:

Teilekatalog:

Auswahl	TeileNr	Farbe	Bezeichnung	Preis
<input type="checkbox"/>	AAG	1	Aufbauanhänger	479.08
<input type="checkbox"/>	AAG	2	Aufbauanhänger	526.96
<input type="checkbox"/>	AKK	1	Kippanhänger ku	318.08
<input type="checkbox"/>	AKK	2	Kippanhänger ku	472.08
<input type="checkbox"/>	AKL	1	Kippanhänger lai	519.26
<input type="checkbox"/>	AKL	2	Kippanhänger lai	540.54
<input type="checkbox"/>	CAH	1	Chassis Aufbau	311.36
<input type="checkbox"/>	CAH	2	Chassis Aufbau	342.44

TeileNr	Farbe	Bezeichnung	Preis	Anzahl
---------	-------	-------------	-------	--------

Abbildung 75: Schritt Erfassen Auftrag –Subformular Erfassen Auftragspositionen

Will man die Subformulare gezielt einblenden, etwa dass man erst nach vollständigem Ausfüllen des Subformulars *Anlegen Auftrag* (Abbildung 76) das Subformular *Erfassen Auftragspositionen* (Abbildung 77) bearbeiten können soll, so kann man dies mittels der Funktionen `af_setSubformHidden(subformID, hidden)` und `af_activateSubform(subformID)` realisieren.

*Auftrag erfassen

Fertig Speichern Zurücksetzen

Anlegen Auftrag

Kundenliste

	Nummer	Name	Stadt	Bonitaet
<input type="checkbox"/>	105	Aberer	Stuttgart	-1
<input checked="" type="checkbox"/>	123	Babel	Senden	1
<input type="checkbox"/>	137	Becker	Stuttgart	0
<input type="checkbox"/>	219	Beyer AG	Bretten	2
<input type="checkbox"/>	188	Breu	Karlsruhe	0

Gewählte KdNr

Abbildung 76: Erfassen Auftrag- erstes Subformular aktiv

*Auftrag erfassen

Fertig Speichern Zurücksetzen

Erfassen Auftragspositionen

Kunde: Auftragsdatum:

Teilekatalog:

Auswahl	TeileNr	Farbe	Bezeichnung	Preis
<input type="checkbox"/>	AAG	1	Aufbauanhänger geschlossen	479.08
<input type="checkbox"/>	AAG	2	Aufbauanhänger geschlossen	526.96
<input type="checkbox"/>	AKK	1	Kippanhänger kurz	318.08
<input type="checkbox"/>	AKK	2	Kippanhänger kurz	472.08
<input type="checkbox"/>	AKL	1	Kippanhänger lang	519.26
<input type="checkbox"/>	AKL	2	Kippanhänger lang	540.54
<input type="checkbox"/>	CAH	1	Chassis Aufbauanhänger	311.36
<input type="checkbox"/>	CAH	2	Chassis Aufbauanhänger	342.44

TeileNr	Farbe	Bezeichnung	Preis	Anzahl
---------	-------	-------------	-------	--------

Abbildung 77: Erfassen Auftrag – zweites Subformular aktiv

Hierzu wird auf Formularebene ein *onload*-Ereignis definiert und dort die Funktion *init()* hinterlegt, die dafür sorgt, dass das Subformular *Erfassen Auftragspositionen* ausgeblendet und der Button *auftrag_anlegen_button* auf *disabled* gesetzt wird.

```
function init()
{
  // Hide Erfassen Auftragspositionen + disable "Auftrag anlegen"
  af_setSubformHidden("Erfassen Auftragspositionen", true);
  af_setDisabled("auftrag_anlegen_button", true);
}
```

Abbildung 78: Erfassen Auftrag – zweites Subformular aktiv

Der Button *Auftrag anlegen* wird *enabled*, wenn die Felder für Kundennummer und Auftragsdatum mit Werten versorgt wurden. Hierfür wurde bei beiden Feldern ein *onchange*-Ereignisse definiert, welche die Funktion *checkEnableButton()* (siehe [Abbildung 79](#)) rufen, die ggf. den Button „freischaltet“.

```
function checkEnableButton()
{
  // Prüft, ob Kundennummer und Auftragsdatum gesetzt sind
  // und schaltet ggf. den Button ‚Auftrag anlegen‘ frei
  if (af_hasValue("kdnr_textbox") && af_hasValue("auftragsdatum_date"))
    af_setDisabled("auftrag_anlegen_button", false);
  else
    af_setDisabled("auftrag_anlegen_button", true);
}
```

Abbildung 79: Auftrag anlegen button freischalten oder sperren

Das „Umschalten“ von Subformular *Auftrag anlegen* auf das Formular *Erfassen Auftragspositionen* erfolgt durch die Funktion *validateSubform0()*, die beim *onclick*-Ereignis des Buttons *auftrag_anlegen_button* hinterlegt ist.

```
function validateSubform0()
{
  // Prüfung, ob Subform0 vollständig ausgefüllt wurde.
  // Falls ja, umschalten auf Subform1
  if (af_hasValue("kdnr_textbox") && af_hasValue("auftragsdatum_date"))
  {
    af_setSubformHidden("Anlegen Auftrag", true);
    af_setSubformHidden("Erfassen Auftragspositionen", false);
    af_activateSubform("Erfassen Auftragspositionen");
  }
  else
    alert("Auftragserfassung ist unvollständig!");
}
```

Abbildung 80: Umschalten von einem Subformular auf ein anderes

7 Umgang mit Parametern vom Typ *USERDEFINED subtable*

7.1 Allgemeines zu erweiterten AristaFlow-Datentypen

AristaFlow verfügt über eine Reihe vordefinierter „erweiterter Datentypen“, um bei Bedarf auch mit nichtelementaren Datentypen als Eingabe- oder Ausgabeparameter umgehen zu können. Sie gehören zur Typ Kategorie *USERDEFINED* und sind jeweils als Java-Klassen realisiert.

Die Beschreibung dieser Java-Klassen mit ihren Klassen- und Objektmethoden findet sich in jedem AristaFlow BPM Suite Release unter dem Namen *aristaflow-javadoc-all*. Sie kann über den Download-Server der AristaFlow GmbH als Online Dokumentation eingesehen werden. Z.B. <https://pubweb.aristaflow.com/builds/Releases/6.1.0/aristaflow-javadoc-all/>

Der Typ *USERDEFINED subtable* gehört zu diesen erweiterten AristaFlow-Datentypen. Seine Beschreibung findet sich unter Class *WebFormSubtable* und kann über die Packages *de/aristaflow/adept2/extensions/datatypes/* erreicht werden.

7.2 Erzeugung von Parameter-Objekten vom Typ *USERDEFINED subtable*

Die Möglichkeit, ein solches Parameterobjekt als Ausgabeparameter vom WebForm -Designer erzeugen zu lassen, haben wir bereits kennen gelernt (siehe z.B. Abschnitt 5.5.2 *Realisierung von Output-Subtabellen*). Man kann ein solches Objekt aber auch selbst erzeugen. In Class *WebFormSubtable* wird beschrieben, wie man

- mittels Konstruktor ein neues Objekt erzeugt (*new WebFormSubtable()*)
- Spalten hinzufügt (*addColumn(...)*) und löscht (*removeColumn(...)*)
- Zeilen einfügt (*addRow()*) und löscht (*removeRow(...)*)
- einzelne Zellen oder eine gesamte Zeile mit Werten versorgt (*setValue(...)*, *setValues(...)*)

Ein solches Objekt könnte z.B. von einer Scripting-Aktivität *executeScript* oder einem anderen Java-Programm erzeugt und als Ausgabeparameter ausgegeben werden.

Der in *Abbildung 81* dargestellte Beispielprozess erzeugt im ersten Prozessschritt per Ausgabeparameter ein Ausgabeobjekt vom Typ *UDT: subtable*, das dem zweiten Schritt als Eingabeparameter dient und dort angezeigt wird.

Die Formulardeklaration ist in *Abbildung 82* dargestellt. Wie man dort sieht, ist die *SubtableView* sowohl mit dem Eingabeparameter namens *subtable* verknüpft und als auch intern mit der Subtable-Datenstruktur namens *test* assoziiert. Die Datentypen der test-Struktur sind – wie die Namen schon andeuten – vom Typ *Boolean*, *Integer*, *Float* und *String*.

Das Script in *Abbildung 84* erzeugt ein passend strukturiertes Subtable-Objekt für seinen Ausgabeparameter *subtable* und demonstriert die beiden alternativen Möglichkeiten, diesem Objekt mit Werten versehene Zeilen hinzuzufügen. *Abbildung 85* zeigt die resultierende Tabelle, die im zweiten Prozessschritt ausgegeben wird.

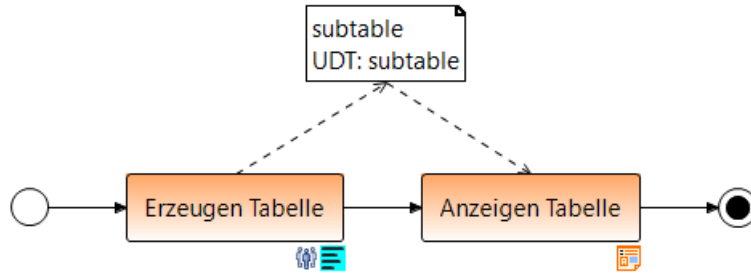


Abbildung 81: Erzeugung eines UDT:subtable-Objektes per Scripting-Aktivität

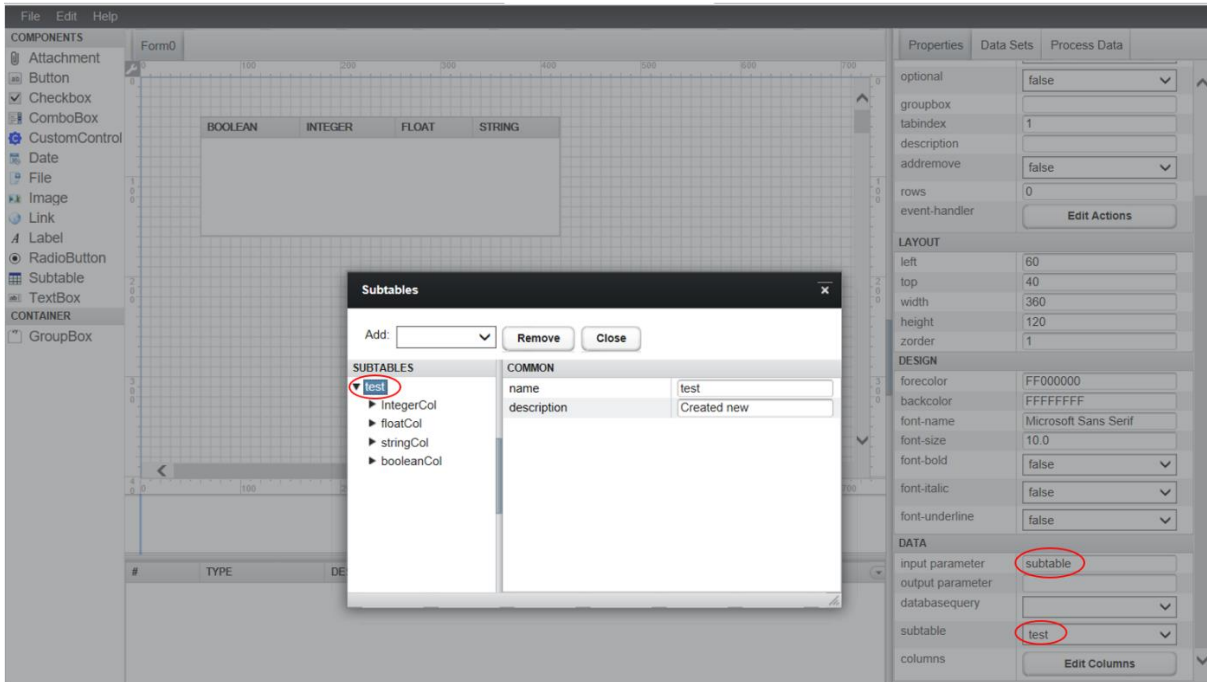


Abbildung 82: SubtableView und subtable-Datenstruktur des Schrittes Anzeigen Tabelle

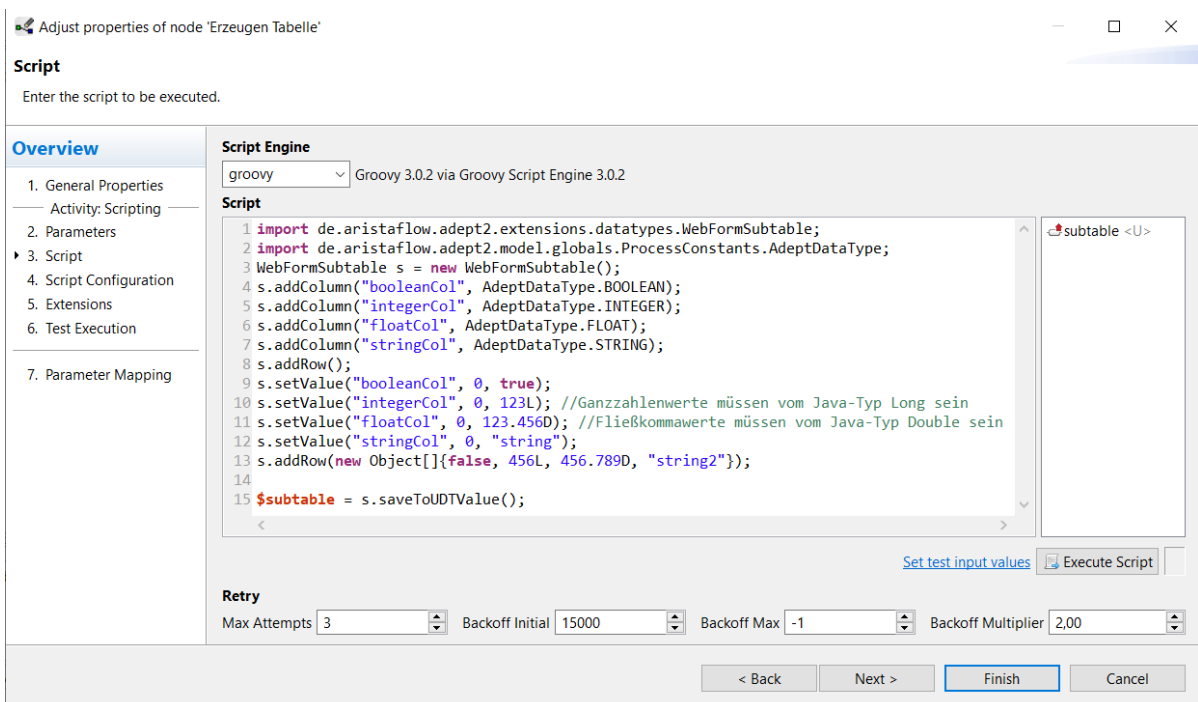


Abbildung 83: Scripting-Aktivität des Schrittes Erzeugen Tabelle

```

import de.aristaflow.adept2.extensions.datatypes.WebFormSubtable;
import de.aristaflow.adept2.model.globals.ProcessConstants.AdeptDataType;
WebFormSubtable s = new WebFormSubtable();
s.addColumn("booleanCol", AdeptDataType.BOOLEAN);
s.addColumn("integerCol", AdeptDataType.INTEGER);
s.addColumn("floatCol", AdeptDataType.FLOAT);
s.addColumn("stringCol", AdeptDataType.STRING);
s.addRow();
s.setValue("booleanCol", 0, true);
s.setValue("integerCol", 0, 123L); // Ganzzahlenwerte müssen vom Java-Typ Long
    sein
s.setValue("floatCol", 0, 123.456D); // Fließkommawerte müssen vom Java-Typ
    Double sein
s.setValue("stringCol", 0, "string");
s.addRow(new Object[]{false, 456L, 456.789D, "string2"});

$subtable = s.saveToUDTValue();

```

Abbildung 84: Script zum Erzeugen und Befüllen des Subtable-Ausgabeobjekts

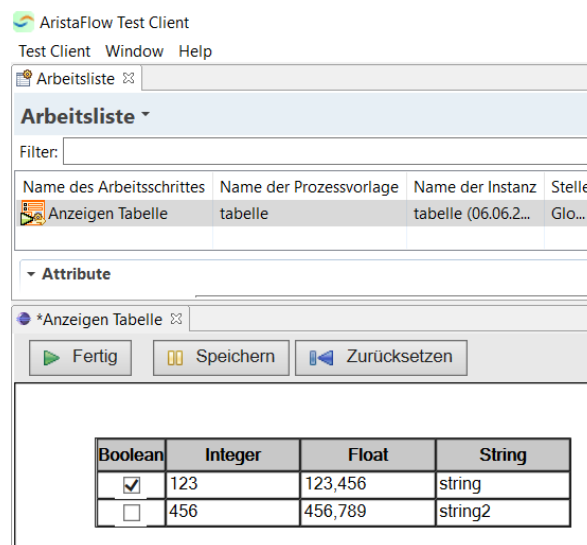


Abbildung 85: Resultierende Tabelle

7.3 USERDEFINED subtable als Input-Parametertyp einer WebForm-Aktivität

Um in einer WebForm-Aktivität einen Input-Parameter von Typ USERDEFINED subtable in eine SubtableView einlesen und dort ggf. weiterverarbeiten zu können, geht man im Prinzip genauso vor,

wie wenn man die Subtable als Output-Parameter ausgeben möchte (siehe Abschnitt 5.5.2

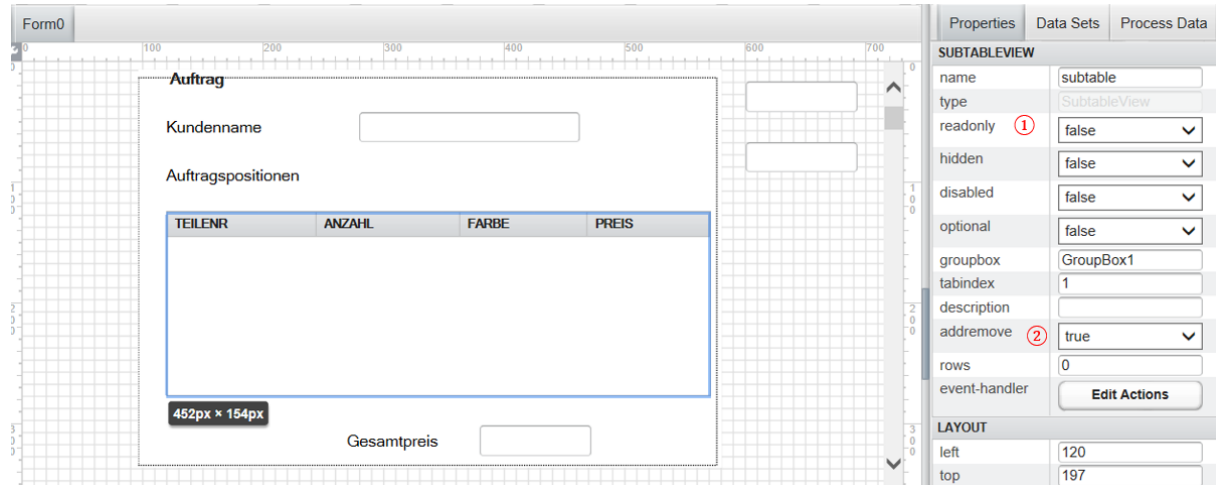


Abbildung 51: Festlegung von readonly- und addremove-Eigenschaften für Subtables

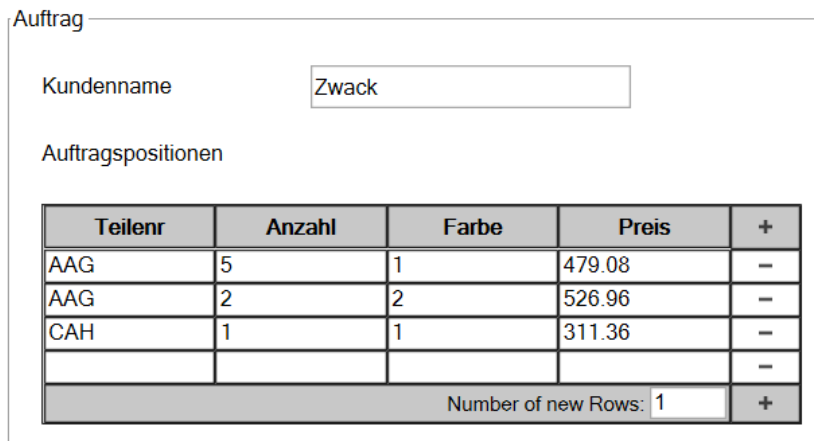


Abbildung 52: Subtable mit addremove = true Eigenschaft

Realisierung von Output-Subtabellen), nur dass man jetzt bei der Definition der Subtable-Datenstruktur darauf achten muss, dass die dort verwendeten Attributnamen exakt gleich wie in der Input-Subtable-Struktur gewählt werden, da die Zuordnung „by Name“ vorgenommen wird. Hinsichtlich der Datentypen hat man hier zwar etwas größere Freiheiten, aber aus Gründen der Nachvollziehbarkeit sollte man diese besser auch identisch wählen.

Hinsichtlich der Gestaltung der im Formular angezeigten Tabelle (SubtableView) ist man wieder frei, da die Zuordnung der Spalten der Subtable-Datenstruktur zu den Spalten der SubtableView wieder explizit (und ggf. auch selektiv) vorgenommen wird. Für Details siehe Abschnitt 5.5.2

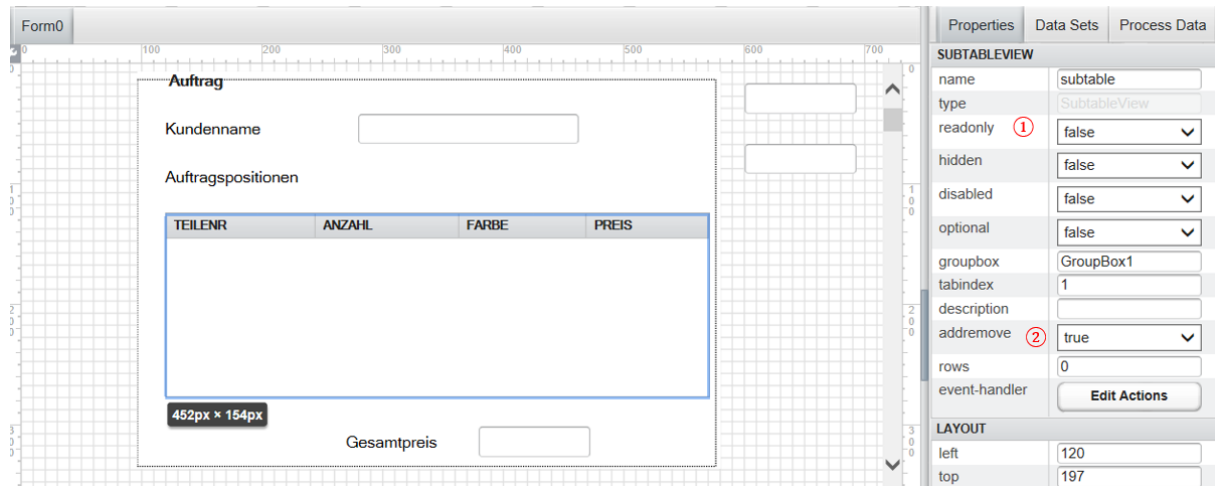


Abbildung 51: Festlegung von readonly- und addremove-Eigenschaften für Subtables

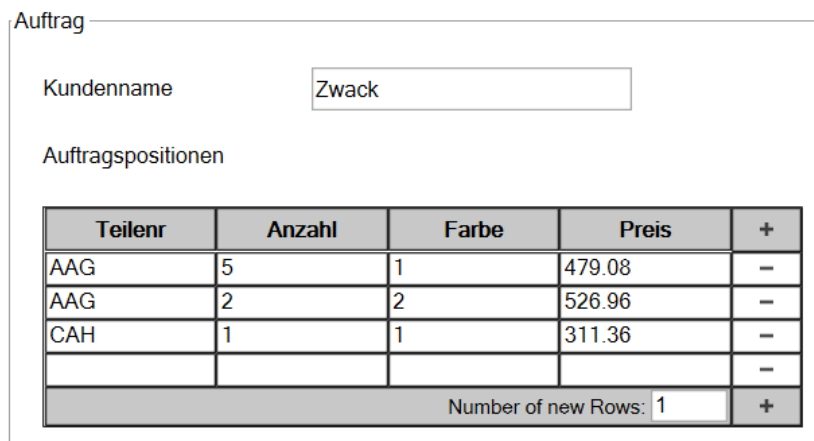


Abbildung 52: Subtable mit addremove = true Eigenschaft

Realisierung von Output-Subtabellen.

7.4 Weiterverarbeitung von USERDEFINED subtable Ausgabeparametern

Für die Verarbeitung von Java-Objekten vom Typ *USERDEFINED subtable* in einer Scripting-Aktivität oder einem konventionellen Javaprogramm stellt die Klasse *WebFormSubtable* ein ganzes Arsenal von Methoden bereit. So kann man unter anderem z.B.

- Die Attributnamen ermitteln (*getColumnNames()*)
- Die Anzahl der Tabellenzeilen ermitteln (*getRowCount()*)
- Gezielt auf Tabellenzeilen und deren Elemente zugreifen (*getValue(...)*)
- Einzelne Zeilen aus der Menge entfernen (*removeRow(...)*)

In Abschnitt 8.5

Komplettbeispiel Auftragserfassung und –speicherung in Datenbank zeigen wir anhand eines Beispiels die Weiterverarbeitung von subtable-Objekten mittels Scripting-Aktivität.

8 Anwendungsbeispiele

8.1 Validieren von Werten von Controls

In diesem Beispiel werden Formulareingaben überprüft und eine Fehlermeldung ausgegeben, wenn diese nicht korrekt sind. In das in Abb. 86 dargestellte Formular sollen zwei Entfernungsangaben in die Controls *vonWert* und *bisWert* eingegeben werden. Ist *vonWert* größer als *bisWert* soll mittels der *alert()*-Funktion von JavaScript jeweils eine entsprechende Fehlermeldung ausgegeben und das Control *entfernung* mit Fragezeichen gefüllt werden.

Die JavaScript-Funktion heißt *checkIntegerDifference()* und wird im Control *berechnung* beim Auslösen eines *onclick*-Events aufgerufen.

Abbildung 86: Formular zur Berechnung einer Entfernung

Die in [Abbildung 87](#) dargestellte Funktion leistet das Geforderte. Bei korrekten Eingaben gibt sie die berechnete Entfernung im Control *entfernung* aus, ansonsten meldet sie den Fehler und füllt das Control *entfernung* mit Fragezeichen aus ([Abbildung 88](#) und [Abbildung 89](#)).

```
function checkIntegerDifference() {
  /* Werte der Felder "vonWert" und "bisWert" abfrage */
  var von = af_getValue("vonWert", false);
  var bis = af_getValue("bisWert", false);
  var diff = parseInt(bis) - parseInt(von);
  if (isNaN(diff)) {
    af_setValue("entfernung", "???");
  }
  else {
    if (diff < 0) {
      alert( "Werte nicht korrekt. "+
            "Der Wert von \"bis\" muss groesser als der von \"von \" sein!");
      af_setValue("entfernung", "???");
    }
    else {
      af_setValue("entfernung", diff);
    }
  }
}
}
```

Abbildung 87: JavaScript: function checkIntegerDifference() – Version 1

Abbildung 88: JavaScript: Formular mit gültigen Eingaben

Abbildung 89: Formular mit ungültigen Eingaben

Geben wir im „von“-Feld Buchstaben ein, erhalten wir eine Fehlermeldung darüber, dass es keinen gültigen Wert ist. Da die Funktion *parseInt()* das eingegebene Zeichen nicht in eine ganze Zahl konvertieren kann, gibt sie *NaN* (steht für *not a number at all*) zurück. Daher kommt die Fehlermeldung.

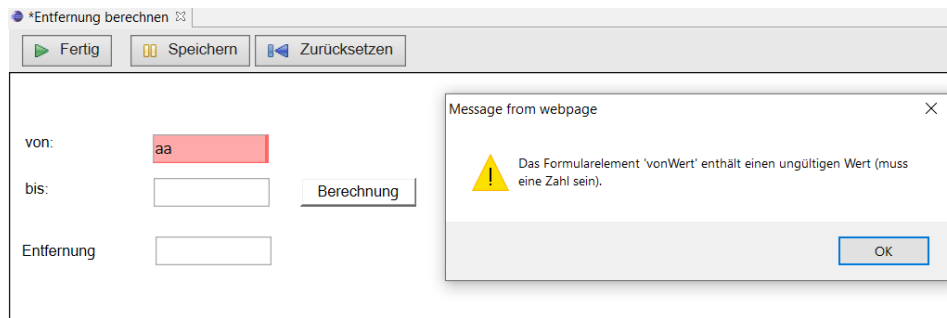


Abbildung 90: Formular mit falscher nicht-numerischer Eingabe

8.2 Ein- und Ausblenden von Controls

Im Formular *Stammdaten abfragen* sollen die Controls *kdname_text*, *kdstadt_textbox*, *kdbonitaet_textbox* und *auftrnr_combobox* nur dann eingeblendet werden, wenn in der DropDown-Liste *kunde_combobox* (Abbildung 5) ein Kunde ausgewählt wurde. D.h. initial sowie bei „Abwahl“ des angezeigten Kunden, sollen diese Controls wieder ausgeblendet werden.

Um diese Controls (inkl. deren Labels) nicht einzeln mittels *af_setHidden(controlName, hidden)* ein- und ausblenden zu müssen, packen wir sie in eine *GroupBox*, deren *Border* wir auf *false* setzen, so dass sie keine Umrandung aufweist (Abbildung 91). Mittels *af_setGroupBoxHidden(groupId, hidden)* können wir die *GroupBox* nebst Inhalt ein- und ausblenden.

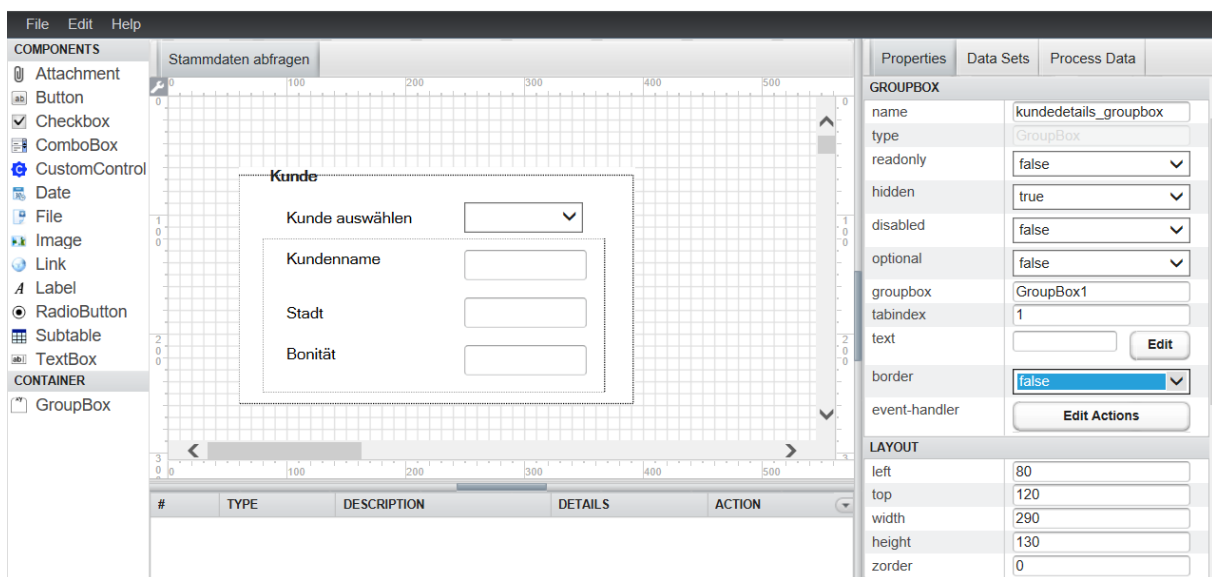


Abbildung 91: Formular Stammdatenabfrage mit zusätzlicher Groupbox

Für das initiale Ausblenden bietet es sich an, im JavaScript-Fenster eine entsprechende Funktion zu hinterlegen, in die wir alles hineinpacken, was beim Laden des Formulars ausgeführt werden soll. Im Beispiel von [Abbildung 91](#) haben wir unsere Groupbox *kundedetails_groupbox* genannt, somit könnte unsere Funktion etwa wie folgt aussehen:

```
function init()
{
    af_setGroupBoxHidden("kundedetails_groupbox", true);
}
```

Abbildung 92: Init() Funktion

Den Aufruf von *init()* platzieren wir im Eventhandler des Formulars(*Properties -> Edit Actions*) indem wir dort ein *onload*-Ereignis definieren.

Für das Ein- und Ausblenden definieren wir uns die analoge *einAusblenden*- Funktion, in der wir den Parameter *Hidden* auf *false* bzw. *true* setzen, wenn *kunde_combobox* einen Wert bzw. keinen Wert aufweist. Die *einAusblenden*-Funktion rufen wir im Eventhandler des Controls *kunde_combobox* auf, indem wir hierfür dort ein *onchange*-Ereignis definieren.

```
function einAusblenden()
{
    af_setGroupBoxHidden("kundedetails_groupbox", !af_hasValue("kunde_combobox"));
}
```

Abbildung 93: einAusblenden() Funktion

8.3 Zeilenweise Verarbeitung von Subtabellen

Mittels `<subtable_name>:<spaltenname>:<zeilennummer>` kann man auf jede Zelle einer Subtabelle zugreifen und deren Wert auslesen oder setzen.

Angenommen, wir möchten eine Tabelle realisieren, bei der eine Spaltensumme berechnet werden soll, etwa wie in [Abbildung 94](#) dargestellt. Sei der Name der Subtabelle *subtable* und der interne Spaltenname *preis* und der interne Name des Summenfelds *gpreis_textbox*, dann liefert die Funktion *berechnePreis* ([Abbildung 95](#)) das gewünschte Ergebnis. Man beachte hier bei, dass der Zeilenindex 1-basiert ist, d.h. die Zählung beginnt nicht bei 0 sondern bei 1.

TeileNr	Anzahl	Farbe	Preis
AAG	5	1	479.08
AAG	2	2	526.96
CAH	1	1	311.36

Abbildung 94: Auftragserfassung mit Gesamtwert-Berechnung

```
function berechnePreis()
{
    var gesamtPreis = 0;
    for (var i = 1; i <= af_getSubtableViewRowCount("subtable"); i++)
    {
        gesamtPreis = parseFloat(af_getValue("subtable:preis:" + i)) + gesamtPreis;
    }
    af_setValue("gpreis_textbox", af_formatNumber(gesamtPreis, "#0.00"));
}
```

Abbildung 95: JavaScript-Funktion: Ermitteln Spaltensumme

8.4 Realisierung von Auswahl-Tabellen

	Nummer	Name	Stadt
<input type="checkbox"/>	105	Aberer	Stuttgart
<input type="checkbox"/>	123	Babel	Senden
<input checked="" type="checkbox"/>	137	Becker	Stuttgart
<input type="checkbox"/>	219	Beyer AG	Bretten
<input type="checkbox"/>	188	Breu	Karlsruhe
<input type="checkbox"/>	150	Dehling	Augsburg

Abbildung 96: Auswahl-Tabelle

Die Auswahl von Zeilen in einer Tabelle erfolgt im WebForm-Designer mittels Checkbox ([Abbildung 96](#)). Wir müssen daher einer Tabelle noch eine Checkbox-Spalte hinzufügen ([Abbildung 97](#)).

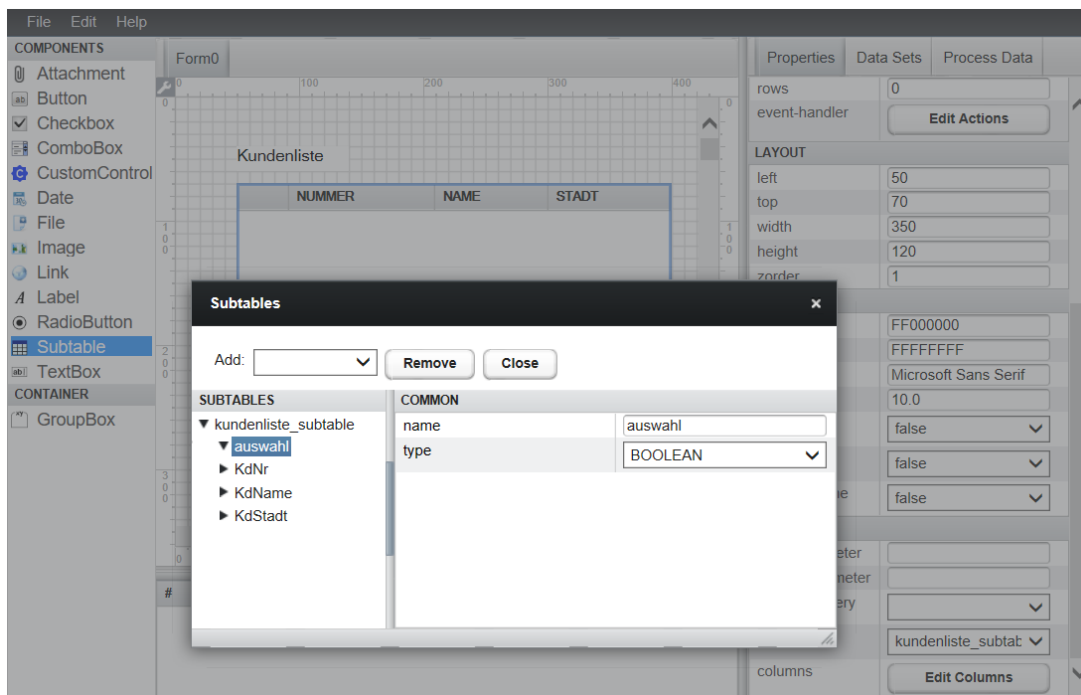


Abbildung 97: Anlegen einer Checkbox-Spalte

Die Checkbox-Einträge in den einzelnen Zeilen sind hierbei erst einmal unabhängig voneinander. D.h. wir können eine Checkbox aktivieren und deaktivieren, ohne dass dies sich auf die anderen Checkboxes in derselben Spalte auswirkt; die Mehrfachauswahl ist somit der Default. Welche Zeilen ausgewählt wurden, können wir mittels eines Schleifendurchlaufs (siehe nächster Abschnitt) und Anwendung der Funktion `af_isChecked(controlname)` ermitteln.

8.4.1 Realisierung von „Multi-Selection-Auswahltabellen“

Wie bereits vorstehend erwähnt, ist die Mehrfachauswahl der Default. Falls uns nur die Anzahl der ausgewählten Zeilen oder die Spaltensumme interessiert, dann müssen wir lediglich eine JavaScript-Funktion realisieren, mit der wir die einzelnen Zeilen durchlaufen der `SubtableView` durchlaufen und das Checkbox-Attribut mittels `af_isChecked(...)` auf `true` bzw. `false` prüfen.

In dem in [Abbildung 98](#) und [Abbildung 99](#) dargestellten Beispiel gehen wir einen Schritt weiter und löschen die nicht markierten Zeilen aus der `SubtableView`. Die verbleibenden Zeilen geben wir als

Ausgabeparameter vom Typ *USERDEFINED subtable* zurück. Für Demonstrationszwecke geben wir *TeileNr* und *Farbe* der verbliebenen Zeilen zusätzlich auch noch in einer Textbox aus.

The screenshot shows a web form with a title bar containing the text '*Erfassen Auftrag'. Below the title bar are three buttons: 'Fertig', 'Speichern', and 'Zurücksetzen'. A 'Kunde:' label is followed by an empty text input field. To the right, there is a section titled 'Kontrollausgabe Auswahl' containing a large empty text area and a button labeled 'Erzeuge Auswahl'. Below the customer field is a table with the following data:

	TeileNr	Farbe	Bezeichnung	Preis
<input type="checkbox"/>	AAG	1	Aufbauanhänger geschlossen	479.08
<input type="checkbox"/>	AAG	2	Aufbauanhänger geschlossen	526.96
<input type="checkbox"/>	AKK	1	Kippanhänger kurz	318.08
<input type="checkbox"/>	AKK	2	Kippanhänger kurz	472.08
<input type="checkbox"/>	AKL	1	Kippanhänger lang	519.26
<input type="checkbox"/>	AKL	2	Kippanhänger lang	540.54

Abbildung 98: Auswahlformular nach Start

The screenshot shows the same web form as in the previous image, but with the 'Fertig' button highlighted in blue. In the table, the first three rows are now selected, indicated by checked checkboxes in the first column. The 'Kontrollausgabe Auswahl' text area now contains the following text:

```
AKK 2
AKK 1
AAG 2
```

Abbildung 99: Auswahlformular nach erfolgter Auswahl

Die in [Abbildung 100](#) dargestellte JavaScript-Funktion *erzeugenAuswahlmenge()* führt die Löschung der „abgewählten“ Zeilen (von hinten nach vorne!) sowie das Befüllen der Textbox durch. Um die übrig gebliebenen „Rest-Tabelle“ als Ausgabeparameter zu verwenden, müssen wir lediglich die Subtabelle mit einer Subtabellen-Datenstruktur „unterfüttern“ (siehe Abschnitt 5.5.2 [Realisierung von Output-Subtabellen](#)) und festlegen, welche Attribute der *SubtableView* in die Ausgabe-Tabelle übernommen werden sollen. In unserem Beispiel hat die *SubtableView* den internen Namen *TeileKatalog_SubtableView* und die mit ihr assoziierte Subtable den Namen *auswahl_subtable*. Da – bis auf das Löschen von Zeilen – die Zelleninhalte in der *SubtableView* nicht geändert werden sollen, setzen wir diese (mit Ausnahme der Checkbox-Spalte) auf *read-only*.

```

function erzeugenAuswahlmenge()
/* Entfernt alle nicht-ausgewählten Teile aus der Tabelle und gibt die ausgewählten
Teile mit */
/* TeileNr und Farbe in der Kontroll-Textbox aus. */
/* Die mit der SubtableView assoziierte Subtable auswahl_subtable ist
gleichzeitig */
/* Ausgabeparameter der Aktivität. */

{
  var kontrollausgabe = "";
  var teilenr;
  var farbe;

  var rowCount = af_getSubtableViewRowCount("TeileKatalog_SubtableView");
  for (var i = rowCount; i > 0; i--)
    if (af_isChecked("TeileKatalog_SubtableView:auswahl:" + i))
      {
        teilenr = af_getValue("TeileKatalog_SubtableView:teilenr:" + i);
        farbe = af_getValue("TeileKatalog_SubtableView:farbe:" + i);
        kontrollausgabe = kontrollausgabe + teilenr + " " + farbe + "\n";
        /* oder anderweitige Verarbeitung der Zeile */
      }
    else
      {
        af_removeSubtableRow("TeileKatalog_SubtableView", i);
      }
  af_setValue("kontrollausgabe_textbox", kontrollausgabe);
}

```

Abbildung 100: JavaScript-Funktion zum Löschen der abgewählten Zeilen

8.4.2 Realisierung von „Single-Selection“-Auswahltabellen

In einer „Single-Selection“-Auswahltabelle kann stets nur max. eine Zeile per Checkbox ausgewählt sein, wie in [Abbildung 101](#) illustriert. Ein solches „Single-Selection“-Verhalten können wir einer *SubtableView* dadurch aufprägen, dass wir bei Auswahl einer Zeile die Checkboxes aller anderen Zeilen mit *af_setValue(...)* auf *false* setzen.

Kundenliste

	Nummer	Name	Stadt
<input type="checkbox"/>	105	Aberer	Stuttgart
<input type="checkbox"/>	123	Babel	Senden
<input checked="" type="checkbox"/>	137	Becker	Stuttgart
<input type="checkbox"/>	219	Beyer AG	Bretten
<input type="checkbox"/>	188	Breu	Karlsruhe
<input type="checkbox"/>	150	Dehling	Augsburg

Kontrollausgabe
Auswahl

137 Becker

Abbildung 101: Beispiel für „Single-Selection“-Auswahltabelle

Um herauszufinden, in welche Zeile geklickt wurde, definieren wir in der Checkbox-Spalte der *SubtableView* ein *onclick*-Ereignis. Bei diesem hinterlegen wir eine JavaScript-Funktion, der wir als

Aufrufparameter *this* übergeben. [Abbildung 102](#) illustriert die Definition dieses Ereignisses und die Hinterlegung einer JavaScript Funktion namens *rowSelected*.

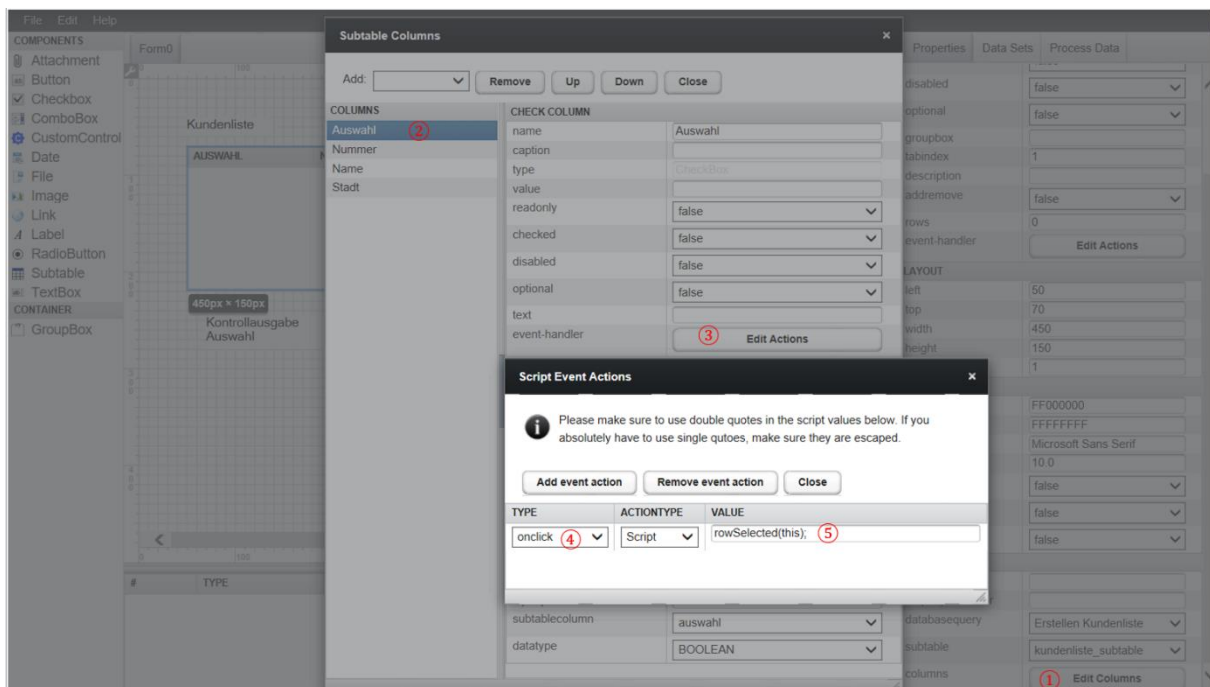


Abbildung 102: Hinterlegen eines onclick-Ereignisses mit Selbst-Referenz

Zu Demonstrationszwecken sollen die Attribute *KdNr* und *KdName* der gewählten Zeile in einer Textbox zur Kontrolle ausgegeben werden ([Abbildung 101](#)) bzw. deren Inhalt gelöscht werden, wenn keine Zeile markiert ist. [Abbildung 103](#) zeigt die Implementierung dieser Funktion in JavaScript. Damit die Anwendungen der *af_setValue*-Funktion in diesen Checkboxes nicht wiederum selbst *onclick*-Ereignisse auslösen („kaskadierende Events“), müssen wir in deren *af_setValue*-Funktionsaufrufen den (optionalen) dritten Parameter auf *false* setzen, was das Auslösen von Events unterdrückt. Siehe hierzu die mit ★ -markierte Stelle in [Abbildung 103](#).

```

function rowSelected(elem)
{
    // Zeilennummer über die ID des Elements herausfinden
    var rowID = elem.id.split(":")[2];

    // falls "checked", alle anderen Checkeinträge löschen und Kontrollausgabe
    aktualisieren
    if (af_isChecked("kundenliste_subtable:Auswahl:" + rowID))
    {
        var maxRowIndex = af_getSubtableViewRowCount("kundenliste_subtable");
        for (var i = 1; i <= maxRowIndex; i++)
        {
            if (i != rowID)
                //uncheck Checkbox und unterdrücken Auslösen eines Events
                af_setValue("kundenliste_subtable:Auswahl:" + i, false, false); //★
            else // i == rowID
            {
                // Übertragen Werte in Kontroll-Anzeige
                var nummer = af_getValue("kundenliste_subtable:Nummer:" + rowID);
                var name = af_getValue("kundenliste_subtable:Name:" + rowID);
                var kontrollausgabe = nummer + " " + name;
                af_setValue("kontrollausgabe_textbox", kontrollausgabe);
            }
        }
    }
    else // Check wurde wieder entfernt
        af_setValue("kontrollausgabe_textbox", "");
}

```

Abbildung 103: JavaScript-Funktion zur Realisierung der Single-Selection Auswahltable

8.5 Komplettbeispiel Auftragserfassung und -speicherung in Datenbank

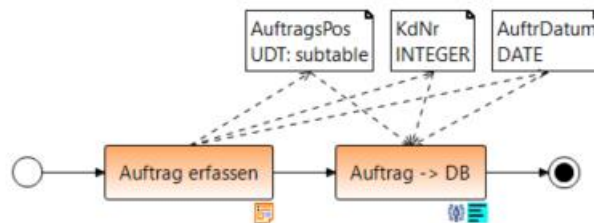


Abbildung 104: Prozess „Auftragerfassung und -speicherung in Datenbank“

8.5.1 Schritt 1: Auftrag erfassen

Abbildung 105: Subformular Anlegen Auftrag des Prozessschrittes Auftrag erfassen

Abbildung 106: Subformular Erfassen Auftragspositionen des Prozessschrittes Auftrag erfassen

In *Tabelle 5* werden die einzelnen Prozessschritte erläutert und vermerkt, in welchen Abbildungen sich ergänzende Informationen finden.

Prozessschritt	Eingabe- parameter	Aktion	Abb.	Ausgabe- parameter
Erfassen Auftrag (WebForm)	...	Im ersten Subformular werden zunächst die Kundennummer und das Auftragsdatum erfasst, wobei die Kundennummer aus einer Single-Selection-Auswahltabelle ausgewählt wird.	Abb. 104 105 106 107 108 109	KdNr AuftrDatum AuftragsPos
		Mittels Button <i>Auftrag anlegen</i> wird dann auf das zweite Subformular „umgeschaltet“, in dem die Positionen des Auftrags erfasst werden. Die Artikel werden per Single-Selection-Auswahltabelle ausgewählt und vom	Abb. 110 111 112 113	

		Anwender um die gewünschte Anzahl ergänzt. Weitere Erläuterungen zu diesem Formular und den darin verwendeten JavaScript-Funktionen folgen weiter unten.		
Auftrag → DB (executeScript)	KdNr AuftrDatum AuftragsPos	Fügt ein neues Auftragsstapel in die DB ein und vergibt dabei eine neue Auftragsnummer (AuftrNr). Ermittelt die Anzahl der Zeilen in der Auftragspos-Subtable und fügt die Daten aus dieser Subtable in die auftragspos-Tabelle der Lego-Trailer-Datenbank ein. Die Position des Auftrags wird dabei automatisch vergeben.	Abb. 116 117 118	

Tabelle 5: Erläuterungen zum Prozess *Auftragserfassung und -speicherung in Datenbank*

Wir erstellen im Subformular *Anlegen Auftrag* eine Subtable *Kunde_SubtableView* mit den Spalten *Auswahl*, *Nummer*, *Name*, *Stadt* und *Bonitaet*. Die weiteren Controls entnehmen Sie der [Abbildung 107](#). Dabei verweist das Auftragsdatum auf das Output Parameter *AuftrDatum* und die Kundennummer TextBox auf *KdNr*.

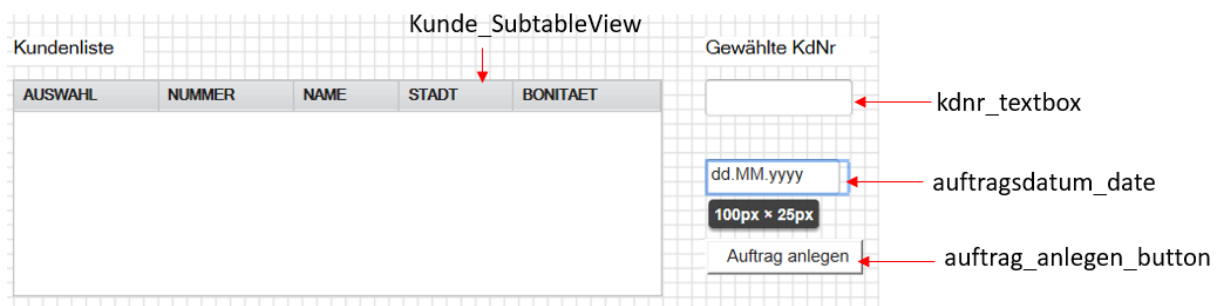


Abbildung 107: Webformular *Anlegen Auftrag*

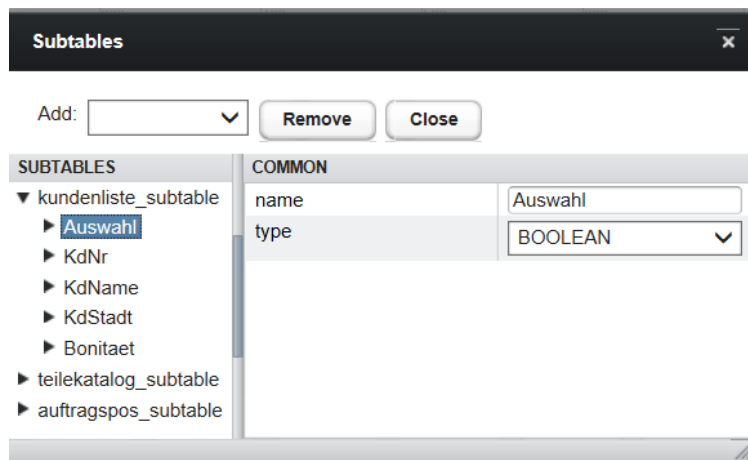


Abbildung 108: Datenstruktur der Subtable *Kunde_SubtableView*



Abbildung 109: SQL Anfrage Erstellen Kundenliste für Kunde_SubtableView

Die im Prozessschritt *Erfassen Auftrag* verwendeten JavaScript-Funktionen sind in [Abbildung 114](#) und [Abbildung 115](#) vollständig abgebildet. In [Tabelle 6](#) ist aufgeführt, welche Ereignisse in welchen Controls definiert sind und mit welchen JavaScript- Funktion verknüpft sind.

Name der Funktion	Enthalten in	Control	Ereignistyp
function init()	Abb. 114	Main Form	onload
checkEnableButton()	Abb. 114	KdNr in Subform() Auftragsdatum in Subform()	onchange
copyDate()	Abb. 114	Datumsfeld in Subform()	onchange
validateSubform0()	Abb. 114	auftrag_anlegen_button	onclick
validateForm()	Abb. 114	Main Form	onsubmit
kundeGewaehlt(elem)	Abb. 115	Spalte Auswahl in SubtableView Kunden_SubtableView	onclick
rowSelected(elem)	Abb. 115	Spalte Auswahl in SubtableView TeileKatalog_SubtableView	onclick

Tabelle 6: Im Formular *Auftrag erfassen* verwendete JavaScript-Funktionen und Ereignisse

Im Subformular *Erfassen Auftragspositionen* sind zwei Subtabellen *TeileKatalog_SubtableView* und *AuftragsPos_SubtableView*. Zusätzlich gibt es Textboxen zur Ausgabe des ausgewählten Kunden und ein Datumselement *auftragsdatum_date2* zur Anzeige des Auftragsdatums aus dem ersten Subformular. Aus dem Teilekatalog in der ersten Subtabelle dürfen mehrere Zeilen ausgewählt und für diese ausgewählten Teilen in der zweiten Subtabelle die Anzahl manuell festgelegt werden.

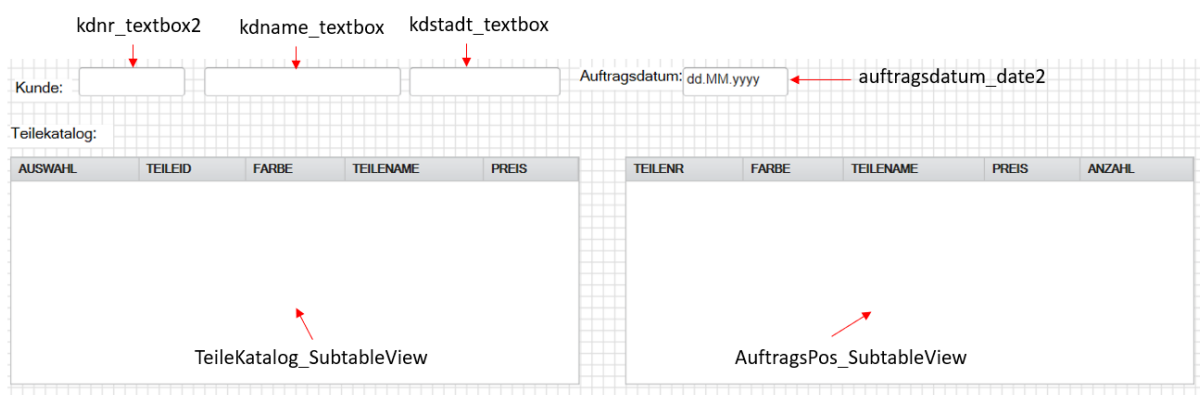


Abbildung 110: Webformular Erfassen Auftragspositionen

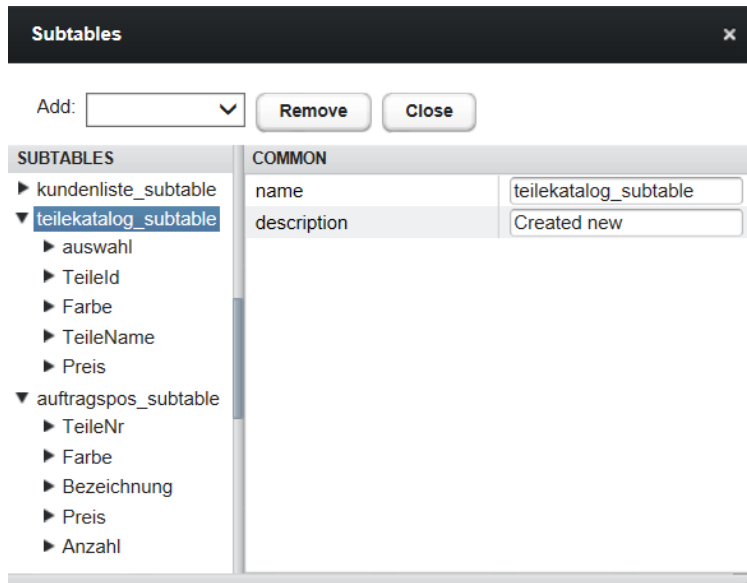


Abbildung 111: Datenstrukturen von TeileKatalog_SubtableView und AuftragsPos_SubtableView

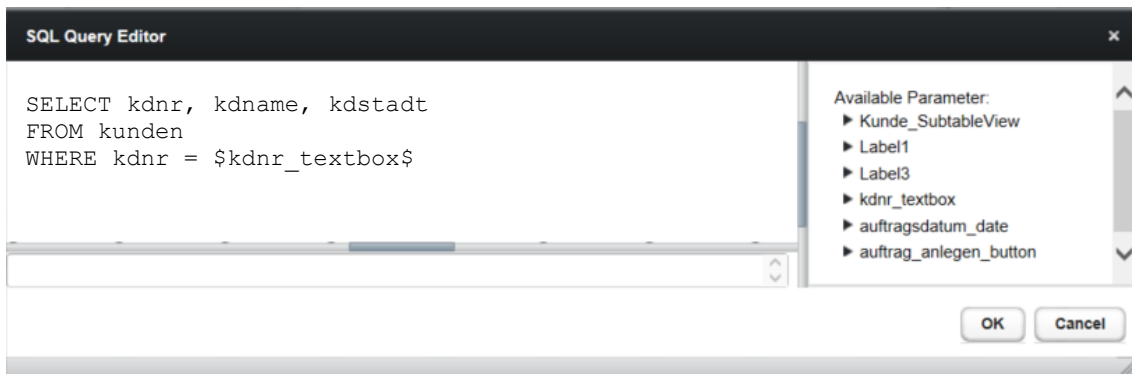


Abbildung 112: SQL Anfrage Ausgabe ausgewählter Kunde für Kunden Textboxen

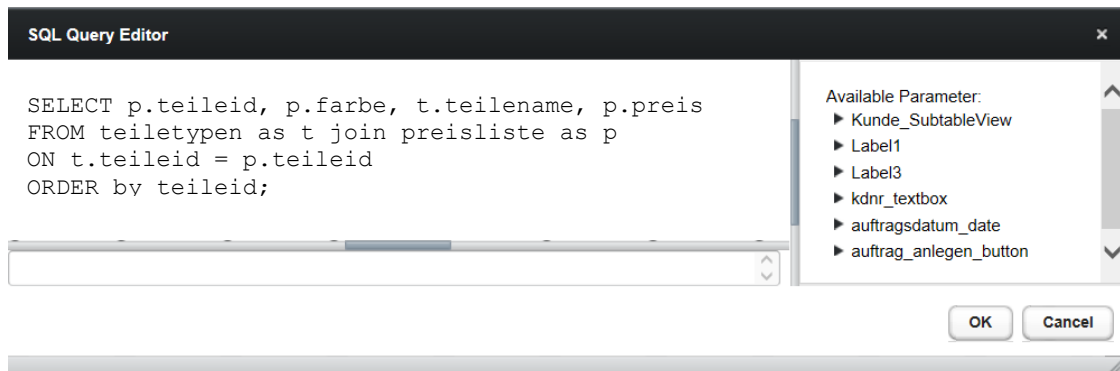


Abbildung 113: SQL Anfrage Teilekatalog für TeileKatalog_SubtableView

```

function init()
{
    // Hide Erfassen Auftragspositionen + disable 'Auftrag anlegen'
    af_setSubformHidden("Erfassen Auftragspositionen", true);
    af_setDisabled("auftrag_anlegen_button", true);
}

function checkEnableButton()
{
    // Prueft, ob Kundennummer und Auftragsdatum gesetzt sind
    // und schaltet ggf. die Subform 'Erfassen Auftragspositionen frei
    if (af_hasValue("kdnr_textbox") && af_hasValue("auftragsdatum_date"))
        af_setDisabled("auftrag_anlegen_button", false);
    else
        af_setDisabled("auftrag_anlegen_button", true);
}

function copyDate()
{
    // Uebernimmt ausgewaehltes Auftragsdatum in Subform2
    var datum = af_getValue("auftragsdatum_date");
    af_setValue("auftragsdatum_date2", datum);
}

function validateSubform0()
{
    // Pruefung, ob Subform0 vollstaendig ausgefuellt wurde. Falls ja, umschalten auf
    // Subform1
    if (af_hasValue("kdnr_textbox") && af_hasValue("auftragsdatum_date"))
    {
        af_setSubformHidden("Anlegen Auftrag", true);
        af_setSubformHidden("Erfassen Auftragspositionen", false);
        af_activateSubform("Erfassen Auftragspositionen");
    }
    else
        alert("Auftragserfassung ist unvollstaendig!");
}

function validateForm()
{
    // Pruefung, ob SEND erlaubt werden kann
    if (!af_hasValue("kdnr_textbox") || !af_hasValue("auftragsdatum_date"))
    {
        alert("Auftragserfassung ist unvollstaendig: Kundennummer o. Auftragsdatum
        fehlt!");
        return false;
    }
    else if (af_getSubtableViewRowCount("AuftragsPos_SubtableView") < 1)
    {
        alert("Keine Auftragspositionen erfasst!");
        return false;
    }
    else
        return true;
}

```

Abbildung 114: JavaScript-Funktionen des Formulars Erfassen Auftrag (1)

```

function kundeGewaehlt(elem)
{
    // Zeilennummer über die ID des Elements herausfinden
    var rowID = elem.id.split(":")[2];
    // falls "checked", alle anderen Checkeinträge löschen und Kontrollausgabe
    // aktualisieren
    if (af_isChecked("Kunde_SubtableView:Auswahl:" + rowID))
    {
        var maxRowIndex = af_getSubtableViewRowCount("Kunde_SubtableView");
        for (var i = 1; i <= maxRowIndex; i++)
            if (i != rowID)
                af_setValue("Kunde_SubtableView:Auswahl:" + i, false, false);
        else // i == rowID
        {
            // Übertragen Werte in Kundennummer-Textbox
            var kdnr = af_getValue("Kunde_SubtableView:Nummer:" + rowID);
            var kdname = af_getValue("Kunde_SubtableView:Name:" + rowID);
            var kdstadt = af_getValue("Kunde_SubtableView:Stadt:" + rowID);
            af_setValue("kdnr_textbox", kdnr); af_setValue("kdnr_textbox2", kdnr);
            af_setValue("kdname_textbox", kdname); af_setValue("kdstadt_textbox",
            kdstadt);
        }
    }
    else
        af_setValue("kdnr_textbox", "");
}

function rowSelected(elem)
{
    // Zeilennummer über die ID des Elements herausfinden
    var rowID = elem.id.split(":")[2];
    // falls "checked", alle anderen Checkeinträge löschen,
    // Kontrollausgabeaktualisieren
    if (af_isChecked("TeileKatalog_SubtableView:auswahl:" + rowID))
    {
        var maxRowIndex = af_getSubtableViewRowCount("TeileKatalog_SubtableView");
        for (var i = 1; i <= maxRowIndex; i++)
            if (i != rowID)
                af_setValue("TeileKatalog_SubtableView:auswahl:" + i, false, false);
        else // i == rowID
        {
            // Übertragen Werte in Auftragspositionstabelle
            var teilenr = af_getValue("TeileKatalog_SubtableView:teilenr:" + rowID);
            var farbe = af_getValue("TeileKatalog_SubtableView:farbe:" + rowID);
            var teilenname = af_getValue("TeileKatalog_SubtableView:teilenname:" +
            rowID);
            var preis = af_getValue("TeileKatalog_SubtableView:preis:" + rowID);
            af_addSubtableRow("AuftragsPos_SubtableView");
            var newRow = af_getSubtableViewRowCount("AuftragsPos_SubtableView");
            af_setValue("AuftragsPos_SubtableView:teilenr:" + newRow, teilenr);
            af_setValue("AuftragsPos_SubtableView:farbe:" + newRow, farbe);
            af_setValue("AuftragsPos_SubtableView:teilenname:" + newRow, teilenname);
            af_setValue("AuftragsPos_SubtableView:preis:" + newRow, preis);
        }
    }
}

```

Abbildung 115: JavaScript-Funktionen des Formulars Erfassen Auftrag (2)

8.5.2 Schritt 2: Auftrag -> DB mit der Aktivität Scripting

In der [Abbildung 116](#) wird dargestellt, wie die Datenbankverbindung durch Extensions Configuration konfiguriert wird. Dabei wird jdbc als Extensions genommen und darunter die Daten für die Verbindung eingetragen.

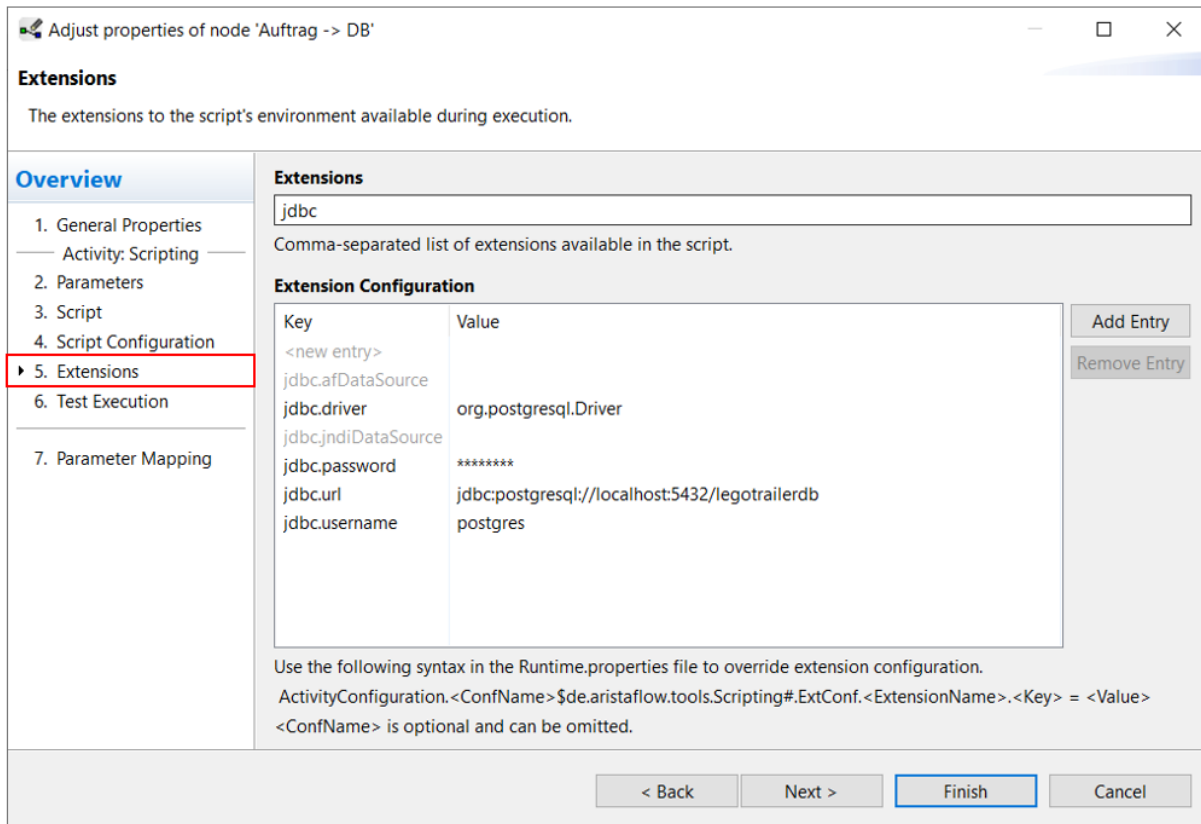


Abbildung 116: Konfiguration für Datenbankverbindung

Das auszuführende Skript wird im Assistenten der Scripting-Aktivität wie in der [Abbildung 117](#) eingegeben. Als Skriptsprache wird Groovy benutzt und während der Eingabe durch Execute Script (rechts unten) auf Syntaxfehler getestet. Für die Speicherung der Aufträge in die Datenbank wird zuerst die Verbindung zur Datenbank hergestellt und danach zwei Insert-Abfragen ausgeführt. Die Inputparameter AuftrDatum und KdNr werden für die erste, die Subtable AuftragsPos für die zweite Abfrage verwendet. Vor der ersten Abfrage wird die höchste Auftragsnummer ermittelt und in die Tabelle auftraege mit den Daten des ausgewählten Kunden eingetragen. Für die Eintragung der Auftragspositionen in die Datenbanktabelle auftragspos wird jede Zeile der Subtable AuftragsPos ausgelesen und dabei die Nummer der Auftragspositionen automatisch ermittelt.

Adjust properties of node 'Auftrag -> DB'

Script
Enter the script to be executed.

Overview

- General Properties
- Parameters
- Script**
- Script Configuration
- Extensions
- Test Execution
- Parameter Mapping

Script Engine
groovy Groovy 3.0.2 via Groovy Script Engine 3.0.2

Script

```

1 import de.aristaflow.adept2.extensions.datatypes.WebFormSubtable;
2 import java.sql.Timestamp;
3
4 def con = _jdbc.getConnection();
5 def sql = new Sql(con);
6
7 def auftrnr = sql.firstRow("SELECT MAX(auftrnr) + 1 AS auftrnr FROM auftrae
8 def datum = new Timestamp($AuftrDatum.getTime())
9
10 sql.executeInsert("""
11     INSERT INTO auftraege (auftrnr, kdnr, auftrdatum)
12     VALUES (${auftrnr}, ${$KdNr}, ${datum})
13     """);
14
15 WebFormSubtable st = new WebFormSubtable();
16 st.loadFromUDTValue($AuftragsPos);
17 var anzahlPos = st.getRowCount();
18
19 for (i = 0; i < anzahlPos; i++)
20 {
21     def pos = i + 1;
22     def teileid = st.getValue("TeileNr", i);
23     def farbe = st.getValue("Farbe", i);
24     def anzahl = st.getValue("Anzahl", i);
25
26     sql.executeInsert("""
27         INSERT INTO auftragspos (auftrnr, pos, teileid, farbe, anzahl)
28         VALUES (${auftrnr}, ${pos}, ${teileid}, ${farbe}, ${anzahl})
29         """);
30 }

```

[Set test input values](#)

Retry

Max Attempts Backoff Initial Backoff Max Backoff Multiplier

Abbildung 117: Script für den Eintrag in die Datenbank (1)

```

import de.aristaflow.adept2.extensions.datatypes.WebFormSubtable;
import java.sql.Timestamp;

def con = _jdbc.getConnection();
def sql = new Sql(con);

def auftrnr = sql.firstRow("SELECT MAX(auftrnr) + 1 AS auftrnr FROM
auftraege")['auftrnr'];

def datum = new Timestamp($AuftrDatum.getTime());

sql.executeInsert("""
    INSERT INTO auftraege (auftrnr, kdnr, auftrdatum)
    VALUES (${auftrnr}, ${KdNr}, ${datum})
""");

WebFormSubtable st = new WebFormSubtable();
st.loadFromUDTValue($AuftragsPos);
var anzahlPos = st.getRowCount();

for (i = 0; i < anzahlPos; i++)
{
    def pos = i + 1;
    def teileid = st.getValue("TeileNr", i);
    def farbe = st.getValue("Farbe", i);
    def anzahl = st.getValue("Anzahl", i);

    sql.executeInsert("""
        INSERT INTO auftragspos (auftrnr, pos, teileid, farbe, anzahl)
        VALUES (${auftrnr}, ${pos}, ${teileid}, ${farbe}, ${anzahl})
        """);
}

```

Abbildung 118: Script für den Eintrag in die Datenbank (2)

9 Trouble Shooting

9.1 Testen der Datenbankverbindung

Im Gegensatz zu den *executeStatements*-SQL-Aktivitätenvorlagen der AristaFlow BPM Suite bietet der WebForm-Designer leider nicht die Möglichkeit bereits im Entwurfsmodus zu testen, ob mit den spezifizierten Verbindungsdaten eine Verbindung zum Datenbanksystem hergestellt werden kann.

Tipp:

Die Verbindungsdaten mit der *executeStatements*-Aktivität (*Check connection*; [Abbildung 119](#)) überprüfen. Wenn ein Verbindungsaufbau dort klappt, dann klappt er auch im WebForm-Designer.

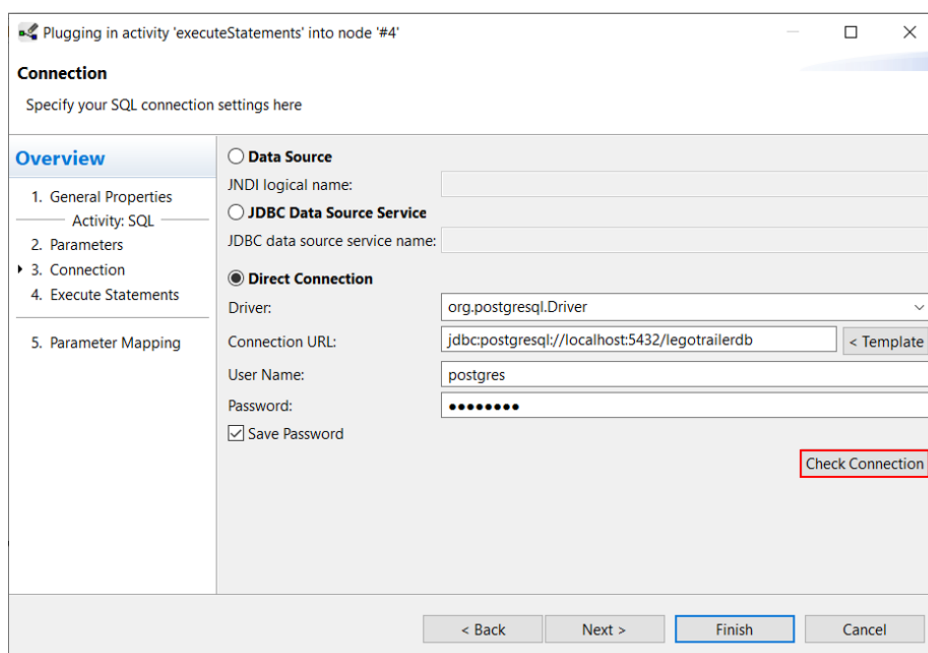


Abbildung 119: Testen der Datenbankverbindung mit der *executeStatements*-Aktivität

Alternativ kann man natürlich z.B. auch eine Verbindung zur gewünschten Datenbank mit dem SQL-Explorer herstellen und schauen, welche Verbindungsdaten dort hinterlegt sind.

9.2 Erkennung von SQL-Ausführungsfehlern

Im Gegensatz zur *executeStatements*-Aktivität führt eine fehlerhafte SQL-Anweisung beim WebForm-Designer zur Laufzeit nicht zu einem Fehlschlag der Aktivität, da dieses Ausführungsfehler von Datenbankabfragen (aus implementierungstechnischen Gründen) intern abfängt und nicht „nach oben“ reicht. Wenn z.B. eine Datenbankabfrage auch eine leere Treffermenge zurückliefern kann, dann sieht man es dem Formular nicht an, ob ein Ausgabefeld nicht gefüllt worden ist, weil die Datenbankabfrage keine Treffer ergeben hat oder weil sie wegen Syntaxfehler fehlgeschlagen ist.



Tipp:

Im AristaFlow-TestClient die *Logger-View* aktivieren. Diese erreicht man über das *Window*-Menü wie folgt: *Window* → *Show View* → *Other...* → *Logger*

Durch Doppelklick auf die Nachricht in der Abb. 118 öffnet sich ein Fenster mit näheren Angaben, warum ein Prozessschritt fehlgeschlagen ist.

So kann man z.B. der Nachricht in [Abbildung 120](#) entnehmen, dass eine SQL-Anfrage nicht ausgeführt werden konnte, da die Spalte *aberer* nicht existiert. Hier konnte die SQL-Anfrage zur Anzeige des ausgewählten Kunden nicht ausgeführt werden, da der Control *kunde_combobox* anstatt der Kundennummer den Kundennamen als Wert enthält.

The screenshot shows the 'Stammdaten abfragen' window with a form for selecting a customer and a log window below it. The form has fields for 'Kunde auswählen' (set to 'Aberer'), 'Kundenname', 'Stadt', 'Bonität', and 'Auftragsnummer'. The log window shows several messages, with one highlighted in blue: 'InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 Could not execute query.'

```
1 severe. 13 warnings. 455 info. (Filter matched 469 of 469 items) (469 items loaded totally)
Message
i InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 Replaced Value: Aberer
i InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 replaced query: SELECT kdname, kdstadt, bonitaetFROM kundenWHERE kdnr = Aberer
i InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 Could not execute query.
i InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 Could not execute query.
i InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 Could not execute query.
i InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 HTTPReply: {UserSignal=Reload, kunde_combobox=Aberer, elementType=textbox, elementName=kdbonitaet_textbox}
i InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 UserSignal: Reload
i InstID-Node-Iter: 4b42c7bc-14cb-42b5-8906-be1a3f3280fd-3-0 Reload: Subtable=null, fieldName=kdbonitaet_textbox, row=0
```

Abbildung 120: AristaFlow-Logger View – Liste der Nachrichten

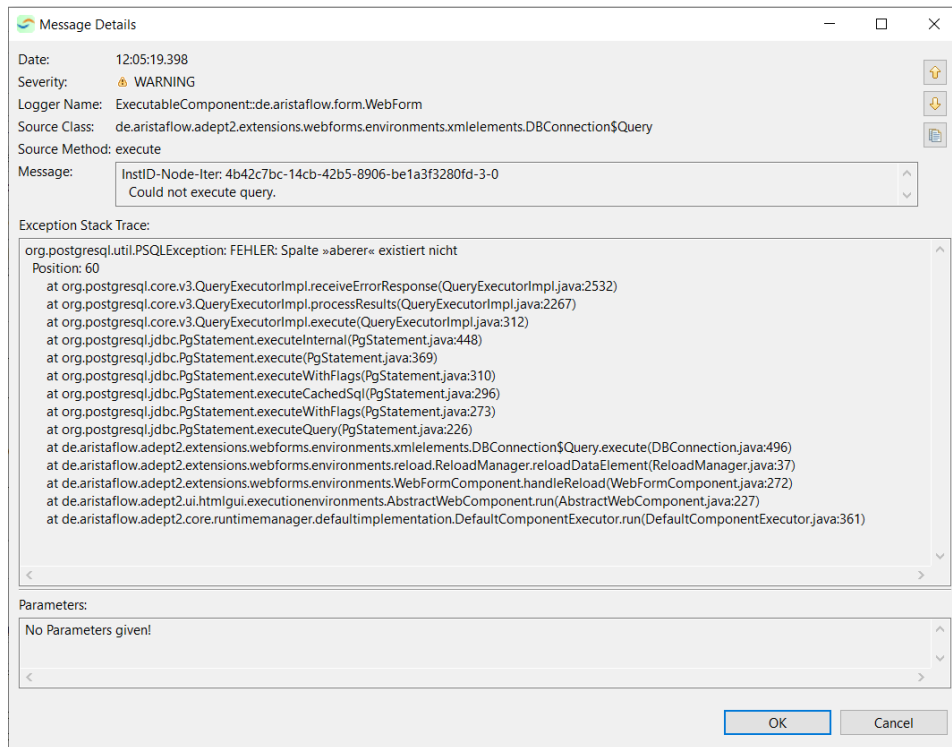


Abbildung 121: AristaFlow-Logger-View Detaildarstellung

9.3 JavaScript-Debugging mit Webentwickler Tools (von Mozilla Firefox)

Die Entwickler Tools von verschiedenen Browsern können eine wertvolle Hilfe bei der Fehlersuche in Javascript-Funktionen sein. Im Debugger dieser Tools können Sie den Javascript Code schrittweise durchgehen und den Fehler genauer identifizieren.

Für das in [Abbildung 122](#) dargestellte Formular sei die Funktion `checkIntegerDifference()` hinterlegt und zwar als onclick-Ereignis des Buttons *Berechnung*. In dieser Funktion weist einen Syntaxfehler auf, und zwar ist das in [Abbildung 122](#) markierte Statement mit Komma anstelle von Semikolon abgeschlossen worden.

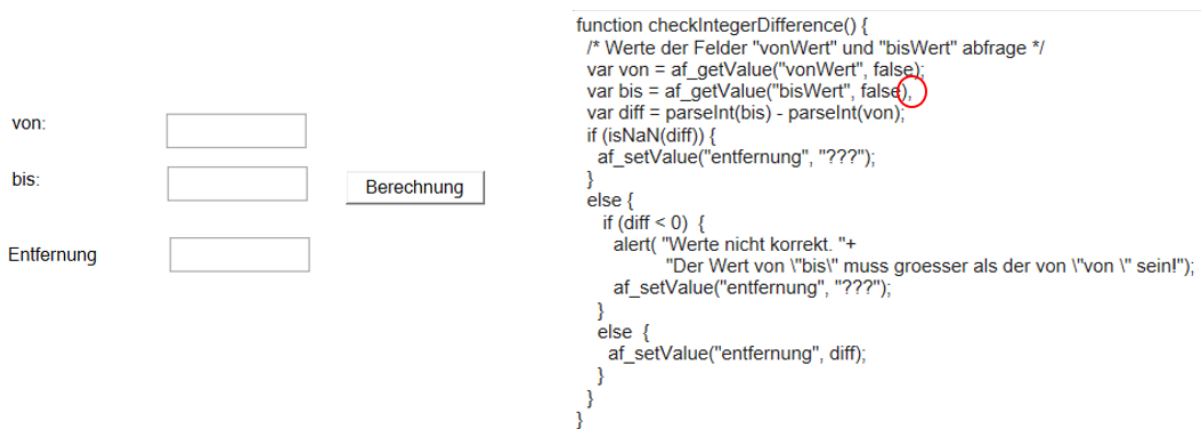


Abbildung 122: Webformular mit JavaScript Funktion

Dieser Fehler wird während der Ausführung des Prozesses abgefangen und eine entsprechende Fehlermeldung angezeigt. Zur Laufzeit führt dann ein Klick auf den Button Berechnung nicht zum gewünschten Ergebnis, da das Script nicht ausgeführt wird. Bei dieser kleinen Funktion würde man

auch auf Basis der Fehlermeldung den Fehler rasch entdecken, bei längeren JavaScript-Funktionen ist die Lokalisierung der richtigen Stelle im Programm-Code oftmals jedoch deutlich schwieriger.

Hierfür kann man beispielsweise Webentwickler Tool von Mozilla Firefox verwenden, in dem man die dem Webformular zugrundeliegende HTML-Seite in den Webbrowser lädt und analysiert. Hierzu geht man wie folgt vor:

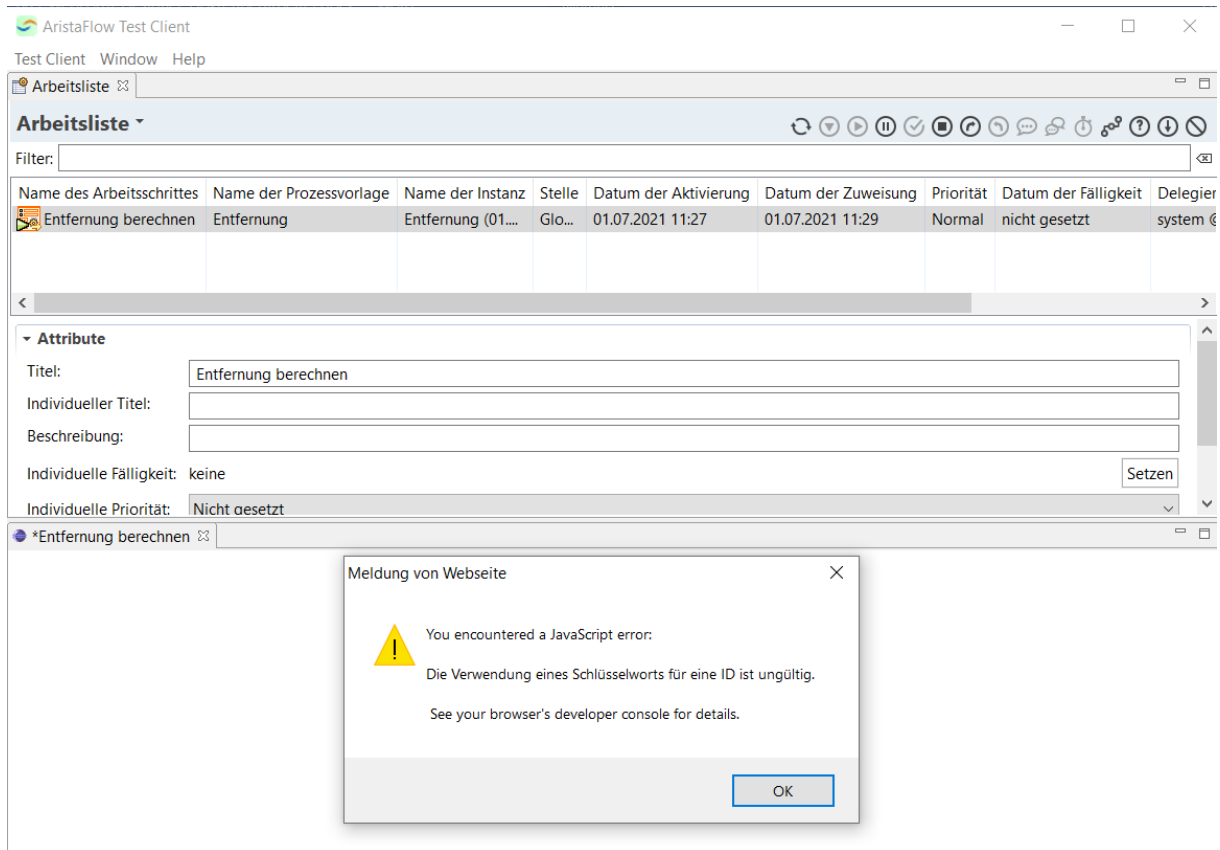


Abbildung 123: Laufzeit-Fehlermeldung

1. Man bestätigt die Fehlermeldung und klickt anschließend mit der rechten Maustaste in das im Client oder Test Client angezeigte Formular und wählt im Kontextmenü den Eintrag *Eigenschaften* (Abbildung 124).
2. Man startet Firefox und öffnet Webentwickler Tools.
3. Die im Eigenschaften-Fenster angezeigte URL (Abbildung 124) kopiert man in Adresszeile von Firefox und löst das Laden der Webseite aus.
4. Die Fehlermeldung wird nochmals angezeigt und bestätigt. Klickt man im Console-Fenster auf die Fehlermeldung, öffnet sich im Debugger-Fenster der Quellcodebereich mit dem markierten Fehler. (Abbildung 125)
5. Nun kann man im Script Haltepunkte setzen und es schrittweise ausführen oder untersuchen.

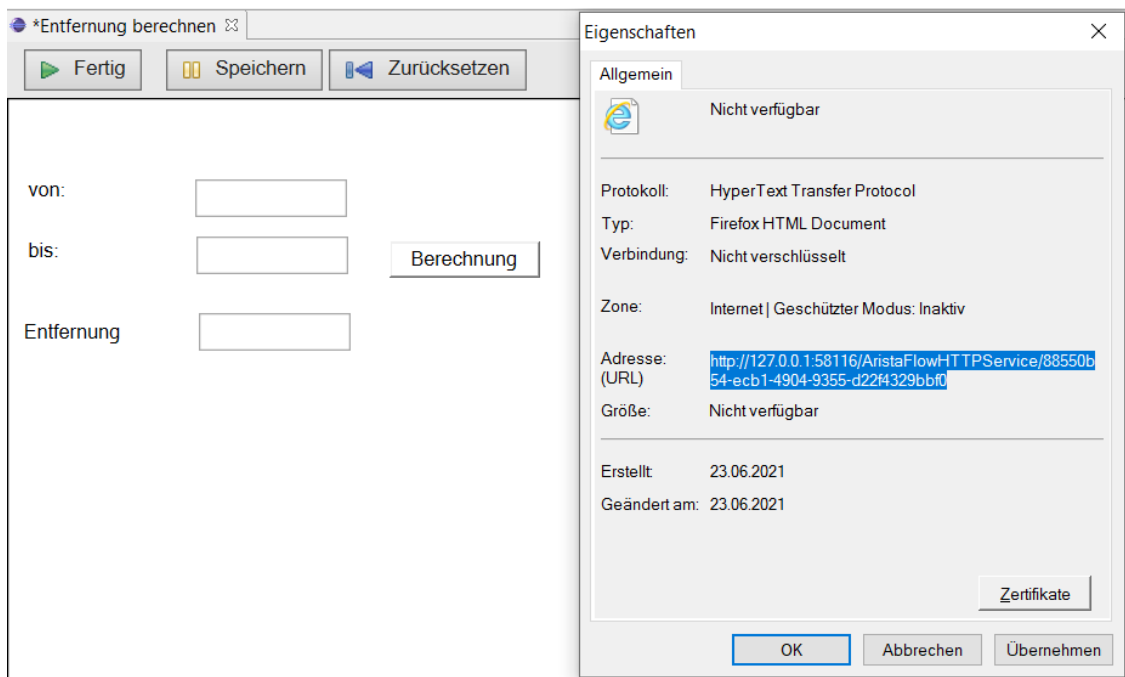


Abbildung 124: Eigenschaften-Fenster mit URL

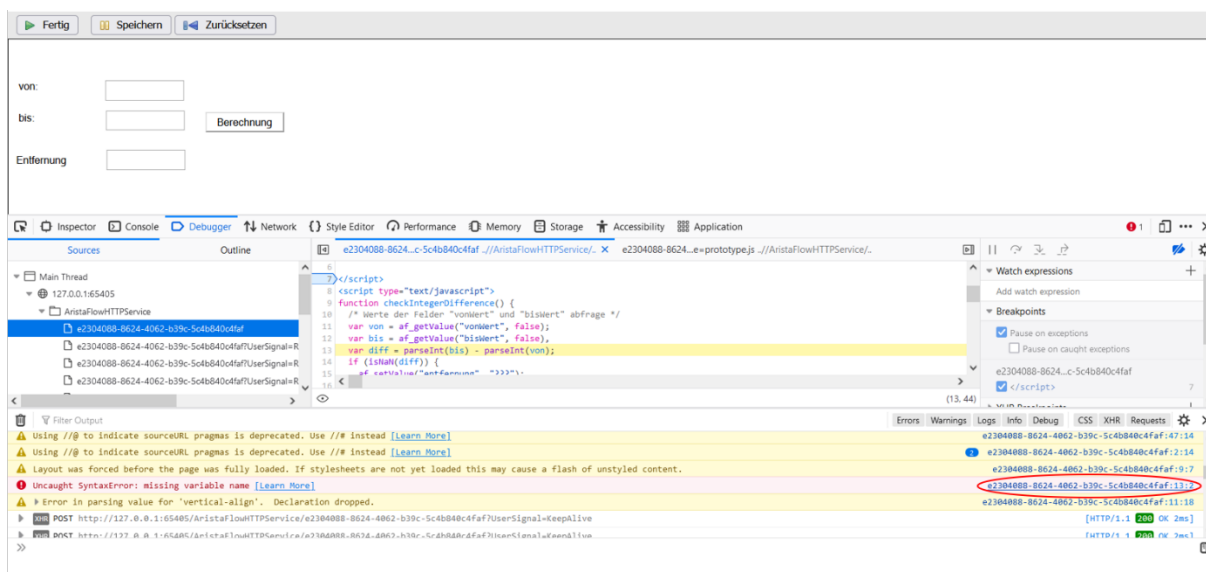


Abbildung 125: Debugger Fenster von Firefox

10 Schlussbemerkungen

Dieses Dokument ist als Einführung gedacht und kann daher nicht alle Aspekte und Möglichkeiten des WebForm-Designers zeigen.

Insbesondere natürlich nicht welche Erweiterungen bei Bedarf realisiert werden können. Sie kann daher eine eingehende Schulung bei der AristaFlow GmbH nicht ersetzen.

11 Anhang: Beispieltabellen der *legotrailerdb*

kunden			
kdnr	kname	kdstadt	bonitaet
[PK] integer	character varying (30)	character varying (30)	Integer {-2, -1, 0, 1, 2}

Tabelle 7: *kunden*-Tabelle

auftraege		
auftrnr	Kdnr	auftrdatum
[PK] integer	Integer	datum

Tabelle 8: *auftraege*-Tabelle

auftragspos				
auftrnr	pos	Teileid	farbe	anzahl
[PK] integer	[PK] integer	character varying (8)	integer	integer

Tabelle 9: *auftragspos*-Tabelle

preisliste		
teileid	Farbe	preis
[PK] character varying (8)	[PK] integer	numeric (8,2)

Tabelle 10: *preisliste*-Tabelle

teiletypen	
teileid	teilename
[PK] character varying (8)	character varying (8)

Tabelle 11: *teiletypen*-Tabelle