

Intrex Professional

RELEASE 5.2



Inside Intrex

Inhaltsverzeichnis

1. Systemarchitektur	6
1.1. Wichtige Begriffe und Techniken	6
1.2. Verzeichnisstruktur	6
1.2.1. Hauptverzeichnisse	6
1.2.2. Verzeichnisaufbau eines Portals	7
1.2.3. External	7
1.2.4. Internal	8
1.3. Application.....	8
1.4. Die Datei api.js	9
2. JavaScript	9
2.1. Allgemeines	9
2.2. JavaScript-Objekte	10
2.3. Erste Schritte.....	10
2.3.1. Erster Schritt: Statische Anzeige.....	10
a) Übung 1	10
b) Übung 2.....	12
c) Übung 3.....	13
2.3.2. Zweiter Schritt: dynamische Anzeige.....	13
2.4. If-Bedingung (WENN-DANN-Bedingung)	15
2.5. Datumsberechnung.....	17
2.5.1. Datumsberechnung mit festem Intervall	17
2.5.2. Datum in zweites Datumsfeld übernehmen	18
2.5.3. Datumsberechnung mit variablem Intervall	19
2.5.4. Differenz zweier Datumsfelder berechnen	19
2.6. Rechenoperationen	20
2.6.1. Berechnung zweier Zahlen mit statischem Operator	20
2.6.2. Berechnen zweier Zahlen mit dynamischem Operator.....	21
2.6.3. Berechnen mehrerer Felder.....	22
2.7. Ein- und Ausblenden.....	23
2.7.1. Gruppen dynamisch ein- und ausblenden	23
2.7.2. Schaltflächen dynamisch ein- und ausblenden	24
2.8. Mit Skript einen Klick auf eine Schaltfläche auslösen.....	25
3. Aufruf von Java-Klassen	27
3.1. Informationen über den angemeldeten User	27
3.2. Request-Werte	29
3.2.1. Request mit festem Parameter	29
3.2.2. Request mit variablem Parameter.....	30
3.2.3. Cookies setzen und auslesen	31
3.3. Datensatz kopieren.....	32
3.4. Bedingte Anzeige	32
4. Optionen für Experten	34

4.1. JSONObject	34
5. VM-Include.....	37
5.1. Was ist Velocity?	37
5.2. Verwenden der JavaKlasse zum Ausführen der Queries	37
5.3. Prepared Query	37
5.4. Methoden der Klasse PreparedStatement	37
5.5. Methoden der Klasse DbPreparedStatement.....	38
5.6. Methoden der Klasse DbResultSet	39
5.7. Anzahl der Benutzer ermitteln.....	39
5.8. Tabelle mit Usern anzeigen	40
6. Ajax 42	
6.1. Grundlagen	42
6.2. Warum Ajax ?.....	42
6.3. Ajax in Intrexx mit JSON Response.....	43
6.3.1. Beispiel 1 – Hallo Welt.....	43
a) Request an den Server via upSimpleAjax	43
b) Verarbeitung des Requests auf dem Server	43
c) Ausgabe des Ergebnisses	43
d) Request an den Server via Response	43
e) Verarbeitung des Requests auf dem Server.....	44
f) Ausgabe des Ergebnisses	44
6.3.2. Beispiel 2 – Summierung	44
6.3.3. Aufbau der Applikation.....	44
6.3.4. Summierung der Preise.....	46
6.4. Ajax in Intrexx mit XML-Antwort.....	48
6.4.1. Beispiel 1 – Hallo Welt.....	48
6.4.2. Request an den Server via ActionController.....	48
6.4.3. Verarbeitung des Requests auf dem Server.....	49
6.4.4. Ausgabe des Ergebnisses	49
7. Anhang - Velocity-Referenz.....	50
7.1. Variablen	50
7.2. Properties	50
7.3. Methoden.....	50
7.4. Kommentare.....	51
7.5. #set.....	51
7.6. #if / #else / #elseif.....	53
7.6.1. #if 53	
7.6.2. #else	53
7.6.3. #elseif	53
7.7. Relationale und logische Operatoren	53
7.8. #foreach	54
7.9. #include.....	54
7.10. #parse	54


7.11. #macro	54
7.12. Range Operator.....	55
7.13. Aufruf einer Seite (2. Transformation)	55
8. Anhang - ValueHolder.....	56
8.1. Typen von Valueholdern.....	57
8.1.1. BooleanValueHolder	57
8.1.2. DateTimeValueHolder	57
8.1.3. DoubleValueHolder	57
8.1.4. FileValueHolder.....	57
8.1.5. ImageValueHolder	57
8.1.6. LongValueHolder.....	58
8.1.7. StringValueHolder	58
8.2. Spezielle Valueholder.....	58
8.2.1. NewGuidValueHolder	58
8.2.2. NullValueHolder.....	58
8.2.3. NowValueHolder.....	58
9. Anhang - Renderer	58
9.1. Basisfunktionalitäten von Renderer.....	58
9.2. Typen von Renderer	59
9.3. Erzeugen von Renderern im Velocity Context	60
9.4. Zugriff auf Defaultrenderer.....	60
9.4.1. \$DefaultHtmlEncodingRenderer	60
9.4.2. \$DefaultJsHtmlEncodingRenderer	60
9.4.3. \$DefaultHtmlEncodingRendererEdit	60
9.4.4. \$DefaultSimpleTextRenderer.....	60
9.4.5. \$DefaultDateTimeRenderer.....	60
9.4.6. \$DefaultDateRenderer.....	60
9.4.7. \$DefaultTimeRenderer	61
9.4.8. \$DefaultIntegerRenderer.....	61
9.4.9. \$DefaultCurrencyRenderer	61
9.4.10. \$DefaultNumberRenderer	61
9.4.11. \$DefaultBooleanRenderer	61
9.4.12. \$SimpleTextRenderer.....	61
9.4.13. \$DefaultJsIntegerRenderer	61
9.4.14. \$DefaultBooleanAsIntRenderer	61
9.4.15. \$DefaultJsNumberRenderer	61
9.4.16. \$DefaultJsEncodingRenderer.....	61
9.4.17. \$DefaultTooltipRenderer	62
9.5. Direktes Erzeugen eines Renderers.....	62
9.5.1. Erzeugen eines Datumsrenderers	62
9.5.2. Erzeugen eines Datumsrenderers mit Formatierung	62
9.5.3. Erzeugen eines komplexen Renderers	63
9.5.4. IRenderer createDefaultUriRenderer().....	64



Copyright






Das vorliegende Dokument ist in all seinen Teilen urheberrechtlich geschützt. Alle Rechte sind vorbehalten, insbesondere das Recht der Übersetzung, des Vortrags, der Reproduktion und der Vervielfältigung. Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen.

Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Schreibkonventionen

In diesem Dokument werden Textstellen *kursiv* dargestellt, wenn sie sich auf Einstellungen in den abgebildeten Dialogen beziehen. Menüpunkte, die in Kontextmenüs erreichbar sind, sind immer auch über das Hauptmenü erreichbar. Hauptmenüpunkte werden nicht beschrieben, es sei denn, sie sind nicht über das Kontextmenü erreichbar. Eine Beschreibung der allgemeinen Hauptmenüpunkte finden Sie im Handbuch  *Portale*. Programmiercode im Text wird in der Schriftart `Courier` dargestellt. Kontextmenüs können mit einem Klick mit der rechten Maustaste auf das beschriebene Element geöffnet werden.

<intrex> bezeichnet im Folgenden Ihren Intrex Installationspfad, unter Windows z.B.  `c:\programme\intrex\`, unter Linux z.B.  `/opt/intrex/`. Folgende Symbole werden für die Kennzeichnung von speziellen Informationen verwendet:

-  Informationen
-  Verweise auf ein Intrex Handbuch
-  Verzeichnisse
-  URLs
-  Klick auf Schaltflächen

1. Systemarchitektur

Der Intrexx Portal Manager kommuniziert mit dem Intrexx Application Server via SOAP auf der Basis von Apache Axis. SOAP ist ein XML basierendes Protokoll zur Kommunikation von Systemen in verteilten Umgebungen.

Der Manager erzeugt die Applikationen, deren Basisformat XML ist, durch eine Transformation via XSL in das Velocity Markup Format im gewünschten Ausgabeformat. Durch die Transformation können schnell neue Ausgabemedien unterstützt und flexibel auf Änderungen in den Kommunikationsformen reagiert werden. Über die Verwendung von Velocity Markup ist eine hohe Performance sichergestellt.

Der Intrexx Application Server übernimmt die Aufgaben

- Benutzerverwaltung und -authentifizierung
- Sessionmanaging
- Integration der Businesslogiken
- Zugriff auf die Datenbanken.

Über Konnektoren (im Lieferumfang enthalten) können verschiedene Webserver wie Apache, Tomcat und Microsoft Internet Information Server unterstützt werden.

1.1. Wichtige Begriffe und Techniken

GUID

Global Unique Identifier – weltweit eindeutige Nummer

Velocity

Javabasierte Template Engine, die es ermöglicht, in statische Webseiten dynamische Daten über Java-Klassenaufrufe einzubinden.

1.2. Verzeichnisstruktur

1.2.1. Hauptverzeichnisse

Der Verzeichnisbau von Intrexx ist in folgende Hauptbereiche gegliedert:

📁 adapter	Dateien für z.Z. Exchange-Adapter
📁 bin:	ausführbaren Dateien
📁 cfg:	allgemeine Konfigurationsdateien
📁 client:	Dateien, die für den Portalmanager benötigt werden
📁 derby:	Dateien der Derby-Datenbank
📁 docs:	Intrexx Dokumentation
📁 export:	Portalexporte
📁 groovy:	Groovydateien
📁 help:	Intrexx Onlinehilfe
📁 installer:	Intrexx Installationsdateien
📁 jre:	Java Runtime Umgebung für Intrexx
📁 jre:	Programm Source Code
📁 jre:	Logdateien Portale

lib:	alle angelegten Portale
log:	Portalvorlagen
org:	Ressourcen für Setup und Portal Manager
orgtempl:	Temporäre Dateien
res:	Dateien, die der Tomcat Webserver benötigt
tmp:	ZIP-Files von durchgeführten Softwareaktualisierungen
tomcat:	Sourcen für Online-Update
update:	
upplib:	

1.2.2. Verzeichnisaufbau eines Portals

Ein Portal ist immer in die Hauptbereiche *Internal* und *External* unterteilt.

- 📁 *Groovy*: Groovy-Skripte aus z.B. Prozessen
- 📁 *Lib*: portaleigene Javaklassen
- 📁 *Log*: Logdateien des Portals

1.2.3. External

Das Verzeichnis *External* enthält alle Dateien, die über den Webserver direkt erreicht werden können. Dazu gehören:



Cascading Stylesheet Dateien (Standard und über den Portaldesigner erzeugte)



Dateien, die über den Link aufgerufen werden sollen (keine Rechteprüfung!)




Dateien, die über dieses Verzeichnis im Browser aufgerufen werden, unterliegen nicht der Rechtestruktur von Intrexx.



JavaScript Dateien, die von Intrexx eingesetzt werden. Ist JavaScript in Applikationen eingesetzt, so wird in diesem Verzeichnis automatisch ein Verzeichnis mit dem Namen der GUID der Applikation erstellt, der das JavaScript enthält.



Änderungen an der JavaScript-Datei in diesem Verzeichnis werden beim nächsten Speichern der Applikation überschrieben. Ändern Sie das JavaScript bitte nur im Modul *Applikationen* oder kopieren Sie den Inhalt im Modul *Applikationen* wieder in den Skripteditor.

Das Verzeichnis  *External/HTMLRoot/Include/Custom* enthält die Datei *custom.js*, die in allen Seiten von Intrexx eingebunden ist. In dieser Datei können Sie individuelle Anpassungen aufnehmen. Die Datei wird nicht durch ein Update überschrieben.



Dateien von Drittanbietern, deren Applikationen von Intrexx verwendet werden. Aktuell liegen hier die Dateien des FCK-Editors, einem freiem HTML-Editor, der für Longtextfelder aktiviert werden kann.

userfiles

Dieses Verzeichnis enthält Bilder, Flash-Dateien usw., auf die der FCK-Editor zugreifen kann.

WEB-INF

Konfigurationsdateien für den Tomcat Webserver.

Stammverzeichnis External

Startdateien für das System und HTML-Dateien, die bei Fehlermeldungen eingesetzt werden.

1.2.4. Internal

Das Verzeichnis *Internal* enthält alle Dateien, die (z.B. zur Verarbeitung von Anfragen) direkt im Intrexx Server abgearbeitet werden.

cfg

Konfigurationsdateien des jeweiligen Portals.

files

Enthält Dateien, die in Applikationen hochgeladen wurden.

mailroot


Enthält E-Mails als eml-Datei


Statistics

Log-Dateien für die Statistik

System

Dateien für spezielle Systemaufgaben (z.B. Terminserien, Sprachverwaltung)

Im Verzeichnis  *internal/system/vm/html/include* liegen benutzerdefinierte Velocity-Dateien, die im ganzen Portal aufrufbar sind.

Unter  *Vm/html/actioncontrols (controls.xml)* befinden sich die Zusatzkontrollen, die im Designer zugeordnet werden können. Über die Anpassung der XML Datei können eigene Zusatzkontrollen im Dialog aufgenommen werden.

uploadfiles

Zwischenspeicher für das Hochladen von Dateien.



Der Webserver benötigt auf diesen Ordner schreibenden Zugriff!

usrimg


Enthält die Bilder der Benutzerverwaltung

1.3. Application


Das Verzeichnis *Application* enthält alle Dateien, die direkt mit Applikationen in Zusammenhang stehen.

store

Enthält die Applikation. Änderungen in Images, Download und Resource sind möglich.

 workingstore
Zwischenstände der Applikationen

1.4. Die Datei `api.js`

Im Verzeichnis  `External/htmlroot/include` finden Sie die im Folgenden beschriebenen JavaScript-Dateien, die Definitionen und Methoden für die Anpassung von Intrexx enthalten.

`api.js`

Die Datei `api.js` enthält hilfreiche Methoden für den Umgang mit Werten.

2. JavaScript

2.1. Allgemeines

Mit JavaScript lässt sich die Funktionalität von Intrexx Applikationen erweitern. Im Eigenschaftendialog von Seiten und Eingabeelementen kann eine Funktion über den Reiter *Skript* einem Event zugeordnet werden. Auf diese Weise können sehr komfortabel Benutzerhinweise eingeblendet oder Eingaben geprüft werden.

Die Eventtypen entsprechen den spezifischen Eigenschaften des jeweiligen Elements. So bietet z.B. eine Schaltfläche das Event *onclick* und *ondblclick*, während bei anderen Elementen andere Events zur Verfügung stehen.

So können z.B. bei Applikationsseiten folgende Events verwendet werden:

<code>onload</code>	wird ausgeführt beim Laden der Seite
<code>onunload</code>	wird ausgeführt beim Schließen der Seite
<code>onsubmit</code>	wird ausgeführt beim Absenden des Formulars

Bei Eingabekontrollen stehen folgende Events zur Verfügung:

<code>onclick</code>	wird ausgeführt beim Klicken
<code>ondblclick</code>	wird ausgeführt beim Doppelklicken
<code>onkeypress</code>	wird ausgeführt, wenn eine Taste gedrückt und gehalten wird
<code>onkeydown</code>	wird in einem aktivierten Feld beim Drücken einer Taste ausgeführt
<code>onkeyup</code>	wird ausgeführt beim Loslassen einer Taste
<code>onfocus</code>	beim Aktivieren eines Feldes
<code>onblur</code>	beim Verlassen eines Feldes
<code>onchange</code>	beim Ändern eines Feldes
<code>onselect</code>	beim Selektieren von Text

Für Gruppierungen stehen folgende Events zur Verfügung:

<code>Ontouchstart</code>	mobile Endgeräte: wird bei Berührung ausgeführt
<code>Ongesturestart</code>	mobile Endgeräte: wird bei Wischgesten ausgeführt
<code>onclick</code>	wird ausgeführt beim Klicken
<code>ondblclick</code>	wird ausgeführt beim Doppelklicken

Da JavaScript clientseitig ausgeführt wird und kein Zugriff auf den Server erfolgt, sind der direkte Zugriff auf die Datenbank und der Aufruf von Javaklassen nicht möglich.

2.2. JavaScript-Objekte

In Intrexx sind alle Kontrollen objektorientiert aufgebaut. Zusätzlich zur normalen HTML-Kontrolle existiert ein spezifisches eigenes *United Planet Objekt*.

Jede Html-Kontrolle enthält die zusätzliche Methode *.oUp*, über die das UP-Objekt angesprochen werden kann. Diese Objekte enthalten zusätzliche Methoden, wie z.B. eine Methode, die aus einer eingegebenen Zahl die Formatierung entfernt und eine Zahl liefert, mit der Berechnungen durchgeführt werden können.

2.3. Erste Schritte

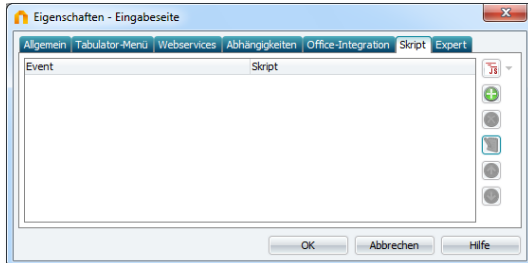
2.3.1. Erster Schritt: Statische Anzeige


a) Übung 1

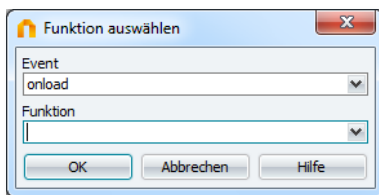


Erstellen Sie eine neue Applikation auf Basis der Vorlage *leere Applikation* mit dem Titel *Erste Schritte*. Beim Aufruf der Eingabeseite soll ein Dialogfenster erscheinen, in dem der Inhalt *Hallo Welt!* ausgegeben wird.

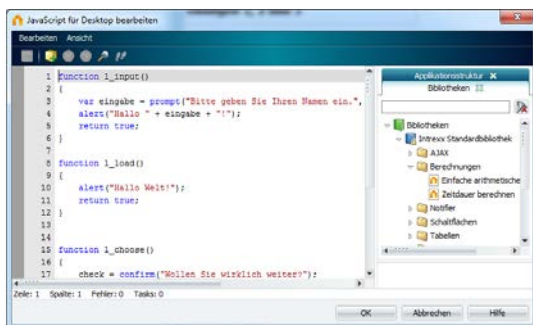
Dazu öffnen Sie den Eigenschaftendialog der Eingabeseite in der Datengruppe und wechseln im Eigenschaftendialog auf den Reiter *Skript*.




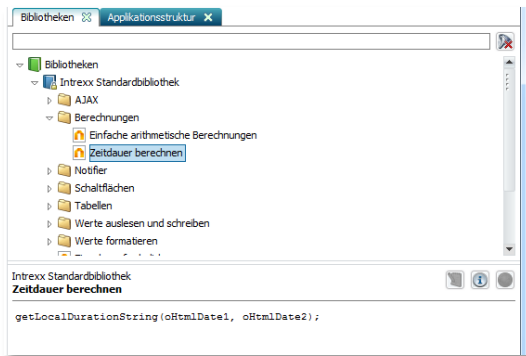
Mit  *Skriptaufruf hinzufügen* wird ein Dialog geöffnet, in dem die Events, denen Skriptfunktionen zugeordnet werden, aufgelistet sind.



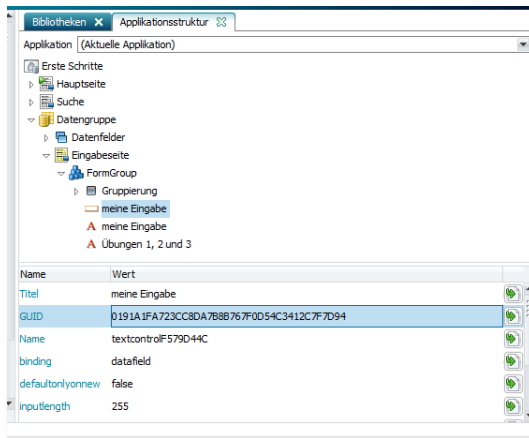
Event und bestehende Funktionen werden über Listen ausgewählt.




Um Skript zu verfassen, klicken Sie auf dem Reiter *Skript* auf  *Skript*. Diese Schaltfläche öffnet den Skript-Editor.






Im rechten Teil des JavaScript-Editors befinden sich die Bibliotheken. Dort sind JavaScripts in verschiedenen Kategorien hinterlegt und können in das Skriptfenster per Doppelklick übernommen werden. Neben der bereits hinterlegten Intrex Standardbibliothek lassen sich weitere Bibliotheken mit eigenen Skripten hinterlegen.



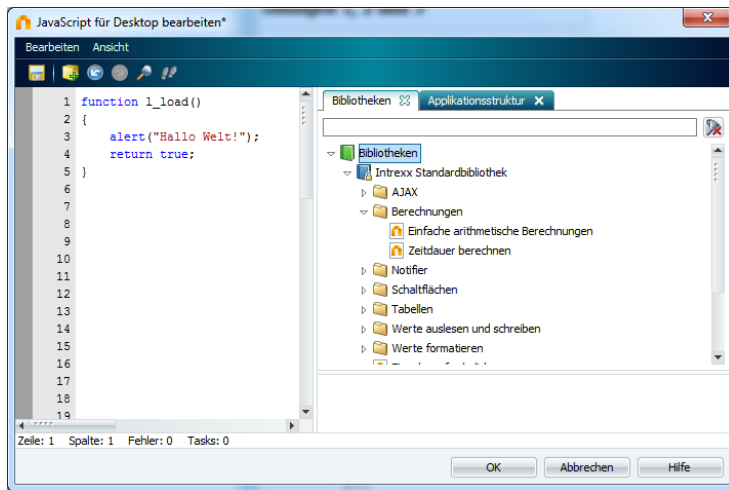
Im Bereich *Applikationsstruktur* kann auf alle Informationen einer Applikation (Titel, GUID, Name etc.) angesehen und über das Symbol  an die aktuelle Stelle im Skript eingefügt werden.

Jedes Skript beginnt mit seinem Funktionsnamen. Schreiben Sie *function* und dahinter den Namen, unter dem das Skript aufgerufen wird. Ein Funktionsname wird immer mit *()* abgeschlossen.

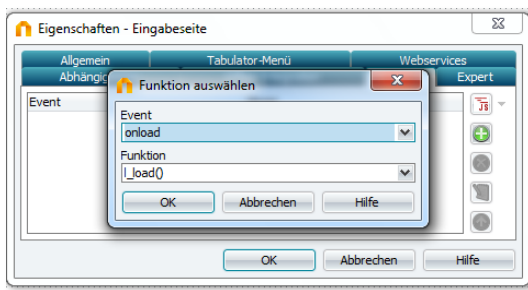
Der Bereich, in dem das Skript ausgeführt wird, wird durch zwei geschweifte Klammern definiert. Innerhalb dieser Klammern stehen die Befehlszeilen, wobei jede Befehlszeile mit einem Semikolon abgeschlossen wird. In unserem Fall soll eine Ausgabe erscheinen, die der Benutzer mit  *OK* bestätigen soll.

Die Ausgabe erfolgt über die Funktion *alert()*. Die Funktion übergibt als Parameter den gewünschten Text. Sie wird mit einem Semikolon abgeschlossen. Die Alert-Funktion gibt die Textmeldung mit Klick auf  *OK* automatisch aus. Nachdem der Benutzer die Meldung *Hallo Welt* mit  *OK* bestätigt hat, soll die Eingabeseite aufgerufen werden. Dies bewirkt die abschließende Befehlszeile *return true;*. Ihre Funktion sollte nun wie folgt aussehen:

```
function l_load()
{
    alert("Hallo Welt!");
    return true;
}
```

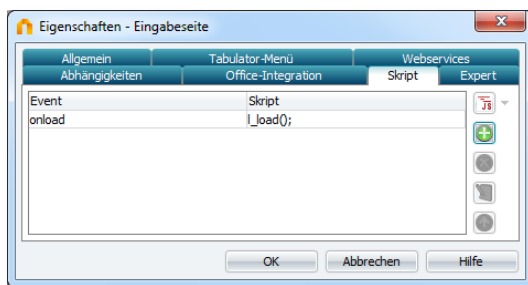


Speichern Sie die Funktion mit Klick auf die Schaltfläche OK.



Ordnen Sie die Funktion *l_load()* nach dem Speichern dem *onload*-Event zu.

Speichern Sie mit OK ab.



Speichern Sie nun die Applikation und testen Sie Ihr erstes Skript.

b) Übung 2



Nun fügen wir eine Schaltfläche auf der Eingabeseite ein. Ein Klick auf diese Schaltfläche soll eine Abfrage erzeugen, mit der der Benutzer gefragt wird, ob er auf die nächste Seite springen möchte oder nicht.

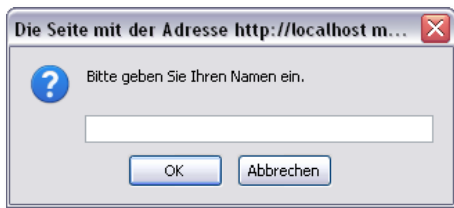
Dazu setzen wir die *confirm()* -Funktion ein, die automatisch eine Meldung mit einer Schaltfläche *OK* und einer Schaltfläche *Abbrechen* ausgibt. Erstellen Sie eine Schaltfläche *Weiter* mit der Aktion *Speichern* und Sprung auf die Hauptseite.

Wechseln Sie in den Skript-Editor. Erstellen Sie unterhalb des ersten Skripts ein weiteres Skript mit dem Funktionsnamen *l_choose()* für das *onclick*-Event der Schaltfläche *Weiter*. In der JavaScript-Funktion *confirm()* wird als Parameter der Ausgabertext des Popup-Fensters

geschrieben. Die Funktion wird in die Variable *check* geschrieben. Die Variable wird in einer If-Bedingung auf Wahrheit geprüft. If-Bedingungen werden später detailliert erläutert. Übernehmen Sie das Skript wie folgt und tragen Sie es im *onclick*-Event ein:

```
function l_choose()
{
  var check = confirm("Wollen Sie wirklich weiter?");
  if(check == true)           //Ok-Schaltfläche wurde geklickt
    return true;
  else                       //Abbrechen-Schaltfläche wurde geklickt
    return false;
}
```

c) Übung 3



In dieser Übung soll ein Fenster geöffnet werden, das eine Eingabe des Benutzers aufnimmt und weiter verarbeiten. Erstellen Sie dazu eine Schaltfläche mit der Aufschrift *Begrüßung*.

Dazu wird die `prompt()`-Funktion genutzt, welche eine Meldung erzeugt, die einen Text und ein Textfeld enthält, in das der Benutzer eingeben kann machen kann. Im Skript-Editor erstellen Sie ein neues Skript mit dem Namen *l_input()* für das *onclick*-Event der Schaltfläche *Begrüßung*. In der Funktion *prompt()* wird als Parameter der Ausgabertext und ein Text, der in dem Eingabefeld erscheinen soll erwartet.

Die Variable wird in einem weiteren Ausgabefenster wieder angezeigt, kann in einer komplexeren Funktion auch zu einer weiteren Bearbeitung verwendet werden.

```
function l_input()
{
  var eingabe = prompt("Bitte geben Sie Ihren Namen ein.", "");
  alert("Hallo " + eingabe + "!");
  return true;
}
```

2.3.2. Zweiter Schritt: dynamische Anzeige

In einem weiteren Skript soll beim Speichern der Wert eines Feldes in einem Dialog-Fenster angezeigt werden.

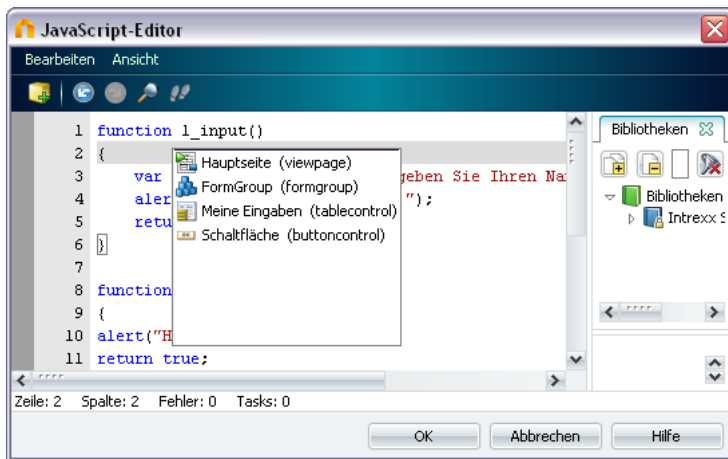
Übung


Erstellen Sie in der Applikation *Erste Schritte* eine Eingabekontrolle mit dem Titel *meine Eingabe* auf der Eingabeseite. Erstellen Sie darunter eine weitere Schaltfläche mit dem Titel *Speichern* und der Aktion *Speichern*. Wechseln Sie in den Skripteditor und erstellen Sie eine neue Funktion mit dem Namen *l_anzeige()* (geschweifte Klammern nicht vergessen!).

Der Text soll aus dem Eingabefeld ausgelesen und angezeigt werden. Dazu definieren wir das Eingabefeld mit dem Ausdruck `var htmlWert` als Variable. *htmlWert* ist ein beliebiger

Name für eine Variable. Beachten Sie, dass keine Umlaute, Sonderzeichen oder Funktionsnamen als Variablennamen verwendet werden dürfen. Zusätzlich muss die Klein-Großschreibung beachtet werden.

Nun muss die Variable einer Referenz auf das Eingabefeld zugeordnet werden. Klicken Sie dazu mit der rechten Maustaste in den Skripteditor. In einer Liste werden alle Eingabe- und Ansichtselemente, ActionControls, Formen und Gruppen der Eingabeseite angezeigt, also alle Elemente, die sich auf der Seite befinden. Die Elemente sind alphabetisch sortiert und mit dem entsprechenden Elementsymbol gekennzeichnet um Verwechslungen zu vermeiden.



 Jedes Objekt besitzt eine eindeutige GUID. Die GUID bezeichnet kein Datenfeld, sondern das jeweilige Eingabe- oder Ansichtselement. Mit Skript kann nicht auf die Datenbank zugegriffen werden, sondern lediglich die *Form* der Seite ausgelesen werden.

Wählen Sie nun aus der Liste das Eingabeelement *meine Eingabe* (textcontrol) aus. Damit ist die Referenz auf das Eingabefeld hergestellt und der Variablen zugeordnet.

```
htmlWert = getElement("13D9...32A4"); /*Meine Eingabe textcontrol*/
```

Die Eigenschaften von Ansichts- und Eingabeelementen variieren pro Typ. Beim Eingabe und Ansichtsfeld kann der aktuelle Wert mit `Browser.getValue()` ausgelesen werden. Der Wert wird in unserem Beispiel ermittelt durch den Ausdruck `Browser.getValue(htmlWert)`. Wieder kann die Funktion `alert()` für die Anzeige des Wertes eingesetzt werden. Das vollständige Skript sollte nun wie unten abgebildet aussehen, wobei die GUID durch die aktuelle GUID des Eingabefeldes in Ihrer Applikation ersetzt werden muss:

```
function l_anzeige()  
{  
  var htmlWert = getElement("84A6...A81C"); /*meine Eingabe textcontrol*/  
  alert(Browser.getValue(htmlWert));  
  return true;  
}
```

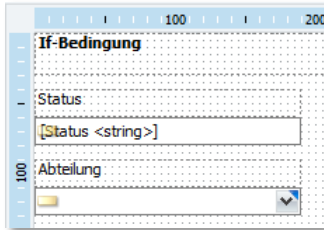
Ordnen Sie die Funktion dem *onclick*-Event der Schaltfläche *Speichern* zu. Speichern Sie die Applikation und testen sie.

2.4. If-Bedingung (WENN-DANN-Bedingung)

Im Folgenden soll ein Status geändert werden, wenn ein Wert aus einer Liste gewählt wurde. In unserem Beispiel nimmt der Status den Wert *zugewiesen* an, wenn in einer Auswahlliste eine Abteilung gewählt wurde. Wird die Wahl zurück (auf einen leeren Wert) gesetzt, soll der Status den Wert *nicht zugewiesen* annehmen.

Übung

Erstellen Sie eine neue Applikation auf Basis der Vorlage *leere Applikation* mit dem Titel *If-Bedingung*. Folgende Eingabeelemente werden auf der Eingabeseite benötigt:



- ein schreibgeschütztes Eingabefeld *Status*: Ein Doppelklick auf das Eingabefeld öffnet den Eigenschaftendialog. Setzen Sie auf dem Reiter *Allgemein* die Einstellung *Schreibgeschützt*. Wechseln Sie auf den Reiter *Optionen* und tragen Sie hier als *Feste Vorgabe* den Wert *nicht zugewiesen* ein. Beachten Sie bitte die Groß- und Kleinschreibung.
- eine Auswahlliste *Abteilung* mit den benutzerdefinierten Einträgen *Abteilung A* und *Abteilung B*. Der Anzeigewert entspricht dabei dem gespeicherten Wert



Damit beim Laden der Eingabeseite ein leerer Eintrag in der Auswahlliste als Default angezeigt wird, fügen Sie einen leeren benutzerdefinierten Eintrag hinzu und definieren ihn mit Klick auf die Schaltfläche *Default* als Startwert der Auswahlliste. Oder öffnen Sie nach der Anlage den Eigenschaftendialog mit einem Doppelklick auf die Auswahlliste und setzen Sie hier die Einstellung *Erster Eintrag der Liste ist leer*.

Mit JavaScript soll nun der aktuell gewählte Wert der Auswahlliste geprüft werden. Wird der Wert geändert, so wird der Wert des Eingabeelements *Status* umgesetzt. Wechseln Sie dazu im Eigenschaftendialog der Auswahlliste in den Skripteditor. Wir erstellen ein neues Skript mit dem Namen *function I_status()*.

Zunächst werden zwei Variablen definiert. Die erste Variable nennen wir *htmlStatus* und bilden eine Referenz auf das Eingabefeld *Status*. Die zweite Variable nennen wir *htmlAbteilung* und bilden eine Referenz auf die Auswahlliste *Abteilung*. Dann folgt die Prüfung der Bedingung.

- Die Bedingung wird eingeleitet mit *if (WENN)*.
Danach folgt in Klammern die Formulierung der Bedingung – bestehend aus Vergleichsoperator und Variable, mit der verglichen werden soll. Vergleichsoperatoren sind:

== Gleich
!= Ungleich
> Größer
>= Größer oder gleich
< Kleiner
<= Kleiner oder gleich

Eine Verknüpfung von Kriterien mit *UND* oder *ODER* erfolgt mit:

Die der Bedingung folgende Anweisung (*DANN*-Teil) kann sich über mehrere Zeilen erstrecken. In diesem Fall muss die Anweisung in geschweiften Klammern stehen. Soll auch eine Anweisung für den Fall erfolgen, dass die Bedingung nicht erfüllt ist (*SONST*-Teil), wird dieser Anweisungsblock durch *else* eingeleitet. Auch dieser Block muss, wenn er sich über mehrere Zeilen erstreckt, in geschweiften Klammern stehen.

Alternativ dazu kann die Abfrage auch in einer Zeile stehen. Dabei wird die Bedingung (Element-Operator-Vergleich - z.B. *Abteilung = leer*) in Klammern gesetzt, gefolgt von *?* für den *DANN*-Teil und einem *:* für den *SONST*-Teil.

Beispiel:

```
(Browser.getValue(htmlAbteilung)) ? "nicht zugewiesen" : "zugewiesen";
```

Um das Feld Status zu beschreiben, wird im Gegensatz zum Auslesen (s.o.) nicht `Browser.getValue()`, sondern `Browser.setValue()` benötigt. Als Parameter werden das zu ändernde Feld und der Wert benötigt.

Die Bedingung wird wie folgt geprüft:

```
if(Browser.getValue(htmlAbteilung)) wenn die Abteilung keinen Wert enthält  
{  
  Browser.setValue(htmlStat, "nicht zugewiesen"); DANN weise der Statusvariablen  
  den Wert "nicht zugewiesen" zu  
  {  
    else SONST  
    {  
      Browser.setValue(htmlStat, "zugewiesen"); weise der Variablen des Status den Wert  
      "zugewiesen" zu  
    }  
  }  
}
```

Die Blöcke enthalten geschweiften Klammern (auch wenn dies bei einer einzeiligen Anweisung nicht unbedingt nötig wäre). Das Einrücken der *DANN*- und *SONST*-Bereiche erleichtert das Lesen der Funktion. Ihre Funktion sollte nun wie folgt aussehen:

```
function l_status()
{
    var htmlStat      = getElement("C8A6...45D5"); /*Status textcontrol*/
    var htmlAbteilung = getElement("D525...A6C3"); /*Abteilung dropdowncontrol*/
    if(Browser.getValue(htmlAbteilung))
    {
        Browser.setValue(htmlStat, "zugewiesen");
    }
    else
    {
        Browser.setValue(htmlStat, "nicht zugewiesen");
    }
    return true;
}
```

Alternativ dazu die abgekürzte Fassung:

```
function l_status2()
{
    var htmlStat      = getElement("C8A6419089B...45"); /*Status textcontrol*/
    var htmlAbteilung = getElement("AD525...26C3"); /*Abteilung dropdowncontrol*/
    var strErgebnis = (Browser.getValue(htmlAbteilung)) ? "zugewiesen" : "nicht zugewiesen";
    Browser.setValue(htmlStat, strErgebnis);
    return true;
}
```

Speichern Sie die Funktion und rufen Sie sie beim *onchange*-Event der Auswahlliste *Abteilung* auf. Speichern und testen Sie Ihre Applikation.

2.5. Datumsberechnung

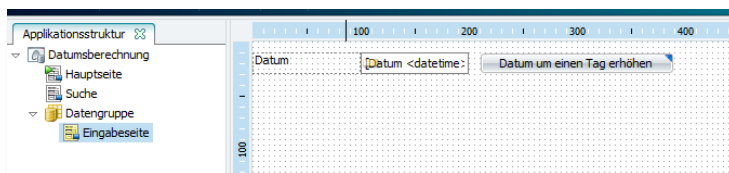
In dieser Applikation werden Beispiele für verschieden Arten der Datumsberechnung aufgezeigt. Dabei wird zu Beginn auf bestehende Funktionen zurückgegriffen, die die Arbeit erleichtern.

2.5.1. Datumsberechnung mit festem Intervall

Ein vorhandenes Datum soll mit Klick auf eine Schaltfläche um einen Tag erhöht werden.

Übung

Erstellen Sie die Applikation *Datumsberechnung* auf Basis der Vorlage *Leere Applikation*. Definieren Sie auf der Eingabeseite ein Eingabefeld mit dem Titel *Datum* und dem Datentyp *Datum*. Öffnen Sie den Eigenschaftendialog des Eingabefeldes und stellen Sie auf dem Reiter *Optionen* die Vorgabe *aktuelles Datum und Uhrzeit* ein. Legen Sie dann eine Schaltfläche mit dem Titel *Datum um einen Tag erhöhen* an.



Fügen Sie beim *onclick*-Event der Schaltfläche das folgende Skript ein:

```
function tomorrow()
{
  var htmlDate = getElement("00B8...07B1"); /*Datum datecontrol*/
  var oDateFrom = getDateObject(htmlDate); /*erstellt JavaScript DateObjekt*/

  var oDate = oDateFrom.getDate(); /*Liest den Monatstag des Objekts */
  oDate = oDate + 1; /*Tag wird hochgezählt*/
  oDateFrom.setDate(oDate); /*schreibt den Monatstag zurück*/
  writeLocalString(htmlDate, oDateFrom); /*Wert wird in lokalem Datumsformat
in html-Datumfeld zurückgeschrieben*/
  return true;
}
```

Mit der Variablen *htmlDate* wird eine Referenz auf das Eingabefeld *Datum* gebildet. Das Datum wird mit *getDateObject(htmlDate)* in ein JavaScript-Date-Objekt umgewandelt, das wiederum eigene Datumsfunktionen bereitstellt. Als nächste Definition folgt die Variable *oDate*, die den Tag des Datums mit der Methode *oDateFrom.getDate()* zurückgibt.

Die Berechnung erfolgt durch Hochzählen mit dem Intervall 1: (*oDate = oDate + 1*).

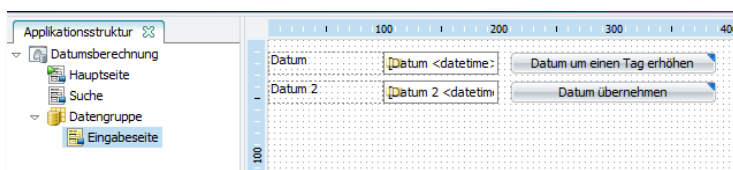
Mit der Methode *setDate(oDate)* wird der Tag in das Objekt zurück geschrieben. Dies reicht jedoch nicht aus, um den Wert im Formular anzuzeigen. Die Formatierung muss angepasst werden. Dazu bedienen wir uns der Funktion *writeLocalString()*, die wir dem Wert von *htmlDate* wieder zuweisen. Bei Klick auf die Schaltfläche im Browser wird das eingegebene Datum um einen Tag erhöht.

2.5.2. Datum in zweites Datumfeld übernehmen

Der Wert des eben erstellten Datumfeldes wird nun in ein weiteres Feld geschrieben. Diese Funktion kommt z.B. bei der Kalender- und der Ressourcenapplikationen zum Einsatz, wenn im Datumfeld *Bis* der bereits veränderte Wert des Datumfeldes *Von* angezeigt werden soll.

Übung

Erstellen Sie ein weiteres Eingabefeld *Datum 2* mit Datentyp *Datum*. Rechts neben dem Eingabefeld legen Sie eine Schaltfläche mit dem Titel *Datum übernehmen* an. Mit dieser Schaltfläche wird beim *onclick*-Event ein Skript ausgeführt, das den Wert des ersten Eingabefeldes *Datum* in das neue Eingabefeld *Datum2* überträgt.



Erstellen Sie die Funktion *datUebernahme()*. Definieren Sie die Variablen *htmlD1* und *htmlD2*. Der Wert von *htmlD1* soll in *htmlD2* geschrieben werden. Vergleichen Sie die Funktion:

```
function datUebernahme()
{
  var htmlD1 = getElement("00BC...77B1"); /*Datum datecontrol*/
  var htmlD2 = getElement("EAFA...ACC6"); /*Datum2 datecontrol*/
  var htmlD1value = Browser.getValue(htmlD1);
  Browser.setValue(htmlD2, htmlD1value);
  return true;
}
```

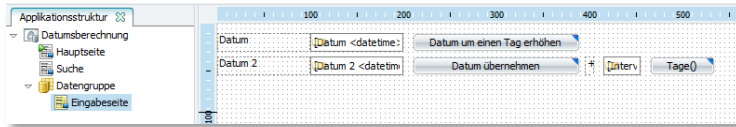
Speichern und testen Sie die Applikation.

2.5.3. Datumsberechnung mit variablem Intervall

Nun wollen wir die Operation Datumsberechnung wiederholen – mit einer kleinen Zusatzschwierigkeit. Wie in der vorletzten Übung soll der Datumswert des Eingabefeldes *Datum2* hochgezählt werden – jedoch nicht mit einem festen, sondern mit einem vom Benutzer selbst zu definierenden Intervall.

Übung

Erstellen Sie rechts neben der Schaltfläche *Datum übernehmen* ein Eingabefeld mit dem Titel *Intervall* und dem Datentyp *Ganzzahl*. Ändern Sie den automatisch erstellten Titel des Eingabefeldes ab und tragen Sie hier ein + Zeichen als Beschriftung ein.



Nun folgt eine Schaltfläche mit dem Titel *Tag(e)*, mit der im *onclick*-Event Skript ausgeführt wird. Die Funktion heißt *tomorrow2()*. Wir können dabei den größten Teil der eben erstellten Funktion *tomorrow()* übernehmen. Nun folgt der Teil des Skripts, der noch modifiziert werden muss:

date = date + Intervall

Damit mit dem Wert des Eingabefeldes *Intervall* gerechnet werden kann, muss der gelieferte *html*-Wert mit der Funktion *getNumberObject(oHtml)* in einen integer-Wert konvertiert werden.

```
function tomorrow2()
{
  var htmlDate2 = getElement("..."); /*Datum 2  datecontrol*/
  var oIntervall2 = getNumberObject(getElement("...")); /*Intervall
integercontrol*/
  var oDateFrom2 = getDateObject(htmlDate2); /*erstellt ein JavaScript
DateObjekt*/
  var oDate2 = oDateFrom2.getDate(); //liest Tag aus Datum aus //
  oDate2 = oDate2 + oIntervall2; //setzt Tag um Intervall hoch //
  oDateFrom2.setDate(oDate2); //schreibt Tag in Datumsobjekt zurück //
  writeLocalString(htmlDate2,oDateFrom2); /*formatiert Datumsobjekt in
lokales Format und schreibt es in html-Wert zurück */
  return true;
}
```

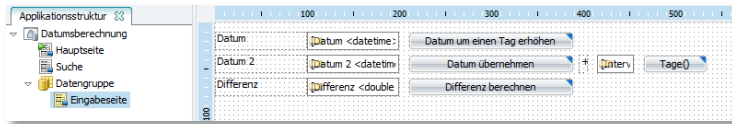
Testen Sie auch diese Applikation.

2.5.4. Differenz zweier Datumfelder berechnen

In der Applikation *Datumsberechnung* gibt es jetzt zwei Datumswerte. Die Differenz der beiden Werte in Tagen kann mit JavaScript ermittelt werden.

Übung

Erstellen Sie ein Eingabefeld *Differenz* mit dem Felddatentyp *Gleitkommazahl*. Rechts davon legen Sie eine Schaltfläche mit dem Titel *Differenz berechnen* an. Im *onclick*-Event der Schaltfläche soll die Funktion *Differenz* ausgeführt werden.



Das Skript enthält die Variablen *htmlDateFrom*, *htmlDateTo* und *htmlResult*. Diese Werte werden mit der UP-Methode *getDateObject* in ein JavaScript-Datumsformat umgewandelt. Damit kann die Differenz in Millisekunden ermittelt werden. Beim Zurückschreiben muss der Wert auf Tage (Millisekunden * 1000 * 60 * 60 * 24) umgerechnet werden. Den Wert der Differenz ermitteln wir mit folgender Funktion:

```
function differenz()
{
  var htmlDateFrom = getElement("704F...66FD"); /*Datum datecontrol*/
  var htmlDateTo   = getElement("FA98...78D8"); /*Datum2 datecontrol*/
  var htmlDiff     = getElement("E784...AD1B"); /*Diff floatcontrol*/
  var htmlDateFromValue = Browser.getValue(htmlDateFrom); /*auslesen*/
  var htmlDateToValue   = Browser.getValue(htmlDateTo); /*auslesen*/

  if(htmlDateFromValue == "" || htmlDateToValue == "") /*prüfe, ob alle Werte
                                                         gefüllt sind*/
  {
    alert("Bitte geben Sie einen Wert ein!");
    return false; /*falls nicht, Funktion abbrechen*/
  }
  else
  {
    /*in Javascript-Datumsobjekte umwandeln*/
    var oDateFrom = getDateObject(htmlDateFrom);
    var oDateTo   = getDateObject(htmlDateTo);

    if(oDateFrom > oDateTo) /*Datum 1 größer als Datum 2?*/
    {
      var diff = oDateFrom.getTime() - oDateTo.getTime(); /*Differenz in
                                                           Millisekunden herausfinden*/
    }
    else
    {
      var diff = oDateTo.getTime() - oDateFrom.getTime(); /*Differenz in
                                                           Millisekunden herausfinden*/
    }
    writeLocalString(htmlDiff, (diff/(1000*60*60*24))); /*Anzahl Tage in
                                                         Differenz-Feld schreiben*/

    return true;
  }
}
```

Testen Sie die Applikation.

2.6. Rechenoperationen

In den folgenden Rechenoperationen werden der Umgang von JavaScript mit Zahlen skizziert und Lösungen aufgezeigt.

2.6.1. Berechnung zweier Zahlen mit statischem Operator

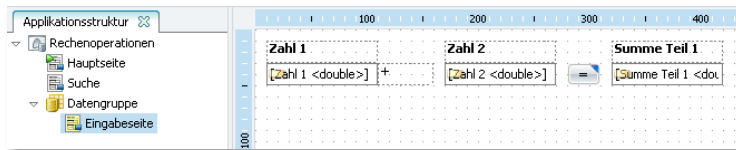
Zwei Zahlen sollen addiert werden. Die Zahlen befinden sich in Eingabefeldern des Datentyps *Währung*. Der Operator ist *+*. Mit der Funktion *calculate()* können zwei Zahlenfelder eines Formulars berechnet und das Ergebnis einem dritten Feld übergeben werden:

```
calculate(html1, html2, Operator, Ergebnis)
```

Diese Funktion gilt nur für Zahlen, die aus der Form ausgelesen werden (html-Objekte).

Übung

Erstellen Sie eine neue Applikation *Rechenoperationen* auf Basis der Vorlage *Leere Applikation*. Legen Sie drei Eingabefelder mit Datentyp *Währung* auf der Eingabeseite an: *Zahl 1*, *Zahl 2* und *Summe Teil 1*. Beschriften Sie den Zwischenraum zwischen *Zahl 1* und *Zahl 2* mit einem + Zeichen (*Statisches Textfeld*), damit die Art der Berechnung ersichtlich ist. Den Zwischenraum zwischen *Zahl 2* und *Summe Teil 1* können Sie mit einem =-Zeichen mittels einer Schaltfläche beschriften.

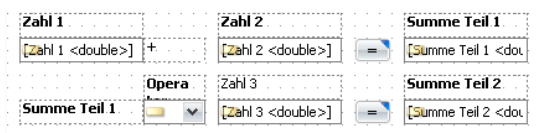


Wechseln Sie nun über Eigenschaftendialog der Schaltfläche in den Skripteditor. Erstellen Sie ein Skriptfunktion mit dem Namen *rechnen1*, die im onclick-Event der Schaltfläche aufgerufen werden soll. Das Skript enthält die Variablen für die drei Eingabefelder sowie die Funktion, die die Elemente aufruft und zurückgibt. Vergleichen Sie:

```
function rechnen1()
{
  var htmlZ1 = getElement("1390...CF96"); /*Zahl 1 currencycontrol*/
  var htmlZ2 = getElement("BF97...5159"); /*Zahl 2 currencycontrol*/
  var htmlS1 = getElement("6FA6...A83D"); /*Summe Teil 1 currencycontrol*/
  calculate(htmlZ1, htmlZ2, "+", htmlS1);
  return true;
}
```

2.6.2. Berechnen zweier Zahlen mit dynamischem Operator

In diesem Beispiel soll der Operator selbst gewählt werden. Dazu legen wir als erstes auf der Eingabeseite eine Auswahlliste mit den benutzerdefinierten Werten +, -, *, / und einem leeren Eintrag (Default) an. Der Liste folgt ein Eingabefeld *Zahl 3* mit Datentyp *Währung* und ein weiteres Zahlenfeld *Summe Teil 2*, ebenfalls Datentyp *Währung*. Berechnet werden soll *Summe Teil 1 OPERATOR Zahl3*. Das Ergebnis davon soll in das Eingabefeld *Summe Teil 2*, Datentyp *Währung*, geschrieben werden.



Hier das Ergebnis:

```
function rechnen2()
{
  var htmlSumme1 = getElement("6FA6...A83D"); /*Summe Teil 1 currencycontrol*/
  var htmlZ3      = getElement("2278...8A18"); /*Zahl 3 currencycontrol*/
  var htmlS2      = getElement("8C24...D178"); /*Summe Teil 2 currencycontrol*/
  var htmlOpera   = getElement("B2EB...E080"); /*Operator dropdowncontrol*/
  calculate(1_ htmlSumme1, htmlZ3, Browser.getValue(htmlOpera), htmlS2);
  return true;
}
```

Tragen Sie das Skript im *onclick*-Event der zweiten Schaltfläche ein. Speichern und testen Sie Ihre Arbeit.

2.6.3. Berechnen mehrerer Felder

Aufbauend auf dem Gelernten wenden wir uns einem etwas komplexeren Beispiel zu. Sie sind gut vorbereitet. Jetzt empfiehlt es sich, die Funktion Schritt für Schritt aufzubauen und immer wieder zu testen.

Übung

Aus der *Summe Teil 2* soll eine dynamische *MwSt* berechnet werden. Die *MwSt* wird als Zahl (z.B. 19) eingegeben, der *MwSt*-Betrag berechnet und in ein weiteres Eingabefeld *MwSt-Betrag* geschrieben. Der Bruttobetrag soll im Eingabefeld *Gesamtbetrag berechnen* ausgewiesen werden. Erstellen Sie folgende Felder auf der Eingabeseite:

- *MwSt* (Gleitkommazahl)
- *MwSt-Betrag* (Währung)
- *Gesamtsumme* (Währung)

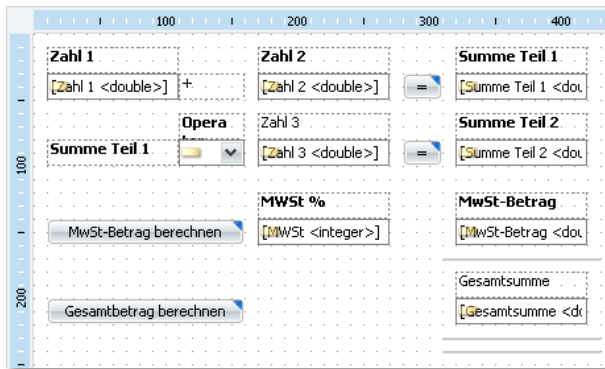
Die Schaltfläche *MwSt-Betrag berechnen* aktiviert das Skript zur Berechnung der *Mwst*, die Schaltfläche *Gesamtbetrag berechnen* das Skript zur Berechnung der *Gesamtsumme*.

Berechnung des *MwSt*-Betrages:

$Summe\ Teil\ 2 * MwSt / 100$

Berechnung des Gesamtbetrages:

$Summe\ Teil\ 2 + MwSt\ -Betrag$



In der *calculate*-Funktion dürfen nur *html*-Objekte übergeben werden – *100* ist jedoch ein Zahlenobjekt. Da nur mit Zahlen gerechnet werden kann, müssen die *html*-Objekte zunächst in Zahlen umgewandelt werden. Wir wandeln das *html*-Objekt *Summe Teil 2* mit *getNumberObject()* in eine Zahl um und berechnen den *MwSt*-Betrag.

Wenn die Zahl später in das Eingabefeld geschrieben werden soll, muss sie im lokalen Zahlenformat vorliegen. Dies bewerkstelligen wir mit der Methode *writeLocalString()*.

```
function mwstBetrag()  
{  
  var htmlSum2 = getElement("8C24...5D18"); /*Summe Teil 2 currencycontrol*/  
  var htmlMwst = getElement("3AF2...EE2E"); /*MwSt floatcontrol*/  
  var htmlMwstbetrag = getElement("D114...22AC"); /*MwSt-Betrag  
currencycontrol*/
```

```

    if(Browser.getValue(htmlMwst) == "")
    {
        alert("Bitte MwSt-Satz eingeben!");
    }
    else
    {
        var fltSum2 = getNumberObject(htmlSum2); /*wandelt html-Objekt in
Javascript-Zahl um*/
        var fltMwst = getNumberObject(htmlMwst); /*wandelt html-Objekt in
Javascript-Zahl um*/

        var tmpMwstbetrag = fltSum2 * fltMwst / 100;
        writeLocalString(htmlMwstbetrag, tmpMwstbetrag);
    }
    return true;
}

```

Tragen Sie die Funktion im *onclick*-Event der Schaltfläche *MwSt-Betrag berechnen* ein.

Nun folgt das nächste Skript, das aus dem gerade errechneten MwSt-Betrag und der Summe Teil 1 einen Gesamtbetrag berechnet. Da alle Werte als html-Wert vorliegen, können wir den Gesamtbetrag mit Hilfe der Funktion *calculate()* berechnen.

```

function gesamtSumme()
{
    var htmlSumme      = getElement("8...C"); /*Summe Teil 2  currencycontrol*/
    var htmlMwStBetrag = getElement("D...1"); /*MwSt-Betrag  currencycontrol*/
    var htmlGesSumme   = getElement("7...9"); /*Gesamtsumme  currencycontrol*/

    calculate(htmlSumme, htmlMwStBetrag, "+", htmlGesSumme);
    return true;
}

```

2.7. Ein- und Ausblenden

2.7.1. Gruppen dynamisch ein- und ausblenden

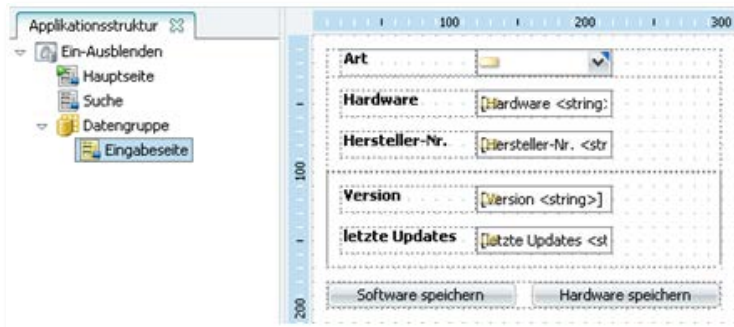
Gruppierungen können abhängig vom Wert eines Feldes ein- bzw. ausgeblendet werden.

Übung

Erstellen Sie die neue Applikation *Ein-Ausblenden* auf Basis der Vorlage *Leere Applikation*. Auf der Eingabeseite wird über eine Auswahlliste eine Eingabe vorgenommen. Abhängig davon, welcher Wert gewählt wird, sollen unterschiedliche Zusatzkontrollen angezeigt werden.

Definieren Sie auf der Eingabeseite eine Auswahlliste *Art* mit den benutzerdefinierten Einträgen *Hardware* und *Software*. Die angezeigten Werte entsprechen dabei den gespeicherten Werten. Legen Sie die beiden Eingabefelder *Hardware* und *Hersteller-Nr.* an, die später bei der Auswahl *Hardware* eingeblendet werden sollen. Für die Auswahl *Software* legen Sie die beiden Eingabefelder *Version* und *letzte Updates* an. Alle Eingabeelemente haben den Datentyp *Text*.

Gruppieren Sie nun die beiden Eingabefelder *Hardware* und *Hersteller-Nr.* inklusive der Feldtitel und geben Sie der Gruppierung im Eigenschaftendialog (Doppelklick auf die Gruppierung) den Titel *Hardware*. Gruppieren Sie dann die beiden Eingabefelder *Version* und *letzte Updates* inklusive der Feldtitel und geben Sie der Gruppierung im Eigenschaftendialog den Titel *Software*.



Nun wird für das onchange-Event der Auswahlliste ein neues Skript erstellt. In diesem Skript wird auf die Style-Eigenschaften der Gruppierungen (*visibility* und *display*) zurückgegriffen. Hierbei ist zu beachten, dass nur die gespeicherten Werte der Auswahlliste per JavaScript überprüft werden.

Übernehmen Sie das folgende Skript:

```
function showControls()
{
    var htmlArt      = getElement("13BA...6116"); /*Art dropdowncontrol*/
    var grHardware   = getElement("F7CA...3AC1"); /*Hardware simplegroup*/
    var grSoftware   = getElement("2B83...CDF4"); /*Software simplegroup*/

    if(Browser.getValue(htmlArt) == "Hardware")
    {
        /* Hardware einblenden */
        grHardware.style.visibility = "visible";
        grHardware.style.display = "block";

        /* Software ausblenden */
        grSoftware.style.visibility = "hidden";
        grSoftware.style.display = "none";
    }
    else
    {
        if(Browser.getValue(htmlArt) == "Software")
        {
            /* Software einblenden */
            grSoftware.style.visibility = "visible";
            grSoftware.style.display = "block";

            /* Hardware ausblenden */
            grHardware.style.visibility = "hidden";
            grHardware.style.display = "none";
        }
    }
    return true;
}
```

Tragen Sie das Skript auch im *onload*-Event der Eingabeseite ein. Speichern und testen Sie dann die Applikation im Browser.

2.7.2. Schaltflächen dynamisch ein- und ausblenden

Auch einzelne Schaltflächen können ein- und ausgeblendet werden. Im folgenden Beispiel wird die Schaltfläche *Software* nur dann aktiv, wenn die Art *Software* gewählt wurde, die *Hardware* nur dann, wenn die Art *Hardware* gewählt wurde.

Übung

Erstellen Sie zwei Schaltflächen *Software* und *Hardware*. Die Schaltflächen besitzen beide jeweils folgende Eigenschaften: Aktion: Speichern; Sprung auf die Hauptseite der aktuellen Applikation.

Bei der Wahl der *Art* muss nun also eine neue Überprüfung gestartet werden, nämlich, welche Schaltfläche auf der Seite aktiv sein soll. Wir werden eine neue Funktion für die Überprüfung erstellen und diese Funktion in der ersten (*showControls*) aufrufen. Dazu wird die gewählte Art als Parameter übergeben. Die Parameterübergabe erfolgt beim Aufruf der Funktion in der Funktionsklammer, beim Abruf der neuen Funktion wird in der Funktionsklammer der Parameter in einer Variablen übergeben. Die neue Funktion sieht nun so aus:

```
function showButton(strArt)
{
  var btHardware = getElement("1...D"); /*Hardware speichern  buttoncontrol*/
  var btSoftware = getElement("1...5"); /*Software speichern  buttoncontrol*/

  if(strArt == "Hardware")
  {
    btHardware.oUp.enable();
    btSoftware.oUp.disable();
  }
  else
  {
    btHardware.oUp.disable();
    btSoftware.oUp.enable();
  }
  return true;
}
```

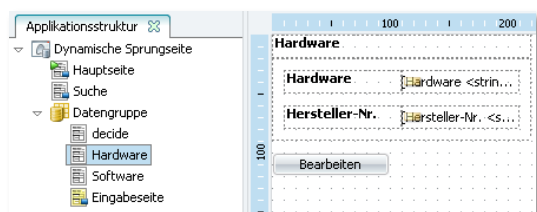
In der Funktion *showControls* fügen Sie bitte den Aufruf der Funktion *showButton* hinzu:

```
function showControls()
{
  (...)
  showButton(Browser.getValue(htmlArt));
  return true;
}
```

2.8. Mit Skript einen Klick auf eine Schaltfläche auslösen

Aus einer Ansichtstabelle heraus soll nun bei Auswahl der Art *Hardware* auf eine Seite gewechselt werden, auf der alle Hardware-Eigenschaften angezeigt werden. Bei Auswahl der Art *Software* soll auf eine Seite gesprungen werden, auf der alle Software-Eigenschaften angezeigt werden.

Übung

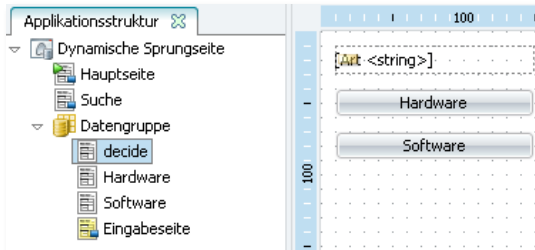


Öffnen Sie eine Kopie der letzten Applikation und speichern Sie sie unter dem Namen *Dynamische Sprungseite* ab. Erstellen Sie aus der *Eingabeseite* die Ansichtstabelle *Hardware*

und löschen Sie alle die Software betreffenden Felder. Die Schaltfläche *Bearbeiten* verweist auf die *Eingabeseite*. Geben Sie der Seite eine Überschrift mit dem Titel *Hardware*.

Erstellen Sie aus der *Eingabeseite* eine weitere Ansichtseite *Software* und löschen Sie alle die Hardware betreffenden Felder. Die Schaltfläche *Bearbeiten* verweist auf die *Eingabeseite*.

Geben Sie der Seite eine Überschrift mit dem Titel *Software*.



Geben Sie in der Ansichtstabelle auf der Hauptseite als neues Sprungziel die Seite *decide* an. Erreicht werden soll, dass bei Auswahl eines Datensatzes aus der Ansichtstabelle automatisch auf die der *Art* entsprechende Eingabeseite weitergeleitet wird. Hat *Art* den Wert *Software*, soll die Ansichtseite *Software* aufgerufen werden, ist der Wert *Hardware*, soll die Ansichtseite *Hardware* aufgerufen werden. Die Entscheidung, welche Zielseite in Abhängigkeit vom Wert des Feldes *Art* aufgerufen wird, wird von einem Skript getroffen.

Das Skript muss

- die *Art* auslesen (Hinweis: das Ansichtselement besitzt keine *value*-Eigenschaft)
- bei der *Art Software* die Schaltfläche *Software* auslösen (einen Klick simulieren)
- bei der *Art Hardware* auf die Schaltfläche *Hardware* "klicken".

Um dies zu erreichen, setzen wir die UP-Methode *processRequest* ein. Das folgende Skript wird im *onload*-Event der Ansichtseite *decide* aufgerufen:

```
function decide()
{
  var strArt    = getElement("8B3A...6AB3"); /*Art  textvcontrol*/
  var buttonHW = getElement("723A...D617"); /*Hardware  buttoncontrol*/
  var buttonSW = getElement("E25B...276C"); /*Software  buttoncontrol*/
  if(Browser.getValue(strArt) == "Hardware")
    buttonHW.onclick();
  else
    buttonSW.onclick();
  return true;
}
```

Verstecken Sie die Felder der Seite *decide*, indem Sie die Felder markieren und den Eintrag *Gruppieren ausgeblendet* aus dem Kontextmenü wählen.

Speichern und testen Sie die Applikation.

3. Aufruf von Java-Klassen

3.1. Informationen über den angemeldeten User

Das *\$User-Objekt* stellt alle Informationen zur Verfügung, die den aktuell angemeldeten User betreffen. Folgende Methoden stehen zur Verfügung:

Methodenname	Information zum aktuellen Benutzer
int getId()	ID
String getGuid()	GUID
String getEmployeeNo()	Personalnummer
String getFirstName()	Vorname
String getMiddleName()	2. Vorname
String getTitle()	Titel
String getFullName()	Vorname und Nachname
String getLastName()	Nachname
String getStreet()	Straße
String getCountry()	Land
String getPostalCode()	Postleitzahl
String getCity()	Ort
String getEmailBiz()	eMail geschäftlich
String getPhoneBiz()	Telefon geschäftlich
String getPhoneMobileBiz()	Telefon Mobil geschäftlich
String getPhoneFax()	Fax geschäftlich
String getEmailHome()	eMail privat
String getPhoneHome()	Telefon privat
String getPhoneMobileHome()	Telefon Mobil privat
String getDescription()	Beschreibung
Date getBirthday()	Geburtsdatum
Date getEnterDate()	Eintrittsdatum
int getGender()	Geschlecht
String getLoginName()	Benutzername (ohne Domäne)
String getQualifiedLoginName()	Benutzername mit Domäne
String getLoginDomain()	Domäne
TimeZone getTimeZone()	Zeitzone
int getBoss()	Vorgesetzter (ID)
String getPhonePager()	Pager
boolean isAnonymous()	Anmeldestatus
boolean isDisabled()	Konto gesperrt
int getContainerId()	ContainerId (z.B. System)
String getDefaultLanguage()	Standardsprache
String getDefaultLayout()	(nicht der Layoutname, sonder Art!)
String getExternalLogin1()	Externer Loginname 1
String getExternalLogin2()	Externer Loginname 2

String getExternalLogin3()	Externer Loginname 3
String getExternalPassword1()	Passwort für Externes Login 1
String getExternalPassword2()	Passwort für Externes Login 2
String getExternalPassword3()	Passwort für Externes Login 3
String getImageType()	Bildtyp
String getName()	Objektname
String getPoBox()	Postfach
int getProxyPerson()	Id des Stellvertreters?
String getState()	Bundesland
String getTimeZoneId()	Zeitzone
String getUserImageContentType()	Bildtyp
String getUserImageFileName()	Vollständiger Bildname
String getUserImageUrl()	Bildpfad
Boolean isDeletable()	löschar
Boolean isDeleted()	Benutzer gelöscht

Die Werte der Benutzer-Zusatzfelder (Attribute) können mit folgenden Methoden ausgelesen werden:

Stringwerte:

```
$.DefaultHtmlEncodingRenderer.writeOutput($Response.getWriter(),
    $!User.getCustomMapVH().get('Kostenstelle'))
```

Integerwerte:

```
$.DefaultIntegerRenderer.writeOutput($Response.getWriter(),
    $!User.getCustomMapVH().get('Durchwahl'))
```

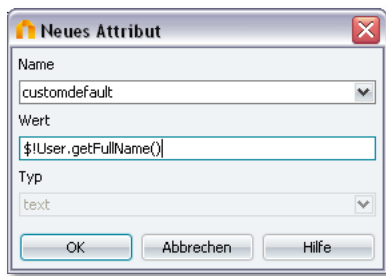
Nachstehend eine vollständige Liste der Renderer, die in Abhängigkeit zum Attribut-Typ verwendet werden müssen:

```
$.DefaultSimpleTextRenderer
$.DefaultDateTimeRenderer
$.DefaultDateRenderer
$.DefaultTimeRenderer
$.DefaultIntegerRenderer
$.DefaultCurrencyRenderer
$.DefaultNumberRenderer
$.DefaultBooleanRenderer
$.SimpleTextRenderer
$.DefaultBooleanAsIntRenderer
```


Übung

In einem Eingabeformular sollen einige den Benutzer betreffende Felder beim Laden der Seite mit den Daten aus der Benutzerverwaltung gefüllt sein. Erstellen Sie eine neue Applikation mit dem Namen *Benutzerdaten*.


Auf der Eingabeseite legen Sie die Eingabefelder *Voller Name*, *eMail* und *Telefon* als Eingabefelder an. Damit die Daten des aktuellen Benutzers bei Aufruf der Seite angezeigt werden, öffnen Sie den Eigenschaftendialog des Eingabefeldes *Name* und wechseln auf den Reiter *Expert*.



Hier wird das neue Expertattribut *customdefault* angelegt. Das Attribut definiert den Default-Wert für ein Element. Als Wert geben Sie *\$User.getFullName()* an. Beachten Sie bitte die Groß-Kleinschreibung (case-sensitive). Das *\$User*-Objekt stellt die Informationen über den User zur Verfügung, die die entsprechende Methode (aus obiger Liste) spezifiziert. Objekt und Methode werden immer durch einen Punkt getrennt.

-  Geben Sie nach dem *\$*-Zeichen ein *!*-Zeichen ein (Beispiel: *\$!User.getLastName()*). Damit bleibt das Eingabefeld leer, wenn kein Nachname hinterlegt ist.

Speichern Sie dann die Applikation ab und testen Sie sie, indem Sie die Eingabeseite aufrufen und kontrollieren, ob alle Daten des aktuellen Benutzers in den entsprechenden Eingabefeldern übernommen sind.

-  Die Defaultwerte werden nur eingetragen, wenn ein Datensatz neu angelegt wird, also nicht beim Sprung auf eine zweite Eingabeseite in der gleichen Datengruppe. Sie können jedoch auch als Parameter im JavaScript-Aufruf übergeben werden.

Natürlich lassen sich die Vorgabewerte auch aus der grafischen Vorgabe erzeugen. Erstellen Sie die drei Felder als statische Ansichtselemente mit dem Inhalt *\$User.getFullName()* etc. und achten Sie darauf, dass in den Optionen *Programmierung, nur Standardsprache* eingestellt ist. Die Darstellung in für z.B. JavaScripte interessant, die mit Benutzerwerten arbeiten. Weitere Einsatzgebiete: in Requestwerten und in der Velocityumgebung.

3.2. Request-Werte

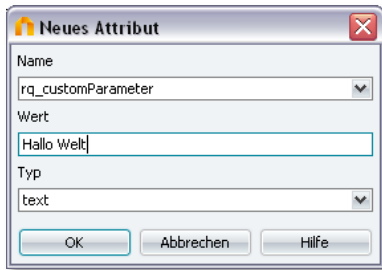
Das Requestobjekt ist ein JavaObjekt, das alle Daten, die zur Request-Verarbeitung benötigt werden, enthält. Bei einer Anfrage an den Server (z.B. bei Klick auf eine Schaltfläche) wird dieses Objekt in der Methode *processRequest()* erzeugt, mit den notwendigen Daten gefüllt, wie z.B. den Werten der Kontrollen, die gespeichert werden sollen, und an den Server geschickt. Die Methode *processRequest()* ist in der Datei *Object.js* enthalten. Über das so erzeugte Objekt können auch Werte von einer Seite zur nächsten geschickt werden, die dort wiederum über Velocity abgefragt und weiterverarbeitet werden können.

3.2.1. Request mit festem Parameter

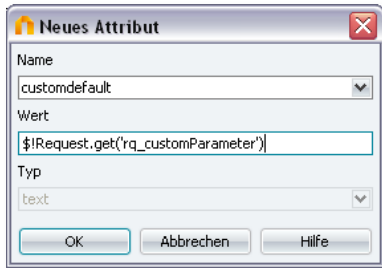
Wir befinden uns auf einer Seite und wollen mit Hilfe einer Schaltfläche einen festen Wert an eine andere Seite übergeben.

Übung

Erstellen Sie eine neue Applikation mit dem Namen *Requestwerte*. Erstellen Sie auf der Eingabeseite das Feld *Wert 1*. Auf der Hauptseite erstellen Sie eine Schaltfläche mit dem Titel *Neuer Datensatz mit Request-Wert* und Sprung auf die Eingabeseite. Öffnen Sie den Eigenschaftendialog der Schaltfläche und wechseln Sie auf den Reiter *Expert*. Legen Sie das neue Expertattribut *rq_customParameter*, Wert *Hallo Welt*, Typ *text* an.



Nun wechseln Sie auf die Eingabeseite und rufen den Eigenschaftendialog des Feldes *Wert1* auf. Legen Sie das neue Expertattribut *customdefault*, Wert *`\${Request.get('rq_parameter')}`*, Typ *text* auf dem Reiter *Expert* an:



Das Ausrufezeichen vor dem Request bedeutet, dass kein Inhalt angezeigt wird, wenn kein oder ein leerer Request-Wert übergeben wurde. Speichern und testen Sie die Applikation.

3.2.2. Request mit variablem Parameter

Ein Parameter kann auch variabel übergeben werden. Diese Funktionalität benötigen Sie, wenn Sie z.B. einen Feldinhalt auslesen möchten. Beachten Sie, dass Sie den folgenden Request nur dann verwenden können, wenn der Datensatz bereits angelegt ist.

Beim Seitenaufbau werden angezeigte Werte aus dem *DataCollectionObjekt* (abgekürzt *DC*) ausgelesen. Sie sind unter dem Namen der anzeigenden Kontrolle hinterlegt. Dabei definieren wir den Request mit Präfix *rq_custom* und hängen einen beliebigen Namen an. Den Wert lesen wir mit Hilfe der Methode *DC.getValueHolder()* aus:

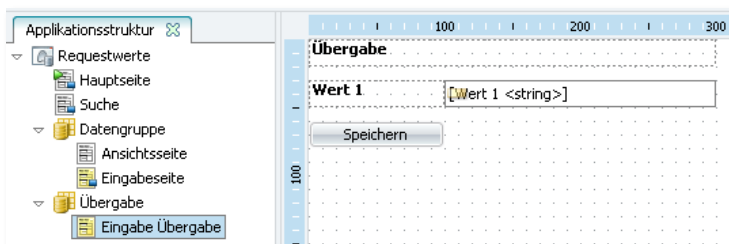
```
`${SimpleTextRenderer.getOutput(`${DC.getValueHolder('Name des Elements')}`)}`
```

Ausgegeben wird der Wert in einem Eingabefeld mit dem Expertattribut *customdefault* und dem Wert *`\${Request.get('rq_customName')}`*.

Übung

Um Feldinhalte in eine zweite Datengruppe zu übergeben und dort auslesen zu können, erstellen Sie in der Applikation *Request-Werte* eine neue Datengruppe *Übergabe* mit einer Eingabeseite *Eingabe Übergabe*.

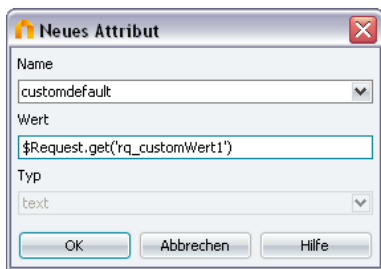
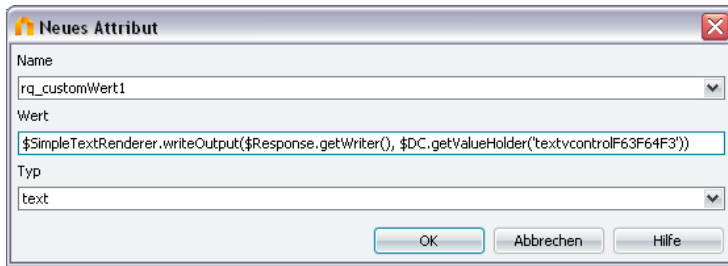
Fügen Sie am Kopf der Seite ein Ansichtselement *statischer Text* und dem Titel *Übergabe* ein und wechseln Sie auf den Reiter *Ansicht*. Hier kann der Stil *Überschrift 1* zugeordnet werden. Legen Sie das Eingabefeld *Wert 1* an.



Erstellen Sie dann eine Schaltfläche *Speichern* mit der Aktion *Speichern*. Dann definieren Sie in der ersten Datengruppe eine Ansichtseite mit einem Ansichtsfeld *Wert 1*, das mit

dem Datenfeld *Wert 1* verbunden wird. Auf der Hauptseite definieren Sie eine Ansichtstabelle mit den Daten der Datengruppe und Sprung auf die Ansichtseite. Erstellen Sie auf der Ansichtseite eine Schaltfläche *Werte in zweite Datengruppe übergeben* und Sprung auf die Eingabeseite der Datengruppe *Übergabe*. Öffnen Sie den Eigenschaftendialog der Schaltfläche und wechseln Sie auf den Reiter *Expert*. Erstellen Sie das neue Expertattribut *rq_customWert1*. Den Wert des Attributes liefert die Methode

```
$SimpleTextRenderer.writeOutput($Response.getWriter(),
$DC.getValueHolder('Name des Ansichtsfeldes Wert 1'))
```



Wechseln Sie auf die Eingabeseite der Datengruppe *Übergabe* und legen Sie im Expertmodus des Eingabefeldes *Wert 1* das neue Expertattribut *customdefault*, Wert *\$Request.get('rq_customWert1')* an. Mit diesem Attribut wird der Requestwert ausgelesen.

Speichern und testen Sie die Applikation im Browser.

3.2.3. Cookies setzen und auslesen

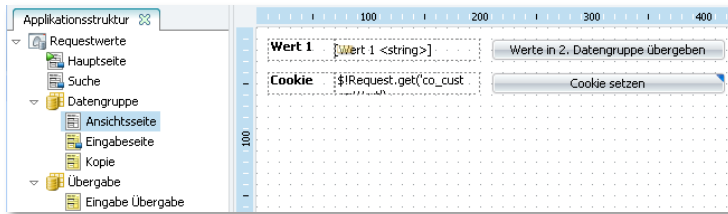
Das folgende Beispiel zeigt das Speichern von Requestwerten in Cookies. Dabei wird zuerst geprüft, ob das Setzen von Cookies möglich ist.

Übung

Erstellen Sie auf der Ansichtseite der Applikation *Requestwerte* eine Schaltfläche *Cookie setzen*. Diese Schaltfläche führt keine Aktion aus, springt aber auf die Ansichtseite (die aktuelle Seite) und enthält ein Skript, mit dem geprüft wird, ob Cookies gesetzt werden können und dann das Cookie setzt. Dazu wird die Funktion *Helper* verwendet.

```
function setCookie()
{
    if (Helper.isCookieEnabled())
    {
        var l_Wert = getElement("8...0"); /*Wert 1 textvcontrol*/
        Helper.setCookie("co_customWert", Browser.getValue(l_Wert), false);
        return true;
    }
    else
    {
        alert("Bitte aktivieren Sie Ihre Cookies!");
        return false;
    }
}
```

Das Skript soll im *onclick*-Event der Schaltfläche ausgeführt werden. Unter der Schaltfläche erstellen Sie nun ein Ansichtselement *statischer Text* mit dem Befehl *!\$Request.get('co_customWert')*.



3.3. Datensatz kopieren

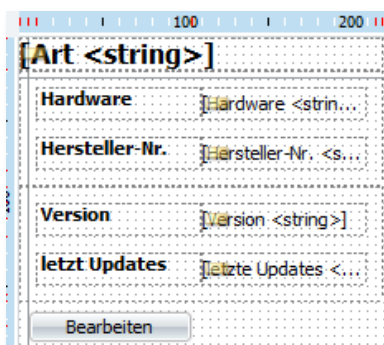
Wenn sie die Daten des aktuellen Datensatzes kopieren möchten, können Sie nicht mit selbst definierten Requestwerten arbeiten, da diese Werte beim Aufruf der Seite noch nicht existieren. Wir lesen die Eingabefelder direkt über *customdefault* aus, da diese sich noch im Systemrequest befinden.

- Die eingegebenen Werte einer Eingabeseite, die mit einer Schaltfläche gespeichert werden, werden im Request unter dem Namen der Eingabekontrolle hinterlegt. Sie können auf einer folgenden Seite unter diesem Namen abgefragt werden.

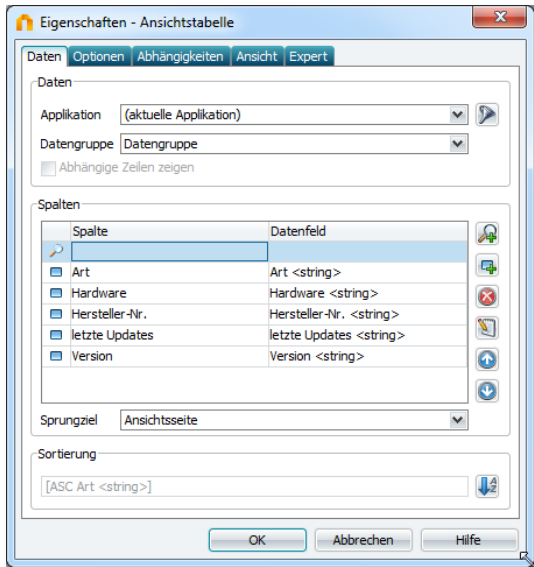
Kopieren Sie in der Applikation *Requestwerte* die Eingabeseite der Datengruppe und benennen Sie diese Seite um in *Kopie*. Auf der Eingabeseite ermitteln Sie mit der Taste *F4* den Namen des Eingabefeldes *Wert 1* und kopieren diesen. Wechseln Sie nun auf die Eingabeseite *Kopie* und ändern Sie den Wert des Attributes *customdefault* auf *\$Request.get('Name des Eingabefeldes aus der Eingabeseite')*. Wechseln Sie auf die Eingabeseite und erstellen Sie die neue Schaltfläche *Kopie*, die den aktuellen Datensatz *speichert*, auf die Seite *Kopie* wechselt und dabei einen neuen Datensatz anlegt. Wechseln Sie dann auf die Ansichtseite und erstellen Sie die neue Schaltfläche *Wert Ändern* mit Sprung auf die Eingabeseite. Speichern Sie Ihre Applikation und testen sie.

3.4. Bedingte Anzeige

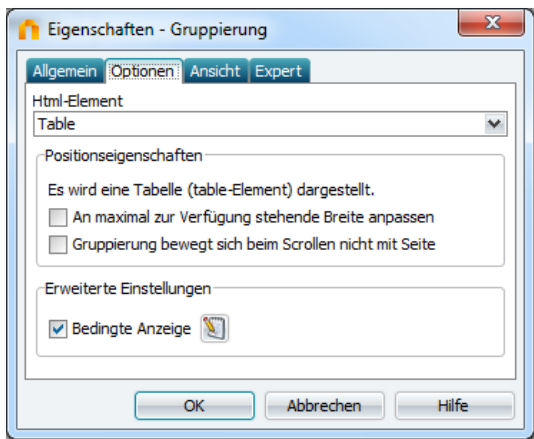
Im Anwendungsbereich der Applikation kann die Anzeige aller Elemente über die *bedingte Anzeige* beeinflusst werden. Im Folgenden wird eine Alternative zur Übung *dynamische Filter* mit Hilfe der *bedingten Anzeige* aufgebaut. Kopieren Sie dazu über den Applikationsmanager die Applikation *dynamische Sprungseite*. Speichern Sie die neu kopierte Applikation unter dem Namen *Bedingte Anzeige* ab. Nehmen Sie folgende Änderungen vor:




Erstellen Sie eine neue Ansichtseite in der Datengruppe. Lassen Sie alle relevanten Daten anzeigen. Die Schaltfläche *Bearbeiten* verweist auf die Eingabeseite.




Ändern Sie auf der Hauptseite das Sprungziel der Tabelle. Die Zielseite muss die eben angelegte *Ansichtssseite* sein.



Wechseln Sie zurück auf die *Ansichtssseite* und wechseln in die Eigenschaften der Gruppierung der *Hardware*. Aktivieren Sie die *Bedingte Anzeige* und klicken Sie auf das  Symbol.

Fügen Sie folgenden Code in das Textfeld:

```
#set($Art = $!DC.getValueHolder('NameAnsichtsfeldArt').getValue())
#ife($Art == "Hardware")
  #set($show_'NameGruppierung' = true)
#else
  #set($show_'NameGruppierung' = false)
#end
```

Bestätigen Sie mit *OK* und wechseln in die Eigenschaften der Gruppierung der *Software*. Aktivieren Sie die *Bedingte Anzeige* und klicken auf das  Symbol. Fügen Sie folgenden Text in das Textfeld:

```
#set($Art = $!DC.getValueHolder('NameAnsichtsfeldArt').getValue())
#ife($Art == "Hardware")
  #set($show_'NameGruppierung' = true)
#else
  #set($show_'NameGruppierung' = false)
#end
```

Speichern Sie die Applikation und testen Sie sie.

4. Optionen für Experten

4.1. JSObject

Das Expertattribut *jobject* ermöglicht den Zugriff auf Werte von Spalten und Reihen in Tabellen und die direkte Abfrage von Werten in Ansichtsfeldern.

Wird das Attribut bei einer Ansichtstabelle oder frei gestalteten Tabelle eingesetzt, so wird automatisch ein UpTable-Objekt erzeugt. Das UpTable-Objekt kann über das globale Objekt *oTableReg* erreicht werden. Neben Properties wie ID und GUID der Tabelle enthält es eine Liste der RecIds der auf der Browserseite angezeigten Datensätze und eine Liste der Tabellenspalten.

Ein Spaltenobjekt enthält Properties wie *ID*, *GUID*, *displayName* und *controlType* und bietet Methoden für den Zugriff auf Nodes (<a> oder Tags) oder direkt auf Inhalte der Textnodes.

Beispiel für den Zugriff auf das Objekt *oTableReg*:


```
oTableReg.getTableByGuid("GUID der Tabelle").getColByGuid("GUID der Spalte").getNodeValue(recid)
```

Bei der freien Tabelle wird hier die RecordID des Datensatzes übergeben, bei Ansichtstabellen der VelocityCounter.

Diese Methode liefert den Wert aus der Spalte für die übergebene Spalte:

```
oTableReg.getTableByGuid("GUID der Tabelle").aRecIds
```

Hier wird ein Array der RecordIDs der Seite gebildet (new Array(1,2,5)). Weitere Methoden sind in der Datei *object.js* enthalten.

-  Bei der *Frei gestalteten Tabelle* muss die GUID über den Eigenschaftendialog (Reiter *Expert*) ermittelt werden. Bitte setzen Sie nicht die GUID ein, die sich mit der Taste *F4* ermitteln lässt (Details).

Bei Ansichtselementen baut das Attribut eine bidirektionale Beziehung zwischen dem HTML-Objekt des Elements und dem zugehörigen UP-Objekt für die Verwendung in JavaScript auf. Bei Eingabekontrollen ist die bidirektionale Beziehung Standard. So können z.B. mit dem Wert eines Datums-Ansichtsfeldes Berechnungen durchgeführt werden, wobei die kontrollspezifischen Formatierungen berücksichtigt werden.

Der Wert der Ansichtskontrollen lässt sich direkt über die Funktion

```
getElement("GUID").oUp.displayValue
```

abfragen.

Beim Ansichtselement *Kontrollkästchen Ansicht* kann der Wert über die Funktion

```
getElement("GUID").oUp.checked
```

abgefragt werden. Der Rückgabewert ist *true* oder *false*.

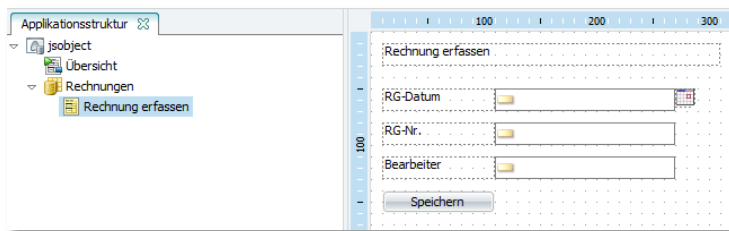
- Bitte beachten Sie, dass die Namen und GUIDs beim Import einer Applikation geändert werden. Wird die Applikation nicht mit der Option *Bestehende Applikationen überschreiben* importiert, so müssen die verwendeten Elementnamen und GUIDs anschließend ersetzt werden.

Tabellenberechnung mit jsubject

Im nachfolgenden Beispiel werden Rechnungspositionen in einer Gesamttabelle berechnet, eine Gesamtsumme ausgegeben und eine Zusatzspalte mit kumulierten Rechnungsbeträgen gefüllt. Als Beispiel soll eine Rechnungsverwaltung dienen, welche in einer Datengruppe die Rechnungsdaten speichert und in einer untergeordneten die dazugehörigen Positionen.

Erstellen Sie dafür eine neue Applikation vom Typ *Leere Applikation* und benennen Sie sie um in *jsubject*.

Benennen Sie die Datengruppe um in *Rechnungen* und erstellen Sie auf der Eingabeseite (umbenennen in *Rechnung erfassen*) die Felder *RG-Datum* (Typ *Datum*), *RG-Nr.* (Typ *kurzer Text*), *Bearbeiter* (Typ *kurzer Text*). Außerdem wird eine Schaltfläche *Speichern* mit der Aktion *Speichern* mit Sprung auf die Übersicht benötigt.



Generieren Sie aus der Seite *Rechnung erfassen* eine Ansichtssseite mit allen Elementen namens *Ansicht Rechnung*.

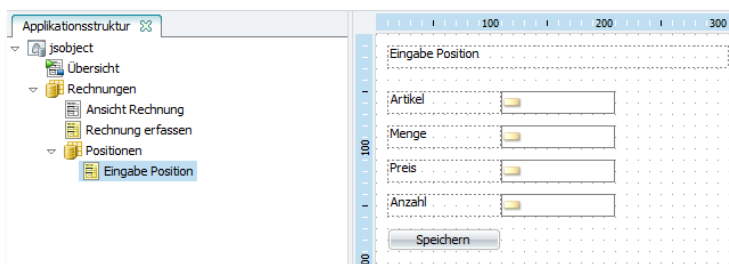
Erstellen Sie auf der Hauptseite eine Ansichtstabelle mit den Feldern der Datengruppe *Rechnung* und Sprung auf die Seite *Ansicht Rechnung*.

Fügen Sie unterhalb der *Rechnungen* die untergeordnete Datengruppe *Positionen* und eine Eingabeseite mit dem Namen *Eingabe Position* ein.

Darauf erstellen Sie die Felder

- *Artikel* (Typ *Text*)
- *Menge* (Typ *Gleitkommazahl*)
- *Preis* (Typ *Währung*)
- *Anzahl* (Typ *Ganzzahl*) (Vorgabe: 1)
-

Die Schaltfläche *Speichern* mit der Aktion *Speichern* lädt die Seite *Ansicht Rechnung*.



Auf der Seite *Ansicht Rechnung* benötigen wir noch eine Schaltfläche *Neue Position* mit Sprung auf die Eingabeseite der Positionen sowie eine Ansichtstabelle mit den Daten der Datengruppe *Positionen*.

Speichern und testen Sie die Applikation.

Erstellen Sie nun für die Summierung der Preise das neue Datenfeld *Dummy* in der Datengruppe *Positionen* (Typ *Gleitkommazahl*). Wechseln Sie auf *Ansicht Rechnung* und nehmen dieses neue Feld in die Tabelle auf, wobei Sie es in *Summe* umbenennen. Wechseln Sie in den Expertmodus der Tabelle und fügen Sie folgendes Attribut neu hinzu:

Name: *jobject*, Wert: *true*, Typ: *Boolean*

Legen Sie das Ansichtselement *Statischer Text* mit der Bezeichnung *Summe*: unterhalb der Tabelle an. Dahinter folgt ein weiteres Ansichtselement *Statischer Text* mit der Bezeichnung *_SummeGes*. Es soll später über ein JavaScript die Summe aller Rechnungen anzeigen.

Kopieren Sie folgendes JavaScript in den JavaScript-Editor:

```
function calcTable()
{
  /* GUID der Tabelle, aus der die Summe ermittelt werden soll. Bei */
  /* Vorselektion die GUID der Tabelle verwenden, in der die Datensätze */
  /* angezeigt werden.*/

  var oTable = oTableReg.getTableByGuid("45C2...150F");
  /* GUID des Feldes in der Ansichtstabelle, das summiert werden soll. */
  var oColumnSumme = oTable.getColByGuid("575C...00CA"); /*Column Preis*/
  var oColumnAnzahl = oTable.getColByGuid("181E...D6ED"); /*Column Anzahl*/
  var oColumnDummy = oTable.getColByGuid("4191...7633"); /*Column Dummy*/
  var oRecids = oTable.aRecIds;
  var summeRG = 0;
  var summe = 0;
  for (var i=1; i <= oTable.aRecIds.length; i++)
  {
    var oValueRG = oColumnSumme.getNodeValue(i);
    var oValueAz = oColumnAnzahl.getNodeValue(i);
    var valueRG = Helper.parseFloat(oValueRG);
    var valueAz = Helper.parseInt(oValueAz);

    summe = valueRG * valueAz;
    var oFloatdummy = new upFloatControl();
    var ausgabedummy = oFloatdummy.toLocalFormat(summe);
    oColumnDummy.getNode(i).innerHTML = summe;

    summeRG = summeRG + summe;
  }
  /* GUID des Ansichtsfeldes, das das Ergebnis anzeigt*/
  var oHTMLSummeGes = getElement("165D...FCF6"); /*_SummeGes labelcontrol*/

  var oFloat = new upFloatControl();
  var ausgabeSumme = oFloat.toLocalFormat(summeRG);

  Browser.setValue(oHTMLSummeGes, ausgabeSumme);
  return true;}


```

Passen Sie die GUIDs im JavaScript entsprechend an. Die Funktion soll im *onload*-Event der Seite *Ansicht Rechnung* aufgerufen werden. Speichern Sie anschließend die Applikation und testen Sie das Ergebnis.

5. VM-Include

5.1. Was ist Velocity?

Velocity ist ein Open-Source-Projekt der Jakarta-Projektgruppe von Apache. Velocity ist eine javabasierte Template Engine. Sie ermöglicht den direkten Aufruf von Javaklassen aus Web-Seiten heraus. *VTL* ist die Abkürzung für *Velocity Template Language*.

-  VTL-Referenzen beginnen mit dem Zeichen \$ und werden für Abfragen verwendet (z.B. Variablendefinitionen oder Klassenaufrufe, die Daten liefern). VTL-Anweisungen beginnen mit dem Zeichen # und werden für die Ausführung verwendet.

Wo wird Velocity angewendet?

Velocity-Aufrufe können ausschließlich auf VM-Seiten verarbeitet werden. Velocity-Befehle können direkt in eine statische VM eingetragen oder über das XSL in dynamische Seiten eingebunden werden.

Der Applikationsdesigner bietet die Möglichkeit der Verarbeitung von Velocity-Befehlen im Expertmodus oder in der VTL-Kontrolle. Darüber hinaus können über die Eigenschaftendialoge der Elemente im Applikationsdesigner Informationen aus Javaklassen abgefragt werden (z.B. Informationen über den aktuellen Benutzer oder Parameter aus dem Request).

5.2. Verwenden der JavaKlasse zum Ausführen der Queries

Zum Ausführen direkter Abfragen wird eine Javaklasse benötigt, welche die Abfrage ausführt und die Daten zurückliefert. Die Javaklasse wird folgendermaßen strukturiert:

1. Der ConnectionString an die Datenbank übergeben. Dieser ConnectionString muss in der Datei *Server.cfg* definiert sein, damit die Klasse sich mit der Datenbank verbinden kann.
2. Die Query wird in der VM definiert und an die Klasse übergeben.
3. Die Javaklasse führt die Abfrage aus und liefert das ResultSet direkt an die VM.
4. Die Javaklasse stellt weitere Hilfsklassen, z.B. Formate für Datumswerte zur Verfügung.

5.3. Prepared Query

Die Klasse *PreparedQuery* ist der "Nachfolger" der Klasse *CustomQuery* und bietet neben der Sicherheit gegen einen Angriff durch SQL-Injection die Möglichkeit der Wiederverwendung von Abfragen und vor allem auch die Möglichkeit, geschachtelte Abfragen durchzuführen. Die Rückgabewerte dieser Java-Klasse sind so genannte *ValueHolder*, die mit Hilfe von Renderern optimal für die Ausgabe aufbereitet und formatiert werden können.

PreparedQuery hat im Wesentlichen nur eine einzige Methode, die Ihnen die Möglichkeit des Zugriffes auf alle benötigten Klassen für eine Datenbankabfrage bietet. Im Folgenden erhalten Sie eine Übersicht über die Klassen und Methoden.

5.4. Methoden der Klasse PreparedQuery

DbPreparedStatement prepare(final JdbcConnection p_conn, final String p_strQuery)

Liefert das PreparedStatement zurück, das über die gegebene Datenbankverbindung und die Abfrage angefordert wurde.

5.5. Methoden der Klasse DbPreparedStatement

DbResultSet executeQuery()

Wird immer dann zum Ausführen eines Statements verwendet, wenn die Ergebnismenge der SQL-Abfrage ein Resultset ist.

Beispiel:

```
DbPreparedStatement.executeQuery("SELECT dtedit, strtext, strheadline FROM mytable")
```

int executeUpdate()

Wird immer dann für die Ausführung eines Statements verwendet, wenn Datensätze manipuliert werden. In diesem Fall wird als Ergebnis ein Integer-Wert mit der Anzahl der manipulierten Daten zurückgeliefert.

Beispiel:

```
DbPreparedStatement.executeUpdate("DELETE FROM mytable WHERE strstatus = 'delete' ")
```

void execute()

Wird immer dann zum Ausführen eines Statements verwendet, wenn keine Ergebnismenge zurückgeliefert wird.

Beispiel:

```
DbPreparedStatement.Execute("CREATE VIEW...", execute("DROP COLUMN..."))
```

setInt(final int p_idx, final IValueHolder<?> p_value)

Setzt einen Integer-ValueHolder an den gewünschten Parameter-Index. Bitte beachten Sie, dass der Index p_idx mit dem Wert 1 beginnt.

void setLong(final int p_idx, final IValueHolder<?> p_value)

Setzt einen Long-ValueHolder an den gewünschten Parameter-Index.

void setShort(final int p_idx, final IValueHolder<?> p_value)

Setzt einen Short-ValueHolder an den gewünschten Parameter-Index.

void setFloat(final int p_idx, final IValueHolder<?> p_value)


Setzt einen Float-ValueHolder an den gewünschten Parameter-Index.

void setDouble(final int p_idx, final IValueHolder<?> p_value)

Setzt einen Double-ValueHolder an den gewünschten Parameter-Index.

void setTimestamp(final int p_idx, final IValueHolder<?> p_value)

Setzt einen DateTime-ValueHolder an den gewünschten Parameter-Index.

 Verwenden Sie in Ihren Abfragen diese Methode, um einen Datums-/Zeitwert im Statement zu setzen. In diesem Fall wird automatisch die Zeitzone der eingesetzten Datenquelle berücksichtigt.

void setTimestamp(final int p_idx, final IValueHolder<?> p_value, final Calendar p_cal)

Setzt einen DateTime-ValueHolder an den gewünschten Parameter-Index.

void setTime(final int p_idx, final IValueHolder<?> p_value)

Setzt einen DateTime -ValueHolder an den gewünschten Parameter-Index und verwendet von diesem den Zeitanteil für den Parameter.

void setTime(final int p_idx, final IValueHolder<?> p_value, final Calendar p_cal)

Setzt einen DateTime -ValueHolder an den gewünschten Parameter-Index und verwendet von diesem den Zeitanteil für den Parameter.

void setDate(final int p_idx, final IValueHolder<?> p_value)

Setzt einen DateTime -ValueHolder an den gewünschten Parameter-Index und verwendet von diesem den Datumsanteil für den Parameter.

void setDate(final int p_idx, final IValueHolder<?> p_value, final Calendar p_cal)

Setzt einen DateTime -ValueHolder an den gewünschten Parameter-Index und verwendet von diesem den Datumsanteil für den Parameter.

void setString(final int p_idx, final IValueHolder<?> p_value)

Setzt einen String-ValueHolder an den gewünschten Parameter-Index.

void setCharacterStream(final int p_idx, final IValueHolder<?> p_value)

Setzt einen String-ValueHolder an den gewünschten Parameter-Index, um diesen einem CLOB als Wert zu setzen.

void close()

Schließt die Abfrage.

5.6. Methoden der Klasse DbResultSet

public boolean next()

Positioniert den Cursor auf die nächste Zeile des Resultset. Gibt es keine nächste Zeile, wird *false* zurückgegeben, andernfalls *true*. Initial ist der Cursor vor dem ersten Datensatz positioniert und erst der Aufruf von *next()* positioniert den Cursor auf dem ersten Datensatz.

public IValueHolder<?> getValueHolder(int p_idx)

Liefert den entsprechenden ValueHolder der Spalte an dem übergebenen Index. Bitte beachten Sie, dass der Index *p_idx* mit dem Wert 1 beginnt.

public Iterator<IDbRow> iterator()

Liefert einen Iterator, mit dem innerhalb des Velocity-Kontexts die Ergebnismenge an Datensätzen durchlaufen werden kann.

public void close()


Schließt das Resultset.

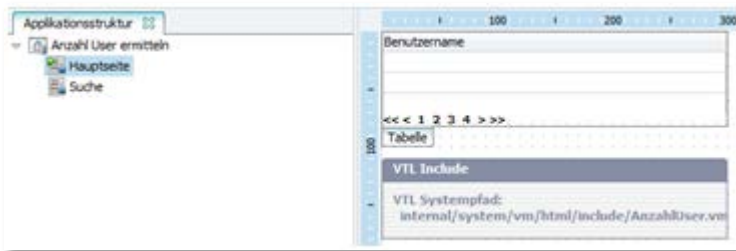
5.7. Anzahl der Benutzer ermitteln

In einer Tabelle sollen alle Systembenutzer angezeigt und die Gesamtanzahl der Benutzer ermittelt werden. Das Ergebnis soll unterhalb der Tabelle angezeigt werden.

Übung

Erstellen Sie eine neue Applikation *Anzahl User ermitteln*. Löschen Sie die enthaltene Datengruppe. Erstellen Sie nun auf der Hauptseite eine Ansichtstabelle mit allen Systemusern (verbunden mit der Datengruppe *Benutzer* aus der Applikation *Benutzer*)

und fügen Sie das Datenfeld *Benutzername* hinzu. Unterhalb der Ansichtstabelle erstellen Sie das Ansichtselement *VTL Include* und referenzieren eine neue *vm* im Applikationspaket, damit die *vm* bei einem Export im Paket enthalten ist. Die *vm* ist unter diesem Pfad zu finden:  *internal/application/resource/<APPGUID>/AnzahlUser.vm*.



Erstellen Sie folgendes Skript:

```
#set($l_statement = $PreparedQuery.prepare($DbConnection,"SELECT
COUNT(DSUSER.STRLOGIN) AS Anzahl FROM DSOBJECT INNER JOIN DSUSER ON
DSOBJECT.LID = DSUSER.LID WHERE (DSOBJECT.BDELETED = 0)")
#set($l_resultset = $l_statement.executeQuery())

Anzahl der Benutzer:
#if ($l_resultset.next())
    $l_resultset.getValueHolder(1).getValue()
#end

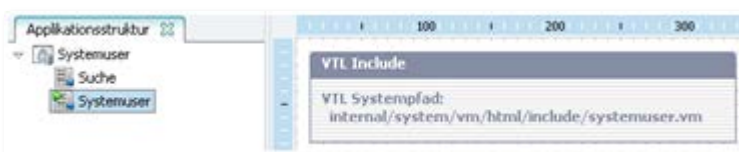
$l_resultset.close()
$l_statement.close()
```

l_resultset.next() positioniert dabei den Cursor auf die nächste Zeile des Resultsets. Gibt es keine nächste Zeile, wird *false* zurückgegeben, andernfalls *true*. Initial ist der Cursor vor dem ersten Datensatz positioniert. Erst der Aufruf von *next()* positioniert den Cursor auf dem ersten Datensatz.

Ist der Rückgabewert *true*, liefert *l_resultset.getValueHolder(1).getValue()* den Wert des entsprechenden ValueHolders der Spalte am übergebenen Index. Bitte beachten Sie, dass der Index mit dem Wert 1 beginnt.

5.8. Tabelle mit Usern anzeigen

Erstellen Sie die neue Datei *systemuser.vm* kopieren Sie den untenstehenden Code. Erstellen Sie eine neue Applikation mit dem Namen *Systemuser*. Löschen Sie die Datengruppe und erstellen Sie auf der Hauptseite (Name: *Systemuser*) ein VTL-Include. Geben Sie den entsprechenden Pfad zur Datei *systemuser.vm* an. Speichern Sie die Applikation ab und testen Sie das Ergebnis.



```

#####
## Makro definieren
#####

#macro(output_data)

<td nowrap class="SCUP_Datarange_RowText_BG">
  <span class="SCUP_Datarange_RowText_normal">
    $!l_row.getValueHolder($l_counter).getValue()
  </span>
</td>
#set($l_counter = $l_counter+1)
#if($l_counter <= $l_arglength)
  #output_data()
#end
#end

#####
##define your query and execute it
#####

#set($l_stmt = $PreparedQuery.prepare($DbConnection, "SELECT LID as
id,STRLOGIN AS Login,STRFULLNAME as Name,STRMAILBIZ as Mail FROM DSUSER"))
#set($l_rs = $l_stmt.executeQuery())

#####
## write the header to the html output
#####
<table cellpadding="0" cellspacing="0"
class="SCUP_Datarange_Container_normal_BG" style="empty-cells: show;">
  <tr>
    <td nowrap class="SCUP_Datarange_Header_BG" align="right"><span
class="SCUP_Datarange_Header">ID</span></td>
    <td nowrap class="SCUP_Datarange_Header_BG" align="right"><span
class="SCUP_Datarange_Header">Login</span></td>
    <td nowrap class="SCUP_Datarange_Header_BG" align="right"><span
class="SCUP_Datarange_Header">Name</span></td>
    <td nowrap class="SCUP_Datarange_Header_BG" align="right"><span
class="SCUP_Datarange_Header">Mail</span></td>
  </tr>

#####
## get the content
#####
  #foreach($l_row in $l_rs.iterator())
    #set($l_arglength = $l_row.size())
    #set($l_counter = 1 )
    #if($l_even)
      #set($l_even = false)
      #set($l_style="SCUP_Datarange_RowOdd_normal_BG" )
    #else
      #set($l_even = true)
      #set($l_style="SCUP_Datarange_RowEven_normal_BG" )
    #end
    <tr class="$l_style">
      #output_data()
    </tr>
  #end
</table>

#####
##close the recordset
#####
$l_rs.close()
$l_stmt.close()

```

In der Tabelle auf der Hauptseite werden im Browser alle (auch die gelöschten User) angezeigt. Legen Sie testweise einen Benutzer im Modul *Benutzerverwaltung* an und löschen Sie diesen Benutzer anschließend wieder. Sollen nur nicht gelöschte Benutzer angezeigt werden, so muss die in der *systemuser.vm* enthaltene SQL-Abfrage wie folgt modifiziert werden:

```
SELECT STREMPLOYEENO AS No, STRFULLNAME as Name, STRPHONEBIZ as Tel,
STRMAILBIZ as Mail FROM DSOBJECT INNER JOIN DSUSER ON DSOBJECT.LID =
DSUSER.LID WHERE (DSOBJECT.BDELETED = '0').
```

Erstellen Sie eine Kopie von *systemuser.vm* Ersetzen Sie die enthaltene SQL-Abfrage mit der oben angegebenen Abfrage. Speichern Sie die Datei unter *systemuser2.vm*, erstellen Sie ein weiteres VTL-Include und testen Sie das Ergebnis im Browser.

6. Ajax

6.1. Grundlagen

Ajax steht für **A**synchronous **J**avascript **A**nd **X**ML.

Es bezeichnet ein Konzept der asynchronen Datenübertragung zwischen einem Browser und dem Server. Dieses ermöglicht es, HTTP-Anfragen durchzuführen während eine HTML-Seite angezeigt wird und die Seite zu verändern ohne sie komplett neu zu laden (Quelle: [http://www.wikipedia.org/wiki/Ajax_\(Programmierung\)](http://www.wikipedia.org/wiki/Ajax_(Programmierung))).

Ajax basiert auf dem Prinzip vom Browser via JavaScript eine Anfrage an den Server zu senden, der eine entsprechende Aktion ausführt und via XML oder JSON Ergebnisse an den Browser zurückmeldet. Häufig wird dabei serverseitig PHP verwendet. Intrexx verwendet serverseitig die Velocity-Template-Language, kurz VTL.

Das so genannte XMLHttpRequest-Objekt, mit dem die Anfragen an den Server gesendet werden ist bereits einige Jahre verfügbar. Leider ist das Objekt nicht Bestandteil des W3C-Standards und daher nicht in allen Browsern einheitlich implementiert. Aus diesem Grund bietet Intrexx ein eigenes Objekt (**upXMLHttpRequest**) an, das die browserspezifische Behandlung intern löst und eine einheitliche Schnittstelle bildet.

Seit dem Ajax stärker thematisiert wird, entstand eine Vielzahl von so genannten Ajax-Frameworks. Diese bieten, je nach Typ verschiedene vorgefertigte Grundfunktionalitäten, um z.B. aus JavaScript übergebene Arrays zu verarbeiten oder Listen mit Drag&Drop-Funktionalität zu realisieren. Intrexx verwendet keinen dieser Frameworks sondern greift serverseitig auf die eigene JAVA-Business-Logik in Verbindung mit der Velocity-Template-Engine zurück. Dieses Prinzip ist offen für den Einsatz von bestehenden oder selbst entwickelten JAVA-Klassen. Browserseitig wird JavaScript sowie die von Intrexx verwendeten Objekte, Eigenschaften und Methoden verwendet.

6.2. Warum Ajax ?

Ajax wird primär dazu verwendet, um mehr Performance im Portal zu erzielen sowie die Bedienung für den Benutzer intuitiver zu gestalten. Performance bedeutet, das überflüssige Neuladen ganzer Seiten zu verhindern und gezielt im Hintergrund Daten abzurufen und direkt an betroffener Stelle zu ändern (DHTML). Die Realisierung von „Rich Applications“, man meint hierbei web basierende Anwendungen, welche sich nun wie eine kompilierte, lokal installierte Software verhält (z.B. Office-Software, Groupware, ...).

6.3. Ajax in Intrexx mit JSON Response

JSON bedeutet JavaScriptObjectNotation und ist ein gut lesbares Datenformat für den Datenaustausch. Im Vergleich zu XML lassen sich die gleiche Menge Daten kompakter darstellen und direkt in JavaScript-Objekte umwandeln.

6.3.1. Beispiel 1 – Hallo Welt



Erstellen Sie eine neue Applikation auf Basis der Vorlage *leere Applikation* mit dem Titel *Hallo Welt mit Ajax*. Beim Klicken auf eine Schaltfläche soll ein Request an den Server abgesetzt werden. Die auf dem Server ausgeführte VM-Datei liefert den Text "Halo Welt!" zurück.

Es soll dann ein Dialogfenster erscheinen, in dem der Inhalt *Hallo Welt!* ausgegeben wird.

a) Request an den Server via upSimpleAjax

Platzieren Sie auf der *Übersicht* eine Schaltfläche mit dem Titel *Hallo?*. Die folgende Funktion muss im JavaScript-Editor von Intrexx hinterlegt werden und anschließend dem onClick()-Ereignis der Schaltfläche zugewiesen werden.

Durch das Verwenden von *upSimpleAjax()* wird das Erzeugen der gesamten Response nicht benötigt.

```
function call_HalloWelt()  
{  
  var mySimpleAjax = new upSimpleAjax();  
  mySimpleAjax.oProcessFunc = hallowelt;  
  mySimpleAjax.loadJsonVm("internal/system/vm/custom/hallowelt.vm");  
  return true;  
}
```

b) Verarbeitung des Requests auf dem Server

Erstellen Sie im Verzeichnis  */org/<portal>/internal/system/vm/custom* die Datei *hallowelt.vm* und hinterlegen Sie das folgende Skript:

```
{"myobject":{"wort1":"Hallo","wort2":"Welt!"}}
```

c) Ausgabe des Ergebnisses

Hinterlegen Sie folgendes Skript im JavaScript-Editor:

```
function hallowelt(oJSON) {  
  alert(oJSON.myobject.wort1 + " " + oJSON.myobject.wort2);  
  return true;  
}
```

d) Request an den Server via Response

Um Fehler mit entsprechenden Meldungen abfangen zu können, eignet sich mit etwas mehr Aufwand der Aufbau einer vollständigen Response.

Um das testen zu können, soll geprüft werden, ob der Administrator angemeldet ist. Falls nicht, wird eine Fehlermeldung erzeugt.

Erstellen Sie dafür eine weitere Schaltfläche und hinterlegen Sie folgendes Skript in das onClick()-Ereignis.

```
function call_HalloWelt2()
{
    var l_strUrl = Helper.getBaseUrl();
    l_strUrl = Helper.setUrlValueByParam("rq_Template",
"internal/system/vm/custom/hallowelt2.vm", l_strUrl);
    var oXmlHttp = new upXmlHttp();
    oXmlHttp.bProcessResponse = true;
    oXmlHttp.processResponse =
function(){ResponseHandler.processJson(hallowelt2)};
    oXmlHttp.strUrl = l_strUrl;
    oXmlHttp.send();
    return true;
}
```

e) Verarbeitung des Requests auf dem Server

Erstellen Sie die Datei hallowelt2.vm und hinterlegen Sie das folgende Skript:

```
##Verhinderung des Antwortversands mit $Response.setIgnoreWrite(true), damit
## Velocity-Skript ausgeführt werden kann
$Response.setIgnoreWrite(true)

## Header Schreiben und Aktivieren des Antwortversands
$Response.setHeader("Cache-Control","no-cache")
$Response.setHeader("Content-Type","text/json; charset=UTF-8")
$Response.setIgnoreWrite(false)
#if($User.getLoginName() == "Administrator")
{"myobject":{"rueckgabe":"Hallo Administrator!"}}
#else
{"error":{"title":"my error title","message":"Sie sind nicht der
Administrator!"}}
#end
```

f) Ausgabe des Ergebnisses

Hinterlegen Sie folgendes Skript im JavaScript-Editor:

```
function halloWelt(oJSON)
{
    alert(oJSON.myobject.rueckgabe);
    return true;
}
```

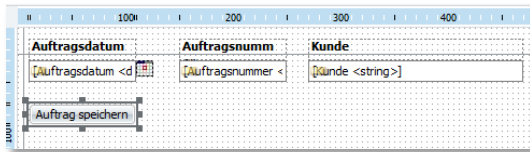
6.3.2. Beispiel 2 – Summierung

Ein klassisches Beispiel ist die Summierung der Teilbeträge in einer Positionstabelle. Das *jobject* in Tabellen ermöglicht zwar eine Summierung von Spaltenwerten, die Summenbildung kann jedoch nur für die in der Tabelle sichtbaren Datensätze durchgeführt werden.

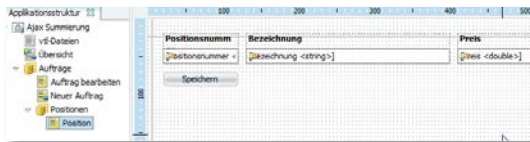
In diesem Beispiel wird gezeigt, wie eine Summierung per Ajax-Request ermittelt und angezeigt werden kann. Nach dem Hinzufügen eines Datensatzes in eine Kinddatengruppe soll die Summe der einzelnen Positionen neu berechnet und angezeigt werden. Dies soll auch beim Blättern in der Tabelle funktionieren, wenn zwischenzeitlich ein neuer Datensatz hinzugefügt wurde.

6.3.3. Aufbau der Applikation

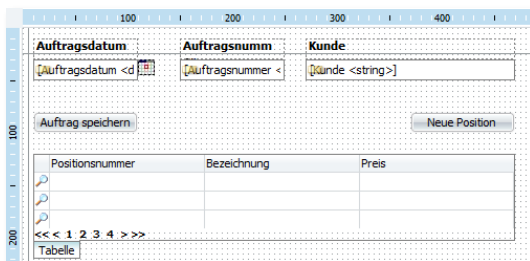
Erstellen Sie eine neue Applikation auf Basis der Vorlage *leere Applikation* mit dem Titel *Ajax Summierung*. Benennen Sie die vorhandene Datengruppe in *Aufträge* um. Die Eingabeseite wird in *Auftrag* umbenannt.



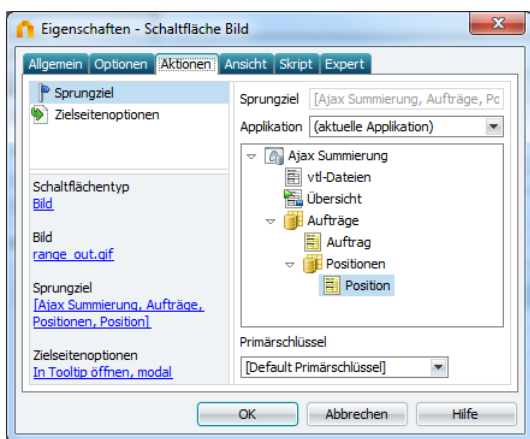
Erstellen Sie auf der Eingabeseite *Auftrag* die folgenden drei Eingabefelder: *Auftragsdatum* (Typ Datum), *Auftragsnummer* (Typ Ganzzahl) und *Kunde* (Typ Text). Platzieren Sie unterhalb der Eingabefelder eine Schaltfläche mit dem Titel *Auftrag speichern* und Sprung auf die Eingabeseite *Auftrag*.



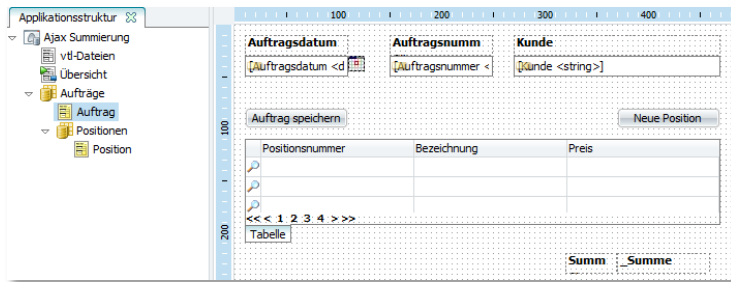
Legen Sie eine untergeordnete Datengruppe mit dem Titel *Positionen* an. Erstellen Sie unterhalb dieser Datengruppe eine Eingabeseite mit dem Titel *Position* und folgenden drei Eingabefeldern an: *Positionsnummer* (Typ Ganzzahl), *Bezeichnung* (Typ Text) und *Preis* (Typ Gleitkommazahl). Über eine Schaltfläche soll die Aktion *Speichern* ausgeführt werden und ein Sprung auf die Seite *Auftrag bearbeiten* erfolgen. Popup/Tooltip wird dabei geschlossen.



Erstellen Sie auf der Eingabeseite *Auftrag* eine Ansichtstabelle mit den zugehörigen Positionen. Sprungziel ist die Seite *Position*. Erstellen Sie eine Schaltfläche mit dem Titel *Position hinzufügen* und dem Aktionstyp *Speichern*. Sprungziel ist die Eingabeseite *Position* mit der Option *Neuen Datensatz anlegen*. Die Seite wird im Tooltip geöffnet.



Wechseln Sie auf die Eingabeseite *Auftrag* und korrigieren Sie den Sprung der Schaltfläche *Auftrag speichern* zur Zielseite *Positionen*. Ein neuer Datensatz wird dabei angelegt. Die Zielseite wird in einem modalen Tooltip geöffnet.



Speichern und testen Sie die Applikation.

6.3.4. Summierung der Preise

Für die Ausgabe der Summe mittels Ajax, werden auf der Eingabeseite *Auftrag bearbeiten* zwei statische Textfelder rechts unterhalb der Positionstabelle benötigt. Das erste Feld enthält den Titel *Summe*; das zweite Feld den Text *_Summe*. Dieses Feld wird zur Anzeige der Spaltensumme *Preis* verwendet.

Request an den Server

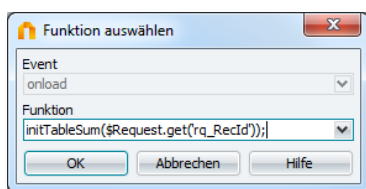
Kopieren Sie die folgende Funktion in den JavaScript-Editor von Intrexx .

```
function initTableSum(pkid) {
    var mySimpleAjax = new upSimpleAjax();
    mySimpleAjax.oProcessFunc = zeigeSumme;

    mySimpleAjax.loadJsonVm("internal/application/resource/64A5833C698BEFA06A87D87C9CB7EE26307A4347/tableSum.vm", {rq_fklid: pkid});
    return true;}

```

Die Datensatz-ID des Elterndatensatzes wird dem Aufruf der *tableSum.vm* mit der Funktion *initTableSum()* mitgegeben.



Hinterlegen Sie die Funktion im *onload()* der Seite *Auftrag bearbeiten*, und übergeben Sie die notwendigen Parameter.

In der Ansichtstabelle muss die Funktion zur Request-Anfrage über das Expert-Attribut *onload* ausgeführt werden. Dies stellt beim Blättern und Sortieren eine Aktualisierung der Summe sicher. Wie bei der Tabelle selbst wird kein kompletter Seiten-Refresh ausgeführt, sondern nur das Summenfeld neu geschrieben. Erstellen Sie auf dem Reiter *Expert* der Ansichtstabelle das Expert-Attribut *onload*. Als Wert wird die Funktion übergeben.

Verarbeitung des Requests auf dem Server

Für die Summenbildung wird das entsprechende SQL-Statement, das als PreparedQuery auf dem Server ausgeführt werden soll, benötigt. Das folgende Grundkonstrukt wird für das Beispiel verwendet:

```
SELECT SUM(<Feldname>)
FROM <Datengruppe>
WHERE FKLID = <LID des Elterndatensatzes>;
```

Erstellen Sie die Datei `tableSum.vm` und hinterlegen Sie das folgende Skript:

```
#####
## Zugriff auf die aktuelle/eigene PortalDatenbank:
#####
## Zugriff auf die aktuelle/eigene PortalDatenbank:
## $DbUtil.getConnection("IxSysDb")
## $DbUtil.getConnection("?")
## Kurzform/Sonderform für Zugriff auf die aktuelle/eigene PortalDatenbank:
## $DbConnection
## Zugriff auf fremde Quellen:
## $DbUtil.getConnection("MeineDatenquelle")
## $DbUtil.getConnection("ERP")
#####

## Requestparameter auslesen
#set($FkLid = $Request.get("rq_fklid"))
#set($Spaltensumme = 0)

##ValueHolder
#set($vhFkLid = $VH.getVH($FkLid))

#set($colName = "FLT_PREIS_480357E8")
#set($tableName =
$RtCache.getDataGroup('A8D2E05273AC3F221732FDC2FF015F8694819CB3').getTableName())

## Prepared Query auf Kindtabelle
#set($l_statement = $PreparedQuery.prepare($DbConnection, "SELECT
SUM($colName) FROM $tableName WHERE FKLID = ?"))

## Einsetzen des Ganzzahlwertes (Valueholder) in das SQL-Statement
$l_statement.setInt(1, $vhFkLid)

## Ausführen des Query - Ergebnis
#set($Spaltensumme = $l_statement.executeAndGetScalarValue(0))

## Schliessen der Abfrage und des Resultsets
$l_statement.close()
{"myobject":{"summe":$Spaltensumme}}
```

Die Velocity-Datei `tableSum.vm` liest zuerst den übergebenen Requestparameter aus und legt ihn in einer Velocity-Variablen ab. Über `$RtCache.getDataGroup()` lässt sich der Tabellename der Datengruppe mit der entsprechenden GUID auslesen. Anschließend wird eine Prepared Query initialisiert und das SQL unter Verwendung der Requestwerte ausgeführt. Nach dem Auslesen des Summenwertes wird dieser über JSON bereitgestellt.

Ausgabe des Ergebnisses

Die über die Funktion `initTableSum()` initialisierte Funktion `zeigeSumme()` wartet auf die Antwort und verarbeitet diese entsprechend.

Kopieren Sie die folgende Funktion in den JavaScript-Editor von Intrexx .

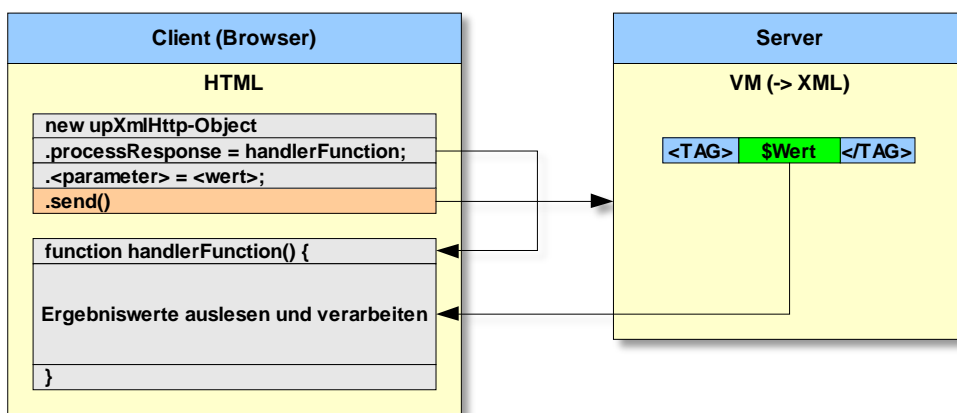
```
function zeigeSumme(oJSON)
{
  Browser.setValue(getElement("3A..5"), oJSON.myobject.summe);
  return true;
}
```

Tragen Sie die GUID des statischen Textfeldes an der gekennzeichneten Stelle ein. Speichern und Testen Sie die Applikation.

6.4. Ajax in Intrexx mit XML-Antwort

Für das *upXmlHttp*-Objekt muss zunächst auf der entsprechenden Intrexx-Seite eine neue Instanz erstellt werden. Für diese Objekt-Instanz werden dann verschiedene Parameter definiert, darunter auch die Funktion, welche die Antwort des Requests ausliest und verarbeitet sowie die VM-Datei, welche den Request serverseitig behandelt und die Ergebnisse in XML-Form bereitstellt.

Liefert der Server eine Antwort auf den Request, wird das *processResponse*-Ereignis im Browser ausgelöst und die Handler-Funktion ausgeführt. Über das *oRequest*-Objekt können nun die Ergebniswerte ausgelesen und verarbeitet werden.



i Im Internet-Explorer 6 wird das XmlHttpRequest-Objekt über ein ActiveX realisiert. Daher ist diese Ajax-Funktionalität nur mit „aktiviertem“ ActiveX nutzbar. Ab Version 7 ist dies nicht mehr der Fall.

6.4.1. Beispiel 1 – Hallo Welt


6.4.2. Request an den Server via ActionControl

Als Vergleich zur Verarbeitung mit JSON ist hier das Hallo-Welt-Beispiel ohne SimpleAjax und JSON umgesetzt.

```
function call_HalloWelt()
{
    var oHallowelt = new upActionControl();
    oHallowelt.oHtml = new Object();
    oHallowelt.oTarget = new upTarget();
    oHallowelt.oTarget.rq_Template =
"internal/system/vm/html/include/hallowelt.vm";
    oHallowelt.requestType = 2; /* Methode GET */
    oHallowelt.oXmlHttp.bAsync = true;
    oHallowelt.oXmlHttp.bProcessResponse = true;
    oHallowelt.oXmlHttp.processResponse = hallowelt;
    oHallowelt.processRequest();
    return true;
}
```

Eigenschaften des upXmlHttp-Objekts		
Eigenschaft	Beschreibung	Werte
bAsync	Asynchrone Kommunikation false = synchrone Kommunikation (Achtung:Endlos-Schleifen möglich)	true, false
bProcessResponse	Initialisierung	true, false
processResponse	Zuweisung der Handlerfunktion, welche bei Auslösen des Request- Antwort- Ereignisses abgearbeitet werden soll.	Funktionsname mit oder ohne ResponseHandler

6.4.3. Verarbeitung des Requests auf dem Server

Erstellen Sie im Verzeichnis  /org/<portal>/internal/system/vm/html/include die Datei hallowelt.vm und hinterlegen Sie das folgende Skript:

```
## Verhinderung des Antwortversands mit $Response.setIgnoreWrite(true)
## damit Velocity-Skript ausgeführt werden kann
$Response.setIgnoreWrite(true)

## Variable mit Text füllen
#set($Hallowelt = "Hallo Welt!")

## Header Schreiben und Aktivieren des Antwortversands
$Response.setHeader("Cache-Control", "no-cache")
$Response.setHeader("Content-Type", "text/xml; charset=UTF-8")
$Response.setIgnoreWrite(false)<?xml version="1.0" encoding="UTF-8"?>

## Erzeugen des XML
<response>
  <hallowelt>$Hallowelt</hallowelt>
</response>
```

6.4.4. Ausgabe des Ergebnisses

Hinterlegen Sie folgendes Skript im JavaScript-Editor:

```
function hallowelt() {
  // Abfrage, ob Request abgeschlossen
  if(oRequest.readyState == 4) {
    // Abfrage, ob Status "OK"
    if(oRequest.status == 200){
      var referenz = oRequest.responseXML.documentElement;
      // Auslesen des XML-Tags
      var antwort =
referenz.getElementsByTagName('hallowelt')[0].firstChild.data;
      alert(antwort);
    }
    else
    {
      // Im Fehlerfall (status <> 200) Fehlermeldung
      alert("Http Error " + oRequest.status + "\n" + oRequest.statusText +
"\nfunction: displayResult");
    }
  }
  return true;
}
```

Die Funktion liest aus dem zurück übermittelten XML den Inhalt des Tags `<hallowelt>` aus und übergibt dieses in die JavaScript-Variable `antwort`. Die Variable wird anschließend mit der Methode `alert()` angezeigt. Speichern und testen Sie ihre Applikation.

ReadyState-Codes	
Code	Beschreibung
0	Nicht initialisiert
1	Objekt ist bereit, keine Daten gesendet
2	Anfrage wurde gesendet
3	Daten werden empfangen (onreadystatechange wird evtl. mehrmals aufgerufen)
4	Alle Daten wurden empfangen

HTTP-Kommunikationscodes	
Code	Beschreibung
100	Bearbeitung wird durchgeführt
200	Erfolg
300	Weiterleiten
400	Client-Fehler
500	Server-Fehler

7. Anhang - Velocity-Referenz

7.1. Variablen

```
$foo
$a

#set( $a = "Velocity" )
$a ist der Name der Variablen, Velocity der Wert.

Beispiel
<html>
<body>#set( $foo = "Velocity" )
Hello $foo World!
</body>
</html>
```

7.2. Properties

```
$customer.Address
$purchase.Total
```

7.3. Methoden

```
customer.getAddress()
$purchase.setTitel("My Home Page")
$person.setAttributes(["gross", "klein", "mittel"])
```

Formale Referenz Notation

`${customer.address}`

Die formale Notation ist notwendig für folgenden Text:

Dies ist ein `${art}`fall. --- `${art}` wird nun als Variable behandelt und ersetzt.

Stille Referenz Notation

Wenn Velocity auf eine undefinierte .Referenz trifft, wird einfach der Name der Referenz ausgegeben.

Beispiel:

```
<input type="text" name="email" value="$email"/>
```

Wenn `$email` keinen Wert hat, sollte besser auch der *value* leer bleiben. Dies erreicht man durch folgende Notation: `<input type="text" name="email" value="$!email"/>`

Kombiniert mit der formalen Notation ist auch folgende Notation möglich:

```
<input type="text" name="email" value="$!{email}"/>
```

Escaping Referenzen

```
#set($email = "foo")
```

```
$email
```

```
\$email
```

```
\\$email
```

```
\\\ $email
```

Die Ausgabe sähe so aus:

```
foo
```

```
$email
```

```
\foo
```

```
\\$email
```

Ist `$email` nicht definiert:

```
$email
```

```
\$email
```

```
\\$email
```

```
\\\ $email
```

7.4. Kommentare

Kommentare sind in der Ergebnisdatei nicht sichtbar.

```
## Dies ist ein einzeiliger Kommentar.  
#*  
Dies ist ein mehrzeiliger Kommentar, der im Web nicht angezeigt wird.  
*#
```

7.5. #set

Die `#set` Anweisung weist einer Referenz ihren Wert zu.

```
#set ( $eigenschaft = "freundlich" )  
#set ( $customer.behavior = $eigenschaft )
```

Links muss eine Variable oder Property Referenz angegeben werden.

Auf der rechten Seite sind möglich:

Variablen Referenz
Zeichenkette
Property Referenz
Methoden Referenz
Nummer
ArrayList
Map

Beispiel für Nummer: `#set ($nummer = 123)`
ArrayList: `#set ($array = ["Not", $my, "fault"])`
Map: `#set ($map = { "banana": "good", "roast beef": "bad" })`

Hinweis: Zugriff auf die ArrayList über

`$array.get(index)`, d.h. `$array.get(0)` liefert den Wert 'Not'.

Zugriff auf die Map über

`$map.get(name)`, d.h. `$map.get("banana")` liefert den Wert ,good'.

Rechts können Berechnungen stehen:

`#set ($value = $foo + 4)`

Der return Wert null überschreibt **nicht** den Inhalt einer Referenz, d.h.

```
#set ( $result = $query.criteria("name" ) )  
#set ( $result = $query.criteria("adresse" ) )
```

Wenn `$query.criteria("adresse")` als Wert null zurückliefert, behält `$result` den Wert aus dem ersten `#set`, also den Wert, den `$query.criteria("name")` geliefert hat, z.B. Maier.

Beispiel: für eine for-each sollte daher Folgendes verwendet werden

```
#set( $criteria = [ "name", "address" ] )  
  
#foreach( $criterion in $criteria )  
  
    #set( $result = false )  
    #set( $result = $query.criteria($criterion) )  
  
    #if( $result )  
        Query was successful  
    #end  
  
#end
```

Für Stringlitterale gilt:

Verwenden von doppelten Hochkommata ""

```
#set( $directoryRoot = "www" )  
#set( $templateName = "index.vm" )  
#set( $template = "$directoryRoot/$templateName" )  
$template
```

Die Ausgabe ist:

www/index.vm

Verwenden von einfachen Hochkommata , ' - es findet kein parsing statt, d.h. Referenzen werden nicht aufgelöst.

```
#set( $foo = "bar" )
$foo
#set( $blargh = '$foo' )
$blargh
```

Die Ausgabe ist:

```
bar
$foo
```

7.6. #if / #else / #elseif

7.6.1. #if

```
#if( $foo )
  <strong>Velocity!</strong>
#end
```

7.6.2. #else

```
#set ( $foo = "deoxyribonucleic acid" )
#set ( $bar = "ribonucleic acid" )

#if ( $foo == $bar )
  In this case it's clear they aren't equivalent. So...
#else
  They are not equivalent and this will be the output.
#end
```

7.6.3. #elseif

```
#if( $foo < 10 )
  <strong>Go North</strong>
#elseif( $foo == 10 )
  <strong>Go East</strong>
#elseif( $bar == 6 )
  <strong>Go South</strong>
#else
  <strong>Go West</strong>
#end
```

7.7. Relationale und logische Operatoren

Bedingungen

Gleichheits Operator: #if(\$foo == \$bar)

Größer als: #if(\$foo > 42)

Kleiner als: #if(\$foo < 42)

Größer gleich: #if(\$foo >= 42)

Kleiner gleich: #if(\$foo <= 42)

Gleiche Zahl: #if(\$foo == 42)

Gleiche Zeichenkette: #if(\$foo == "bar")

Logisches NOT: #if(!\$foo)

logical AND

```
#if( $foo && $bar )
  <strong> This AND that</strong>
#end
```

logical OR

```
#if( $foo || $bar )
  <strong>This OR That</strong>
#end
```

logical NOT

```
#if( !$foo )
  <strong>NOT that</strong>
#end
```

7.8. #foreach

ArrayList:

```
#set ( $allProducts = [ "Handschuh", "Mütze", "Stiefel" ] )
<ul>
#foreach( $product in $allProducts )
  <li>$product</li>
#end
</ul>
```

Automatische Zählvariable: \$velocityCount (beginnt bei 1)

Map:

```
#set ( $allProducts = { "banana" : "good", "roast beef" : "bad" } )
<ul>
#foreach( $key in $allProducts.keySet() )
  <li>Key: $key -> Value: $allProducts.get($key)</li>
#end
</ul>
```

7.9. #include

Erlaubt den Import einer lokalen Datei.

```
#include( "one.txt" )
```

7.10. #parse

Erlaubt das Ausführen einer Velocity Datei.

```
#parse ( "statische.vm" )
#parse ( $varfürDateiname )
```

7.11. #macro

Erlaubt das Definieren von VTL Teilabschnitten, die beliebig oft wiederholt werden können.

Definieren des Macros

```
#macro( d )  
<tr><td></td></tr>  
#end
```

Aufruf des Macros

```
#d()
```

Definieren des Macros mit Parametern

```
#macro( tablerows $color $somelist )  
#foreach( $something in $somelist )  
  <tr><td bgcolor=$color>$something</td></tr>  
#end  
#end
```

Aufruf des Macros mit Parametern

```
#set( $greatlakes = [ "Superior", "Michigan", "Huron", "Erie", "Ontario" ] )  
#set( $color = "blue" )  
<table>  
  #tablerows( $color $greatlakes )  
</table>
```

Ergebnis

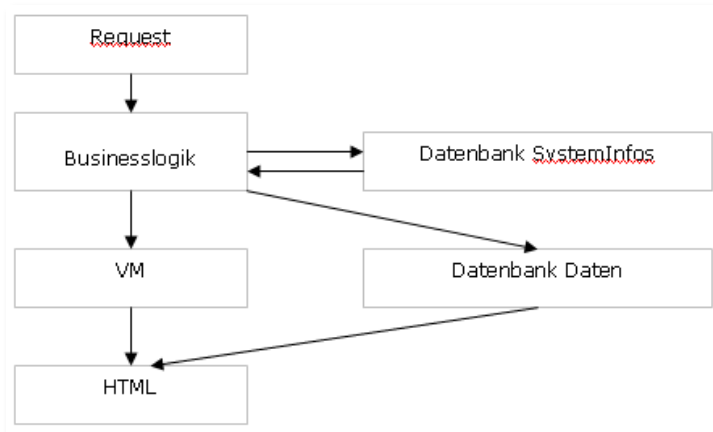
```
<table>  
  <tr><td bgcolor="blue">Superior</td></tr>  
  <tr><td bgcolor="blue">Michigan</td></tr>  
  <tr><td bgcolor="blue">Huron</td></tr>  
  <tr><td bgcolor="blue">Erie</td></tr>  
  <tr><td bgcolor="blue">Ontario</td></tr>  
</table>
```

7.12. Range Operator

```
[n..m]  
  
#foreach( $foo in [1..5] )  
$foo  
#end  
  
#foreach( $bar in [2..-2] )  
$bar  
#end  
  
#set( $arr = [0..1] )  
#foreach( $i in $arr )  
$i  
#end
```

7.13. Aufruf einer Seite (2. Transformation)

Beim Aufruf einer Applikationsseite



8. Anhang - ValueHolder

Valueholder besitzen ein schlankes Interface, das es ermöglicht, Java-Datentypen zu kapseln und um folgende Funktionen zu erweitern:

- `hasValue()`
boolean
zeigt an, ob der Valueholder einen Wert besitzt
- `getValue()`
Object
Gibt den entsprechenden Object-Datentyp zurück.
- `getCanonicalLexicalRepresentation()`
String
Gibt eine kanonisch-lexikografische Entsprechung des Objektes zurück.
- `getType()`
QName
Qualifizierter Name:
<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>
- `getRawValue()`
Object
Das gekapselte Objekt
- `setRawValue(Object)`
void
Setzt den Wert für einen Valueholder

Der Vorteil beim Einsatz von Valueholdern liegt in der transparenten Verwendung beim Einsatz innerhalb der Businesslogiken oder des Velocitykontextes und in der Möglichkeit, die Performance und das Speicherverhalten positiv zu beeinflussen.

8.1. Typen von Valueholdern

8.1.1. BooleanValueHolder

Method	Datentyp	Beispiel
getValue()	Boolean	True / False
getCanonicalLexicalRepresentation()	String	true/false

8.1.2. DateTimeValueHolder

Method	Datentyp	Beispiel
getValue()	Date	
getCanonicalLexicalRepresentation()	String	2002-10-10T12:00:00Z

8.1.3. DoubleValueHolder

Method	Datentyp	Beispiel
getValue()	Double	
getCanonicalLexicalRepresentation()	String	-1E4,1267.43233E12, 12.78e-2, 12

8.1.4. FileValueHolder

Method	Datentyp	Beispiel
getValue()	VHFileDescriptor	
getCanonicalLexicalRepresentation()	UnsupportedOperation Exception	
getFileName()	String	test.txt
getPath()	String	c:\tmp\test.txt
getContentType()	String	text/plain

8.1.5. ImageValueHolder

Method	Datentyp	Beispiel
getValue()	VHFileDescriptor	
getCanonicalLexicalRepresentation()	Unsupported OperationExcepti on	
getFileName()	String	image.gif
getPath()	String	c:\tmp\image.gif
getContentType()	String	image/gif
getScaledImagePath()	String	c:\tmp\image_50x50.gif
getConvertString()	String	scaled=true;width=...
getImageSize()	Dimension	100 / 100
getScaledImageSize()	Dimension	50 / 50

8.1.6. LongValueHolder

Methode	Datentyp	Beispiel
getValue()	Long	
getCanonicalLexicalRepresentation()	String	-1, 0,12667543233, +100000

8.1.7. StringValueHolder

Methode	Datentyp	Beispiel
getValue()	String	
getCanonicalLexicalRepresentation()	String	"Hello World"

8.2. Spezielle Valueholder

8.2.1. NewGuidValueHolder

Methode	Datentyp	Beispiel
getValue()	String	Erzeugt neue Guid
getCanonicalLexicalRepresentation()	String	"0B8C0112D6C59...3F868E1118"

8.2.2. NullValueHolder

Methode	Datentyp	Beispiel
getValue()	Null	
getCanonicalLexicalRepresentation()	Null	

8.2.3. NowValueHolder

Methode	Datentyp	Beispiel
getValue()	Date	Aktuelles Datum
getCanonicalLexicalRepresentation()	String	2002-10-10T12:00:00Z

9. Anhang - Renderer

Renderer dienen dazu, Datenwerte für den Benutzer so aufzubereiten, dass diese im gewünschten Format angezeigt werden. Die Anforderungen an die Darstellung sind äußerst vielschichtig und können über eine Vielzahl von Parametern definiert werden. So müssen z.B. Datumswerte oder Zahlenwerte in der entsprechenden Länderdarstellung aufbereitet, Bilder skaliert, HTML-Textwerte als HTML oder als Plaintext gerendert werden und vieles mehr.

Um diesen Anforderungen Rechnung zu tragen, gibt es eine Vielzahl von Renderern, die diese Aufgaben übernehmen können.

Renderer arbeiten nur mit Valueholdern zusammen (📖 *ValueHolder* aus der Intrexx Entwicklerdokumentation).

9.1. Basisfunktionalitäten von Renderer

Methode: `writeOutput(Writer p_out, IValueholder <?> p_value)`

Beschreibung: Schreibt den formatierten Wert direkt auf den Ausgabestream.

Methode: `getOutput(IValueholder <?> p_value)`
Beschreibung: Gibt einen String mit dem formatierten Wert zurück.

Methode: `setParam(String p_strName, Object p_Value)`
Beschreibung: Kann bestimmtes Defaultverhalten entsprechend beeinflussen.

9.2. Typen von Renderer

Für nahezu jeden Typ von Valueholder gibt es entsprechende Renderer, für den Typ *StringValueHolder* gibt es sogar mehrere Renderer, die zudem noch hintereinander geschaltet werden können, um komplexere Formatierungen zu ermöglichen.

Im Folgenden werden die wichtigsten Renderer vorgestellt:

Renderer	Beschreibung
BooleanRenderer	Zur Anzeige von Boolean-Werten
BRInsertRenderer	Ersetzt Zeilenumbrüche in Plaintext durch das HTML-Tag <code>
</code>
ColorRenderer	Zur Anzeige von Hexadezimal codierten Farbwerten und durch das W3C definierte Farbnamen.
CurrencyRenderer	Zur Anzeige von Währungs-Werten
DateTimeRenderer	Zur Anzeige von Datums-Werten
DoubleRenderer	Zur Anzeige von Fließkomma-Werten
DurationRenderer	Zur Anzeige von einer Zeitdauer aus einem long-Wert
HtmlEncodingRenderer	Zur Anzeige von HTML-Quelltext im Browser
HtmlMarkupStrippingRenderer	Zum Entfernen jeglicher HTML-Tags
HtmlOutputLengthLimitRenderer	Zum Beschneiden von HTML-codiertem Text
IntegerRenderer	Zur Anzeige von Ganzzahl-Werten
JavaScriptDateTimeRenderer	Zur Anzeige von Datums-Werten in der Form <code>yyyy,MM,dd,HH,mm,ss</code>
JavaScriptLiteralRenderer	Ersetzt im Wesentlichen Anführungszeichen, Apostrophe, etc. durch entsprechende Javascript-Escape-Sequenzen
MailToRenderer	Ergänzt einen String um ein eventuell fehlendes mailto: -Prefix
OutputLengthLimitRenderer	Kann Texte in der Länge beschränken
ToolTipRenderer	Kann Texte für die Ausgabe in Tooltips aufbereiten (unter Beachtung von JavaScript, etc.)

9.3. Erzeugen von Renderern im Velocity Context

Es gibt im Wesentlichen zwei Methoden für den Einsatz von Renderern im Velocity-Context:

- Zugriff auf Defaultrenderer
- Erzeugen eines neuen Renderers

Je nach Anwendungsfall kann die eine oder andere Methode gewählt werden.

9.4. Zugriff auf Defaultrenderer

Auf Defaultrenderer können Sie immer dann zugreifen, wenn

- Sie Ihre Velocity-Datei direkt innerhalb einer Intrexx-Seite platzieren
- Ihnen die Funktionalitäten des Defaultrenderer genügen

Folgende Defaultrenderer stehen Ihnen zur Verfügung:

9.4.1. `$DefaultHtmlEncodingRenderer`

Formatiert einen `StringValueHolder` so, dass HTML-Tags auf der Webseite sichtbar werden. Das Zeichen `<` wird als `<` formatiert. Ist der String leer oder `Null` wird das HTML-Entity ` ` als Defaultwert eingesetzt.

9.4.2. `$DefaultJsHtmlEncodingRenderer`

Formatiert einen `StringValueHolder` so, dass HTML-Tags auf der Webseite sichtbar werden. Das Zeichen `<` wird als `<` formatiert. Außerdem werden alle Zeichen, die in einem JavaScript-Funktionsaufruf oder einer Zuordnung (Anführungszeichen, etc.) zu Fehlern führen könnten, entsprechend maskiert.

9.4.3. `$DefaultHtmlEncodingRendererEdit`

Formatiert einen `StringValueHolder` so, dass HTML-Tags auf der Webseite sichtbar werden. Das Zeichen `"<"` wird als `"<"` formatiert. Ist der String leer oder `Null` wird `""` als Defaultwert eingesetzt.

9.4.4. `$DefaultSimpleTextRenderer`

Formatiert einen `StringValueHolder`. Ist der String leer oder `Null` wird `""` als Defaultwert eingesetzt. Zeilenumbrüche werden als `
` umgesetzt.

9.4.5. `$DefaultDateTimeRenderer`

Formatiert einen `DateTimeValueHolder` so, dass der Wert als String entsprechend der eingestellten Ländereinstellung formatiert wird. Der Wert wird als Datum mit Zeitanteil ausgegeben und in die Zeitzone des Benutzers umgesetzt. Ist der Wert leer oder `Null` wird ` ` als Defaultwert eingesetzt.

9.4.6. `$DefaultDateRenderer`

Formatiert einen `DateTimeValueHolder` so, dass der Wert als String entsprechend der eingestellten Ländereinstellung formatiert wird. Der Wert wird als Datum ausgegeben und in die Zeitzone des Benutzers umgesetzt. Ist der Wert leer oder `Null` wird ` ` als Defaultwert eingesetzt.

9.4.7. \$DefaultTimeRenderer

Formatiert einen `DateTimeValueHolder` so, dass der Wert als String entsprechend der eingestellten Ländereinstellung formatiert wird. Der Wert wird als Zeitanteil ausgegeben und in die Zeitzone des Benutzers umgesetzt. Ist der Wert leer oder Null wird * * als Defaultwert eingesetzt.

9.4.8. \$DefaultIntegerRenderer

Formatiert einen `LongValueHolder`, um diesen mit dem Tausendertrennzeichen entsprechend der eingestellten Ländereinstellung auszugeben. Ist der Wert leer oder Null wird * * als Defaultwert eingesetzt.

9.4.9. \$DefaultCurrencyRenderer

Formatiert einen `DoubleValueHolder`, um diesen mit den Tausender- und Dezimaltrennzeichen entsprechend der eingestellten Ländereinstellung auszugeben. Ist der Wert leer oder Null wird * * als Defaultwert eingesetzt.

9.4.10. \$DefaultNumberRenderer

Formatiert einen `DoubleValueHolder`, um diesen mit den Tausender- und Dezimaltrennzeichen entsprechend der eingestellten Ländereinstellung auszugeben. Ist der Wert leer oder Null wird * * als Defaultwert eingesetzt.

9.4.11. \$DefaultBooleanRenderer

Formatiert einen `BooleanValueHolder`. Der Wert wird als *true* für Wahr oder *false* für Unwahr zurückgegeben. Ist der Wert leer oder Null wird * * als Defaultwert eingesetzt.

9.4.12. \$SimpleTextRenderer

Gibt einen `StringValueHolder` ohne weitere Manipulation der Daten aus. Ist der Wert leer oder Null wird "" als Defaultwert eingesetzt.

9.4.13. \$DefaultJsIntegerRenderer

Formatiert einen `LongValueHolder`, um diesen ohne Tausendertrennzeichen und dem Dezimaltrennzeichen in JavaScript-Konvention auszugeben. Ist der Wert leer oder Null wird 0 als Defaultwert eingesetzt.

9.4.14. \$DefaultBooleanAsIntRenderer

Formatiert einen `BooleanValueHolder`, um diesen als Integerwert z.B. in Javascript oder Velocity weiter zu verwenden. Der Rückgabewert ist 1 für Wahr und 0 für Unwahr oder Null.

9.4.15. \$DefaultJsNumberRenderer

Formatiert einen `DoubleValueHolder`, um diesen ohne Tausendertrennzeichen und dem Dezimaltrennzeichen in JavaScript-Konvention auszugeben. Ist der Wert leer oder Null wird 0 als Defaultwert eingesetzt.

9.4.16. \$DefaultJsEncodingRenderer

Formatiert einen `StringValueHolder`. Alle Zeichen, die in einem JavaScript-Funktionsaufruf oder einer Zuordnung (Anführungszeichen, etc.) zu Fehlern führen könnten, werden entsprechend maskiert. Ist der String leer oder Null wird "" als Defaultwert eingesetzt.

9.4.17. \$DefaultTooltipRenderer

Formatiert einen StringValueHolder so, dass der Text problemlos in einem Tooltip angezeigt werden kann. Alle Zeichen, die in einem JavaScript-Funktionsaufruf oder einer Zuordnung (Anführungszeichen, etc.) zu Fehlern führen könnten, werden entsprechend maskiert.

Anwendungsbeispiel:

Bei der Ausgabe innerhalb von Velocity wird eine Variable aus dem aktuellen Datensatz anhand eines *sysidents* des Datenfeldes gesetzt.

```
#set ($l_vhInteger = $DC.getValueHolderBySysident ('SYSIDENT'))
```

Definieren einer Variablen mit einem Writer-Objekt direkt auf den Ausgabestream.

```
#set ($l_writer = $Response.getWriter())
```

Der Output wird über den Renderer formatiert auf den Ausgabestream ausgegeben.

```
$DefaultBooleanRenderer.writeOutput($l_writer, $l_vhInteger)
```

Der Output wird der Velocity-Variablen *\$l_myVar* zugewiesen und kann dann innerhalb von Velocity weiterverarbeitet werden.

```
#set($l_myVar = $DefaultBooleanRenderer.getOutput($l_vhInteger))
```

9.5. Direktes Erzeugen eines Renderers

Wird ein Renderer in einem anderem Kontext als direkt in einer vom Applikationsdesigner erzeugten Seite benötigt, z.B. in einer Velocity Seite, die über AJAX aufgerufen wird oder wird ein spezieller Renderer benötigt, kann ein Renderer auch direkt erzeugt werden. Dafür gibt es in Intrexx Xtreme eine spezielle Factory-Klasse, die nahezu alle benötigten Renderer erzeugen kann.

Die Factory-Klasse steht innerhalb des Velocity-Kontexts *\$RendererFactory* zur Verfügung. Sie verfügt über folgende Methoden für die Erzeugung von Renderern.

9.5.1. Erzeugen eines Datumsrenderers

```
#set ($l_renderingctx =
$RendererFactory.getDefaultRenderingContext($Request, $Session, $User,
$lang))

#set ($l_DateRenderer =
$RendererFactory.createDefaultDateTimeRenderer($l_renderingctx))

## Ausgabe
$l_DateRenderer.writeOutput($Response.getWriter(), $l_ValueHolder)
```

9.5.2. Erzeugen eines Datumsrenderers mit Formatierung

```
#set ($l_renderingctx =
$RendererFactory.getDefaultRenderingContext($Request, $Session, $User,
$lang))
```

```
#set ($l_DateTimeRenderer =
$RendererFactory.createDateTimeRendererWithParameters($l_renderingctx,
false, false, null, "EEEE'<br />'yyyy-MM-dd", "", null))
```

Spezielle Formatierung: CW für Kalender-Woche

Format: "CW dd.MM.yyyy hh.mm.ss"

Output: "1 01.01.2000 10.00.00"

Format: CW

Output: "1"

Format: dd.MM.yyyy hh.mm.ss CW

Output: "01.01.2000 10.00.00 1"

Die Formatierung des Datums kann mit folgenden Formatdirektiven beeinflusst werden:

Letter	Datums- oder Zeitbestandteil	Typ	Beispiel
G	Era designator	Text	AD
Y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

9.5.3. Erzeugen eines komplexen Renderers

HTMLEncoding, mit Ersetzen von Zeilenumbrüchen durch `
` und einer Längenlimitierung auf 300 Zeichen.

```
#set ($l_TextRenderer =
$RendererFactory.createTextAreaHtmlRendererWithParameters(true, true, true,
"&nbsp;", true, 300, "..."))

## Ausgabe
$l_TextRenderer.writeOutput($Response.getWriter(), $l_ValueHolder)
```

9.5.4. IRenderer createDefaultUriRenderer()

Erzeugt einen Renderer zur Ausgabe einer URL. Wird kein Parameter gesetzt, wird *http://* dem Wert vorangestellt.

Beispiel:

```
#set($l UriRenderer = $RendererFactory.createDefaultUriRenderer())  
$l UriRenderer.setParam($l UriRenderer.PARAM_SCHEMA, "callto://")
```