

—BONUSKAPITEL—

Oberflächen-
programmierung
mit
Web Dynpro ABAP
(GUI III)

Ravioli, füllungs- orientierte Pasta

SAP-Bildschirme im Webbrowser angezeigt, ganz ohne HTML-Programmierung! Schrödinger staunt nicht schlecht, wie komfortabel und einfach es ist, Webanwendungen in ABAP zu programmieren. Die ältere Lösung dafür heißt: Web Dynpro ABAP! Das gefällt Schrödinger noch mehr, weil ihm jetzt die Kenntnisse aus der Ravioli-Programmierung (im Gegensatz zur Spaghetti-Programmierung) zugutekommen. So macht Objektorientierung richtig Spaß! Und es wird noch besser, als der Schwaiger Roland sagt: »Schrödinger, heute wird's entspannend!«.

Das kann doch nicht wirklich wahr sein, oder?

Das Bonuskapitel für zu Hause.

Ohne Fleiß keine Entspannung, oder besser: Wer weiß schon, was Entspannung ist, wenn er keine Belastung kennengelernt hat?



*Ich wusste es doch: Kann hat der
Schwaiger Roland zweimal von Ent-
spannung gesprochen, muss ich schon
wieder etwas über Belastung hören.*

Du glaubst also nicht, dass auch ich entspannt sein kann?
Da wirst du aber staunen! Ich werde dir sogar noch einen
Entschleunigungsplan verpassen. Hoffentlich kommst du noch
zurück auf deinen Stuhl, nach all der Entspannung.

Wo ist der Haken?

Kein Haken. Wir werden eine Anwendung in Web
Dynpro ABAP entwickeln. Wie der Name schon andeutet, geht es
um eine Anwendung, die im Webbrowser läuft. Und das alles in
ABAP entwickelt, deiner **Lieblingsprogrammiersprache!**

*Mein Lieblingsfremdwort heißt übrigens
„subliminal“ ...*

Außerdem schauen wir uns noch an, wie SAP die Architektur von
Web Dynpro konzipiert hat und welche Teile du für deine
Anwendung nutzen kannst. Die ich übrigens so getauft habe: **Ent-
spannungsplaner!**

Der Entspannungsplaner hat einige technische **Schmankerl** auf Lager: Scroll-Bereiche,
Grafiken, Drag & Drop (auf den ersten Blick nur schwer zu sehen, ich weiß), Eingabefelder,
Texte und noch viel mehr unter der Haube – oder sollte ich sagen: unter
der Oberfläche?

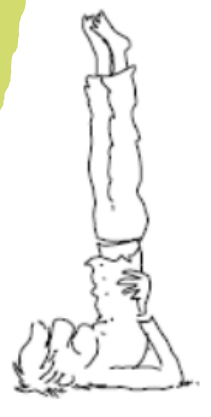
Ich bin begeistert! Aber wohes hast du die Bilder von mir??

http://was.facet.at:8000/sap/bc/webdynpro/sap/zwcdc_sch_014_s facet.at

ENTSPANNUNGSPLAN für BOUSER




Entspannungskatalog


Dein Entspannungsplan

16.05.2010

Entspannung 1

Kniebeuge mit Fingerkrachen

Entschleunigt um 50 %



In der Ausgangsstellung stehst du mit annähernd gestreckten Beinen schulterbreit. Wichtig ist, dass es zu keinen unangenehmen Verspannungen und Schmerzen führt. Die Ellenbogen werden nach vorne genommen, die Hände sind ineinander verdreht verknottet. Aus dieser Stellung gehst du in die Knie, bis die Oberschenkel die Waagerechte erreicht haben, ein rechter Winkel zwischen Ober- und Unterschenkel entsteht und diese richtig zu brennen beginnen. Bei Schmerzen im Kniegelenk solltest du die Bewegung nicht bis zur Waagerechten ausführen. Des Weiteren solltest du die Handinnenseite richtig nach vorne drücken, bis es spannt. Anschließend drückst du den Körper in die Ausgangsposition zurück. Hände wieder lockern, der Oberkörper bleibt dabei immer gerade. Um den Lendenwirbelsäulenbereich zu schonen empfiehlt es sich, eine Katze als Trainingsgürtel zur Stabilisierung zu verwenden. Wichtig: Die Fußspitzen zeigen immer in die gleiche Richtung wie die Kniegelenke.

Entspannung 2

Radieschen von unten

Entschleunigt um 30 %



Der Begriff Kerze bzw. Schulterstand (im Yoga Sarvangāsana) steht für eine Turnübung, bei der du, auf dem Rücken liegend, mit den Händen in die Hüfte gestemmt, die Beine senkrecht nach oben streckst. Wichtig dabei: Stell dir vor, du bist ein Radieschen!

Entspannung 3

Entspannungskuscheln ohne Partner

Entschleunigt um 70 %



Diese Übung wirst du im Stehen ausführen, die Arme legst du von vorne um deinen Körper. Dabei sind deine Arme so angewinkelt, dass du am Rücken die Finger der jeweils anderen Hand berührst. Die Schultern dürfen während der gesamten Übung nicht angespannt werden, sondern müssen hängen bleiben. Sobald der Kuschelleffekt einsetzt, diesen für 3 Sekunden halten, etw as nachlassen und dann nochmals den Kuschelreiz herbeiführen.

Schau dir deinen persönlichen Entspannungsplaner mal genauer an. Du kannst aus einer ausgewogenen Mischung von Entschleunigungsübungen im **Entspannungskatalog auswählen** und dir pro Tag drei Übungen zu einem Entspannungsplan zusammenstellen. Allein vom Anschauen wird man schon ganz ruhig und gelassen. Natürlich kannst du die Übungen per **Drag & Drop** aus dem Katalog holen und – schwuppdiewupp – in den Plan einfügen. Pass nur auf, dass die Entspannung nicht über 100 % geht und du gar nicht mehr in die Gänge kommst!

MVC und ???

Wie schon gesagt: Wir werden uns mit webbasierten Anwendungen befassen. Schon lange hat sich ein Architekturmuster durchgesetzt, das als **Model View Controller (MVC)** bezeichnet wird. Das begegnet dir auch bei SAPUI5 und dem ABAP RESTful Application Programming Model. Du kannst dir unter einem Architekturmuster (für kleinere, wohlumgrenzte Aufgabenstellungen wirst du auch den Begriff **Software Design Pattern** finden) einen **Bauplan** oder ein **Strickmuster** für Softwarearchitekten und Softwareentwickler vorstellen.



[Zettel]

Bei Model View Controller handelt es sich um ein Muster, das die Benutzeroberfläche (View) von der Programmsteuerung (Controller) und dem zugrunde liegenden Datenmodell (Model) trennt. Das Muster stammt aus dem Jahr 1979 und geht auf den norwegischen Softwareentwickler Trygve Reenskaug zurück.

MVC kannte ich bislang nur als die Initialen von Mary Virginia Carey.

Und bei ihr heißt die Benutzeroberfläche Justus, die Programmsteuerung Peter und das Datenmodell

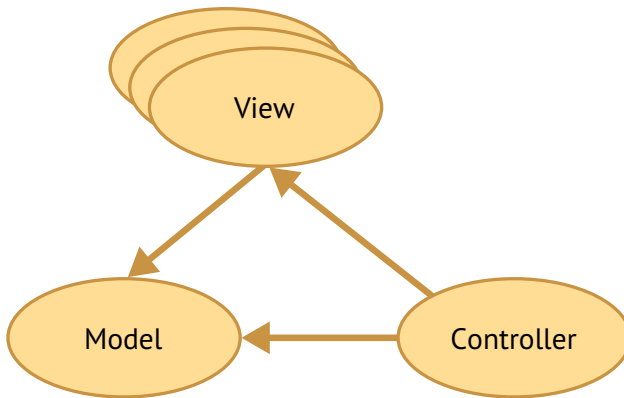
Bob. Oder so.

Die Idee beim Model-View-Controller ist es, **Bausteine** («Spezialisten») für unterschiedliche Aufgaben zu entwickeln. Die Spezialisten lassen sich mehrfach wiederverwenden, um Änderungen an Programmen zu vereinfachen (weil man zum Beispiel nur einen Spezialisten gegen den anderen austauschen muss) und die Arbeitsteilung bei der Entwicklung zu unterstützen.

Sag ich doch: „Wir übernehmen jeden Fall“ – Spezialisten eben!



Ein Spezialist kümmert sich um die Visualisierung der Daten, darum wird er als **View** bezeichnet. Der nächste Spezialist kümmert sich um die Geschäftsdaten und ihre Logik und wird **Model** genannt. Der dritte Spezialist kümmert sich darum, dass View und Model zusammenarbeiten, ein typischer **Controller**. Der Aufbau und das Zusammenspiel im MVC lassen sich sehr einfach veranschaulichen.



Das Model repräsentiert die **persistente Datenschicht**, es ist der Lieferant für den Controller und die Views. Die Views sind die Schnittstelle zum Anwender, also die Mensch-Maschine-Schnittstelle. Sie stellen die Daten auf dem Bildschirm dar und übernehmen Eingaben und sonstige Aktionen. Der Controller steuert das Programm – das heißt, er verarbeitet und reagiert auf Eingaben, indem er mit dem Model kommuniziert und entscheidet, welcher View entsprechend anzuzeigen ist. Jedoch enthält der Controller keine Anwendungs- bzw. Geschäftslogik. Das ist wiederum die Aufgabe des Models.

Mit der Aufteilung der Architektur in Bausteine ist zum Beispiel eine **Wiederverwendung** des Models möglich, wohingegen die Views und der Controller neu zu entwickeln sind. Denk doch mal an die Dynpro-Entwicklung, bei der du ja auch auf Daten zugreifst. Dort könntest du wunderbar das Model wieder einsetzen, das Dynpro (als Kombination von View und Controller) musst – darfst – du jedoch neu entwickeln.



[Belohnung/Lösung]

Ein kleines Gedankenexperiment: Wenn du das Speisekartenprogramm aus Kapitel 13 noch einmal im Geist Revue passieren lässt: Wie würdest du ihn nach MVC-Gesichtspunkten aufteilen? Als Belohnung winkt eine zusätzliche Gehirnwindung für die Anwendungsarchitektur.



Doug McIlroy, Architekt im Geiste

Bei der Entwicklung von Web Dynpro hat das Thema **komponentenbasierte Entwicklung** eine große Rolle gespielt. Das hat sich M. Douglas McIlroy – ein US-amerikanischer Mathematiker, Ingenieur und Entwickler – vermutlich nicht gedacht, als er 1968 einen Vortrag zum Thema »Mass produced software components« gehalten hat. Mit diesem hat er das Thema der komponentenbasierten Softwareentwicklung letztlich begründet und gilt als Pionier des **Component-Based Software Engineerings** und **Software Product Line Engineerings**.



[Notiz]

Auf <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/N1968/index.html> sieht man übrigens ein Foto von Doug McIlroy mit einer besonderen Komponente in der Hand. Prost! Und man lernt noch andere großartige Dinge über ihn, zum Beispiel: »Doug McIlroy eats serial ports for breakfast.«



Die Idee der komponentenbasierten Entwicklung ist es, Anwendungen in **wiederverwendbare Bausteine** zu unterteilen, die sich gegenseitig Services anbieten.

*Komponenten, Bausteine,
Services – Roland,
es wird wohl wieder Zeit für eines
deiner besüchtigten Beispiele ...*



Sehr gerne! Hast du ein Handy dabei?

Schon, aber das ... äh ... also das ist jetzt nicht besonders neu und so.



Perfekt, das macht es noch einfacher!

[Notiz]

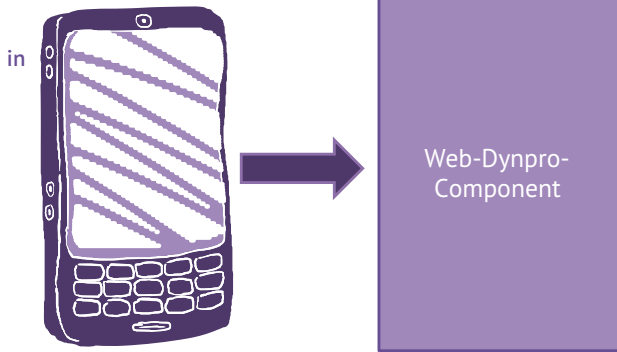
Im Folgenden machen wir einen Theorie-Crashkurs, damit wir uns im Anschluss schnell die Finger dreckig machen können. Zur komponentenbasierten Entwicklung an sich kann ich dir das »Handbuch moderner Softwarearchitektur« (O'Reilly, 2026) von Mark Richards und Neil Fords empfehlen. Es behandelt die sich aufdrängenden Fragen, wenn du dich von der objektorientierten zur komponentenbasierten Entwicklung bewegst.



Stellen wir uns ganz dumm und sagen: Dein Handy ist eine **Komponente**. Sie bietet dir Services an, wie zum Beispiel das Display (Etwas-auf-dem-Bildschirm-sehen-Service) oder die Tastatur (Etwas-eintippen-Service). Wir präzisieren das gleich noch ein wenig.

*Okay, jetzt hat es geklickt:
Und die Kamera meines
Handys ist der Etwas-
fotografieren-Service.*

Ganz genau. Und da du ja schon weißt, dass ich einfache schematische Zeichnungen liebe, werde ich dir jetzt die **Web-Dynpro-Component** und ihre Elemente zeigen. Mithilfe deines Handys. Also nicht erschrecken, es bleibt **ganz einfach**.



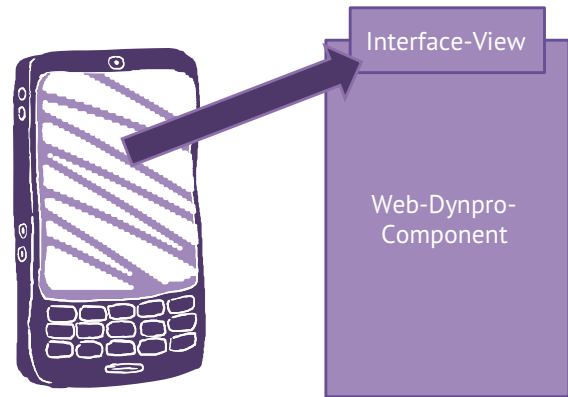
Aus deinem Handy wird zunächst eine **Web-Dynpro-Component**. Das ist auch der Name, der im SAP-System bzw. in Transaktion SE80 verwendet wird.

Stimmt, für diese Scharbilder benötigt man eine ganz besondere Form von Zuneigung.

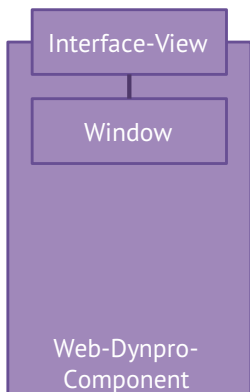
Aus deinem Handy-Display mache ich einen (Interface-)View.

Das ist die **Schnittstelle zu den Usern**, hier kannst du ihnen in deiner Web-Dynpro-Anwendung die gewünschten Daten anzeigen. Der Interface-View ist sozusagen der »visuelle Service« einer Web-Dynpro-Component.

Kann da auch mehr als ein Interface-View sein?

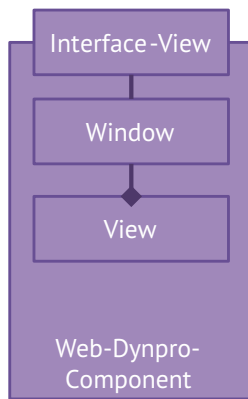


Gute Frage. Sicher, so viele Interface-Views, wie du benötigst bzw. deine Web-Dynpro-Component anbieten soll. Wenn du zum Beispiel eine **Uhr-Component** entwickelst, dann reicht dir ein Interface-View, eben deine Uhr. Aber wenn du die Uhr **digital** und **analog** darstellen möchtest, dann sind das schon zwei Interface-Views.



Die Krux dabei ist, dass du – und das ist jetzt ziemlich eigenartig – für einen Interface-View ein **Window definieren** musst. Ein Interface-View ist also – rein technisch – ein Window.

Achtung, Gehirnknotengefahr: Und um das Window mit Eingabefeldern, Buttons und dem ganzen Pipapo auszustaffieren, benötigst du **einen View** der Web-Dynpro-Component (oder mehrere)!



In ein Window können also Views eingebettet werden. So wird die Oberfläche deiner Anwendung gestaltet. Das ist letztlich wie eine Collage: Das Papier, auf das du deine Bilder klebst, ist das Window. Und die Bilder, die du arrangierst, das sind deine Views. Du kannst beliebig viele Views anlegen, und in einem Window können beliebig viele Views sein.

[Belohnung]

Weil du bis jetzt so tapfer durchgehalten hast, darfst du nun mit Schere und Kleber eine Collage basteln. Wie wäre es mit einer Tortencollage?

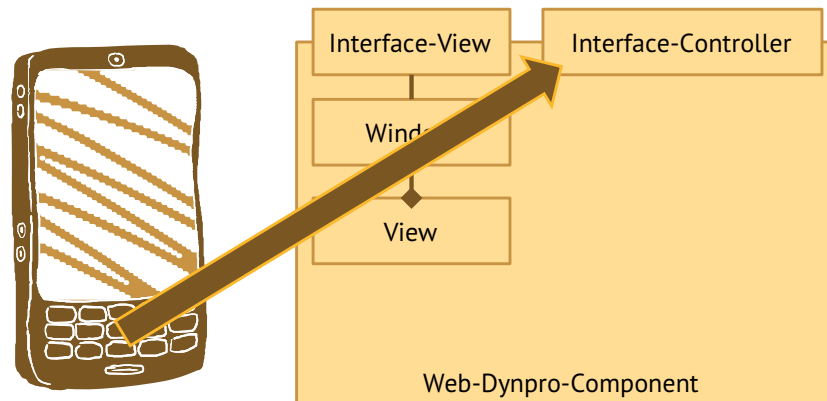


Keine Ahnung, was es meint.
Ich dachte spontan eher
an ein Erpresserschreiben.



Funktionale Schnittstelle

Kommen wir jetzt zur Tastatur deines Handys. Wenn das Display der »visuelle Service« einer Web-Dynpro-Component ist, könnte man die Tastatur als »funktionalen Service« bezeichnen. Oder auch als funktionale Schnittstelle. In einer Web-Dynpro-Anwendung spricht man vom **Interface-Controller**.



Im Interface-Controller steckt Funktionalität, das heißt alles, was benötigt wird, um mit der Component »von außen« zu kommunizieren. Zu diesem Zweck stellt der Interface-Controller Methoden, Ereignisse und Daten zur Verfügung, alles objektorientiert im SAP-System angelegt.

Objektorientierung? Darin habe ich schon das Seepferdchen und das Freischwimmer-Abzeichen. Der Interface-Controller wird wohl eine Klasse sein.

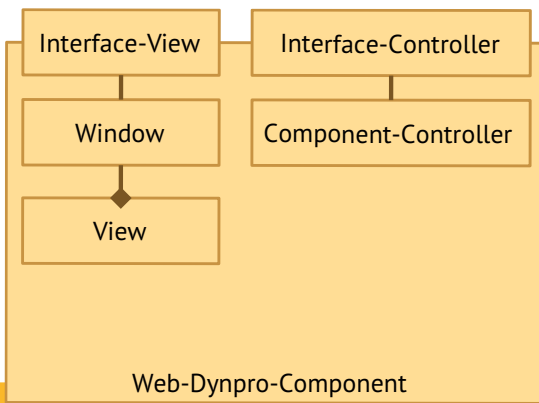


[Notieren/Üben]

Wenn du noch einmal ein kleines Schwimmtraining in Objektorientierung benötigst, schau dir die Kapitel 8 bis 10 an!

Wo kann ich die Implementierung der Methoden finden? Im Interface-Controller selbst?

Leider nein, die Implementierung liegt im nächsten Element, dem **Component-Controller**.



Der Component-Controller beinhaltet die ganzen Methoden, Ereignisse und Daten – das wird auch **Context** genannt. Der Context ist ziemlich wichtig. Und einen Context gibt es auch für den View und das Window. Dazu später mehr.

Wie viele von den Component-Controllern gibt es denn?

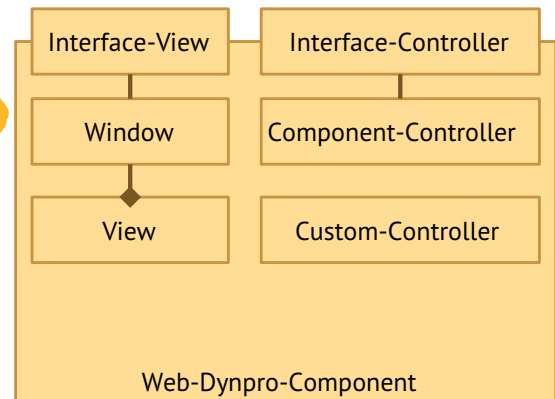
Das ist jetzt gemein, weil bis jetzt immer mehrere Exemplare von Windows und Views möglich waren. Aber vom Component-Controller **kann es nur einen geben**. Totales Highlander-Prinzip!

Ein letztes Element, von dem es mehrere geben kann, ist der **Custom-Controller**. Den kannst du verwenden, falls der Component-Controller zu unübersichtlich wird.

Über den Custom-Controller kannst du die Web-Dynpro-Component sozusagen **intern strukturieren**. Das geschieht zwar nicht so häufig, aber aus Gründen der Vollständigkeit zeichne ich ihn noch ein.

[Einfache Aufgabe]

Konzentriere dich auf das Bild und versuche, dir die gelben Kästchen in Blau und Rosa vorzustellen. Lies aber nicht den Inhalt der Kästchen!



Du erzählst mir jetzt aber nicht, dass es männliche und weibliche Web-Dynpro-Components gibt, oder?

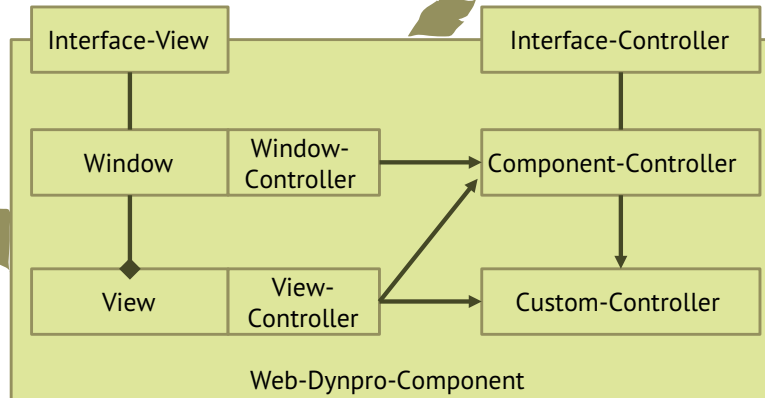
Mach einfach! Und danach versuchst du, mit dem linken Auge auf die Kästchen »Window« und »View« zu schauen, und mit dem rechten Auge auf »Component-Controller« und »Custom-Controller«. Was bemerkst du?

Vor allem hoffe ich, dass du jetzt kein Foto mit deinem Handy von mir machst. Was sollte ich denn bemerken?

Eigentlich nichts, ich wollte nur sichergehen, dass du auch nichts bemerkst.

Jedem seinen Controller

Und wo wir schon dabei sind: Ich hab dich angeschwindelt! Und dafür möchte ich mich aufrichtig entschuldigen. Es gibt nämlich doch noch mehr Controller, den **Window-Controller** und den **View-Controller**. Damit hat es sich jetzt aber – ich schwöre.



Komplett sieht eine Web-Dynpro-Component also so aus. Zusätzlich zu den Window- und View-Controllern, die ebenfalls Methoden und Context-Definitionen besitzen, habe ich auch die **Kommunikation** zwischen den Controllern eingezeichnet.

Wozu das alles, höre ich dich fragen?

Nö, lies war ich jetzt mal ruhig.

Zum Beispiel kann so der View-Controller mit dem Component-Controller kommunizieren. Und wie du aus der Objektorientierung bereits weißt, gilt hier die Regel:

Kommunikation = Methodenaufruf!

Die Controller können auch Daten austauschen, dann spricht man von **Context-Mapping**, da der Austausch über den Context geschieht. Und damit die Daten vom View in die Datenablage des Views – also den Context – transportiert werden (und umgekehrt genauso), wird das sogenannte **Data-Binding** eingesetzt.

uff!



Falls das jetzt alles zu schnell war: Keine Sorge, wir werden das nun im Folgenden alles detailliert besprechen.

Aber einen habe ich noch: Damit du deine Component testen kannst, musst du eine **Web-Dynpro-Anwendung erzeugen** (dazu lies weiter auf Seite 61). Diese startet die Component und ruft einen Interface-View auf. Und diese Anwendung kannst du, nachdem du sie angelegt hast, über eine URL **im Webbrowser aufrufen**.

[Belohnung/Lösung]

Gönn dir doch mal eine kleine Entspannung, und hol dir ein **Crazy Radish** aus dem Kühlschrank.



Alles beginnt mit einer Component

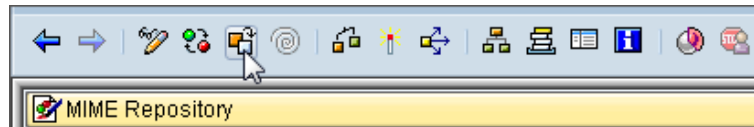


[Notiz]

Checkliste:

- Am SAP-System angemeldet? **Check!**
- Transaktion SE80 gestartet? **Check!**
- Den Button **Anderes Objekt ...** (+ **F5**) lokalisiert? **No Check!**

Ach ja, das ist neu. Nachdem du den **Object Navigator** mit Transaktion **SE80** gestartet hast, findest du in der Druckastenleiste den Button **Anderes Objekt ...** (+ **F5**).

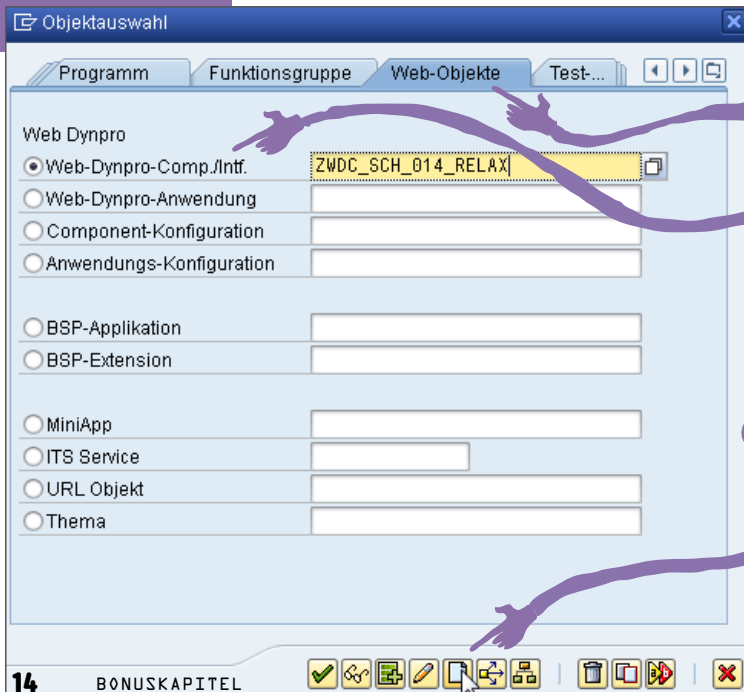


Und – Überraschung, Überraschung! – mittels dieses Buttons kannst du ein anderes Objekt anlegen. Nachdem du auf den Button **Anderes Objekt ...** geklickt hast, erscheint der Objektauswahl-Dialog. Hier kannst du beliebige Entwicklungsobjekte anlegen, ändern, löschen, kopieren und so weiter, und so weiter.

Du möchtest eine Web-Dynpro-Component anlegen, daher:

1. wählst du den Karteireiter **Web-Objekte** aus,
2. gibst du den Namen deiner Web-Dynpro-Component im Feld **Web-Dynpro-Comp./Intf.** ein (Vorschlag: **ZWDC_SCH_014_RELAX**),
3. bestätigst du dein Vorhaben und deine Eingabe über den **Anlegen**-Button, der ein leeres Blatt zeigt (oder über F5).

Der etwas andere Dialog zum Anlegen eines Entwicklungsobjektes. Ein Prachtexemplar.

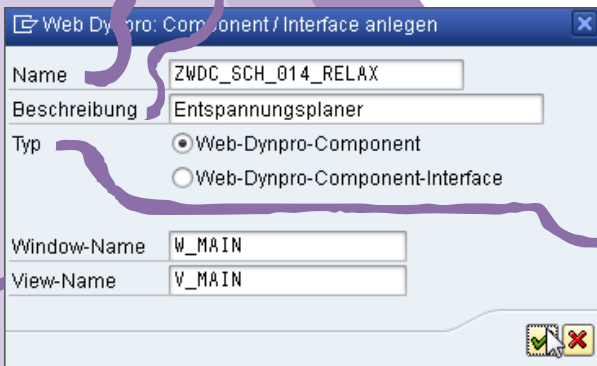


Nachdem du auf **Anlegen** geklickt hast, erscheint der nächste und somit ausschlaggebende Dialog: das Pop-up-Fenster **Web Dynpro: Component/Interface anlegen**, in dem du die grundlegenden Angaben zur Erstellung der Web-Dynpro-Component festlegst.

Im Eingabefeld **Name** könntest du den Namen deiner Component nochmals bearbeiten.

Das Eingabefeld **Beschreibung** dient zur Hinterlegung einer aussagekräftigen Kurzbeschreibung. Entspannungsplaner klingt immer noch gut, oder?

Ich warte vielleicht noch ein bisschen mit meinem endgültigen Urteil ...



Der Dialog zum Anlegen deiner ersten Web-Dynpro-Component. Auch sehr schön.

Der nächste Abschnitt mit der Bezeichnung **Typ** ist interessant. **Web-Dynpro-Component** sagt dir schon etwas, klar. Die zweite Möglichkeit nennt sich **Web-Dynpro-Component-Interface**. Du kannst dir darunter so etwas wie ein Interface in der Objektorientierung vorstellen.

[Notiz]

Wenn nicht, dann ab nach Kapitel 10!

Ein Interface beschreibt die Schnittstelle, also den Interface-View, Methoden, Ereignisse und Context-Daten. Diese werden aber nicht im Interface implementiert, sondern in einer Web-Dynpro-Component.

Déjà-vu? Falls nicht: Auch egal. Denn wir werden das Web-Dynpro-Component-Interface nicht besprechen, das ist zu fortgeschritten für unseren kleinen Entspannungsplaner. Deshalb wählst du hier auch **die erste Möglichkeit** aus: **Web-Dynpro-Component**.

Beim nächsten und übernächsten Eingabefeld, **Window-Name** und **View-Name**, sollte es bei dir läuten: Richtig, die wunderschöne Übersichtsgrafik einer Web-Dynpro-Component. Das Window ist das **Fenster zur Welt** (World Wide Web, im Webbrowser, du verstehst). Und der View dient der Layoutgestaltung. Die Einbettung des Views in das Window wird automatisch beim Anlegen erledigt – was für ein Service!



[Notiz]

Ich verwende gerne das Präfix **W_** für den Namen eines Windows und **V_** für den Namen eines Views. Wenn es dir gefällt, einfach übernehmen.

Hast du etwas gemerkt, Roland? Das war alles so gut nachvollziehbar, dass ich gar nichts fragen musste!

Und du hast es auch praktisch schon geschafft! Jetzt nur noch bestätigen und fertig ist deine **erste Web-Dynpro-Component**! Falls noch lästige Fragen vom System kommen (von wegen Paketzuführung und so), dann wie immer mit **\$TMP** antworten.

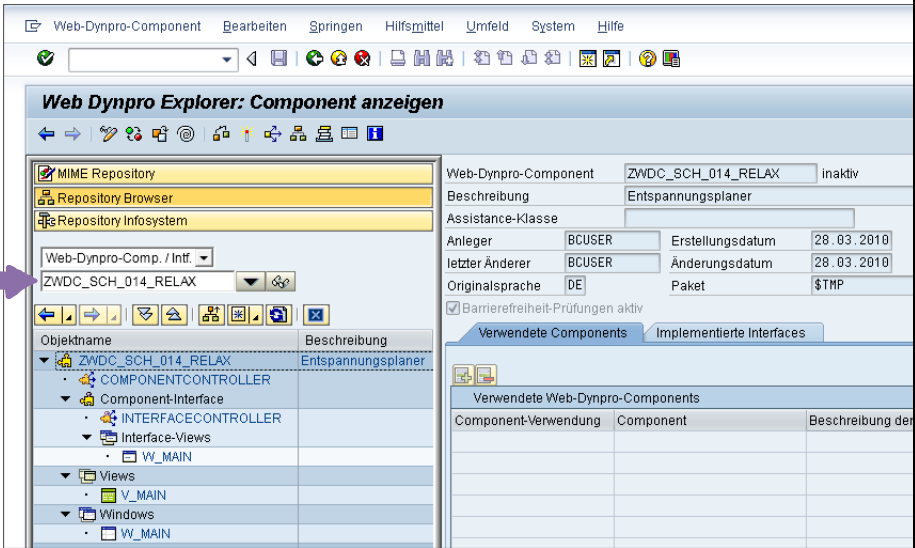
Ich weiß nicht, wie es dir geht, aber ich brauche eine Pause. Du auch? Dann habe ich dafür eine Aufgabe für dich.

Hätte ich eben mal nichts gesagt ...



[Einfache Aufgabe]

Öffne die Objektliste maximal, das heißt auch alle Knoten. Lass dir jedes Element der Web-Dynpro-Component anzeigen (natürlich eines nach dem anderen, bei der Component **ZWDC_SCH_014_RELAX** selbst beginnend). Gewinne dabei einen ersten Eindruck, ohne den Zwang, irgendetwas verstehen zu müssen.



Wenn du alle Knoten in der Objektliste öffnest, wirst du Altbekanntes entdecken: den **COMPONENTCONTROLLER**, von dem es nur einen gibt. Das Component-Interface, bestehend aus einem **INTERFACECONTROLLER** (»funktional«), den Interface-Views (»visuell«) und den Views und Windows.

Hach, es tut gut, alte Bekannte wiederzusehen.



Datenablage mit dem Context

Wie war die Entdeckungsreise? Alle Untiefen ausgelotet?

Ab jetzt bin ich auch wieder mit an Bord.

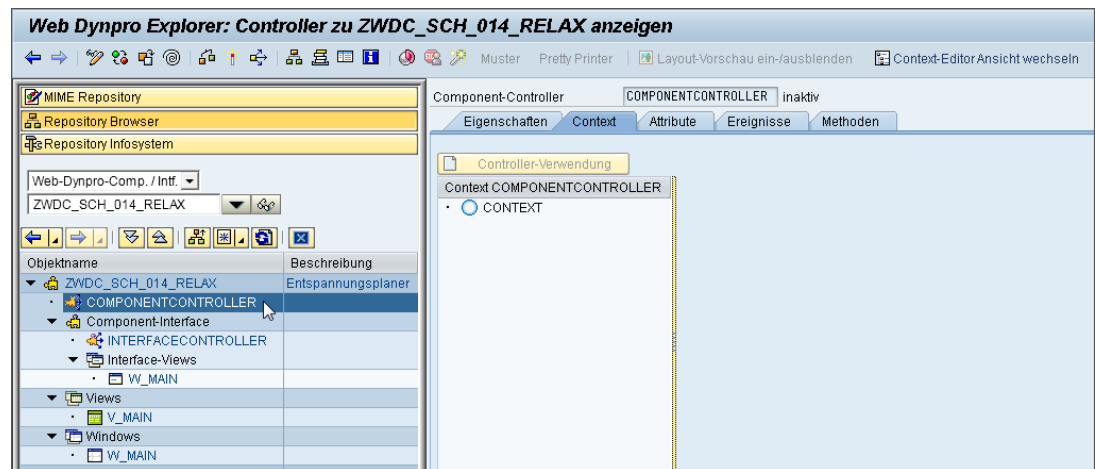
Die erste Untiefe, zu der ich dich begleiten werde, ist der **Context** eines Controllers.

Wie wir bereits festgestellt haben, ist der Context die **Datenablage** eines Controllers.

Natürlich nicht beliebiger Daten, sondern in den Context sollen all jene Daten, die für die Darstellung im View relevant sind. Für alle anderen Daten kannst du die **Attribute** des Controllers verwenden.

Von welchem Controller sprichst du denn gerade?

Der Context ist **in allen Controllern** vorhanden. Schau dir zum Beispiel den **COMPONENTCONTROLLER** an. Den kannst du ganz einfach durch einen Doppelklick auf den Eintrag **COMPONENTCONTROLLER** in der Objektliste erreichen.



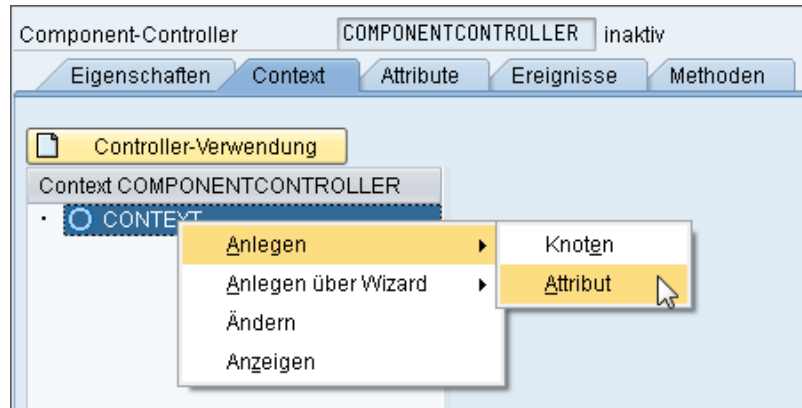
Die Darstellung des Controllers in Transaktion SE80 umfasst die Karteireiter **Eigenschaften, Context, Attribute, Ereignisse** und **Methoden**. Die Ereignisse sind dabei etwas Spezifisches für den **COMPONENTCONTROLLER**, die anderen Karteireiter findest du aber auch bei allen anderen Controllern.

Wie du siehst, besitzt der **COMPONENTCONTROLLER** einen Context, so wie alle anderen Controller auch. Und den entsprechenden **Context**-Karteireiter schauen wir uns nun genauer an. Du findest hier einen Kreis mit der Beschriftung **CONTEXT**. Das ist der sogenannte **Wurzelknoten** des Contextes. Mithilfe dieses Wurzelknotens und dessen Kontextmenü kannst du **Datenobjekte** anlegen, ändern, löschen und kopieren.

Ein Beispiel: Du möchtest den **Namen** des Entspannungssuchenden eingeben, für den der Entspannungsplan zusammengestellt wird, damit der Plan eine persönliche Note bekommt. Also benötigst du ein elementares Feld für den Namen.
In der Web-Dynpro-Nomenklatur nennt sich das **Attribut**.

[Zettel]

Mehr zu elementaren
Feldtypen steht in
Kapitel 5!



Das Anlegen eines Attributes dient zur Definition einer lokalen Datenablage im Controller. Um ein Attribut anzulegen, verwendest du das **Kontextmenü** des Wurzelknotens und dort den Pfad **Anlegen • Attribut**.



[Achtung/Vorsicht]

Kontext ist nicht gleich Context!! C und K machen hier einen wichtigen Unterschied, so ähnlich wie bei Coca und Koka.

Doch halt! Bevor du das Attribut tatsächlich anlegst, möchte ich dich auf ein kleines, aber entscheidendes Detail aufmerksam machen. Wenn du für einen Context direkt ein Attribut anlegst, wie eben beschrieben, ist dieses Attribut **nur für den aktuellen Controller** gültig! Fast so wie eine lokale Variable in einem Unterprogramm. Du kannst das Attribut und somit die Daten, die in diesem abgelegt sind, anderen Controllern **nicht zur Verfügung stellen** (in Web-Dynpro-Sprache: kein Context-Mapping durchführen)!

Wenn du jedoch das Attribut unter einem Knoten anlegst, kann dieser Knoten mit dem Attribut für andere Controller zugänglich gemacht werden (Web-Dynproisch: Context-Mapping durchführen).

Alles ziemlich trocken und theoretisch: **Zeit anzupacken** und den Entspannungsplan weiter zu implementieren. Fangen wir mit dem Namen der oder des Entspannungssuchenden an, und zwar mit einer **harten Randbedingung**: Ich hätte gerne die Verteilung des Namens zwischen dem **COMPONENTCONTROLLER** und dem Controller des Views **V_MAIN**.

Welche Herangehensweise wählst du? (Zutreffendes bitte einkringeln)

- ☛ A: Attribut im **COMPONENTCONTROLLER**-Context anlegen, eh klar!
- ☛ K: Knoten mit Attribut im **COMPONENTCONTROLLER**-Context anlegen und nach View-Controller **V_MAIN** mappen, natürlich!

Antwort: In diesem Fall musst du zuerst einen sogenannten **Knoten** definieren.

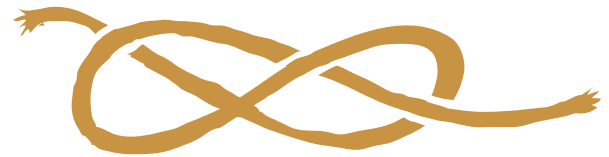
Dazu legst du erst einmal einen Knoten an:

1. Rufe das Kontextmenü des Context-Wurzelknotens auf.
2. Wähle den Menüeintrag **Anlegen • Knoten**.
3. Gib im sich daraufhin öffnenden Pop-up **Knoten anlegen** im Eingabefeld den Namen des Knotens ein, also **NAME**.
4. Bestätige deine Eingabe mit dem ersten grünen Häkchen (**Weiter**) oder der **Enter**-Taste.

The dialog box 'Knoten anlegen' has a title bar with a close button. The 'Knotenname' field is highlighted with a yellow border and contains the text 'NAME'. Below it are several dropdown menus and text fields. At the bottom, there are three green checkmarks and three buttons: 'Attribute aus Struktur hinzufügen', 'Weiterer Knoten', and a red 'X' button.

Die Anlage eines Knotens erfordert nicht viel. Für einfache Fälle reicht der Name des Knotens. Alle anderen Felder und Drop-down-Menüs kannst du fürs Erste unverändert lassen.

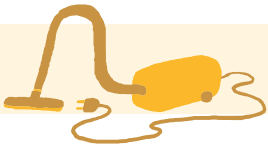
Das Ergebnis der Anlage lässt sich sehen, ein **neuer Knoten**. Du machst das fast wie Alexander der Große mit dem Gordischen Knoten, nur eben umgekehrt.



Man erkennt übrigens am **Ordner-Symbol** vor dem Namen des Knotens, dass es sich um einen Knoten handelt. Wie du gleich sehen wirst, sieht das Symbol für ein Attribut anders aus.

Noch schnell einen zweiten Schritt, bevor ich dir mehr zu den Eigenschaften des Knotens erkläre.

The screenshot shows the 'Component-Controller' window with the 'CONTEXT' tab selected. The 'CONTEXT' node is expanded, and a sub-node 'NAME' is visible with a folder icon next to it. The 'NAME' node is highlighted with a blue background.



[Einfache Aufgabe]

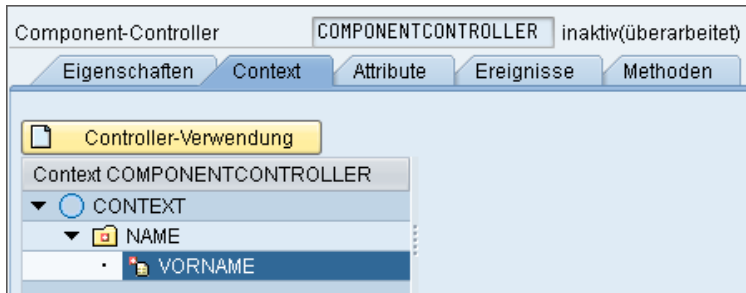
Lege das Attribut **VORNAME** unterhalb des Knotens **NAME** an. Verwende als Typ **STRING**.



Einfache Lösung: Das Kontextmenü des Knotens **NAME** aufrufen und dort dem Pfad **Anlegen • Attribut** folgen, um das Attribut anzulegen. Dort musst du im Eingabefeld **Attributname** den Wert **VORNAME** und im Eingabefeld **Typ** den Wert **STRING** eintragen und mit dem grünen Häkchen bestätigen. Voilà: Das Attribut **VORNAME** ist vom Typ **STRING**.

Das erinnert mich irgendwie an:
DATA: vorname TYPE string. Ob ich langsam verrückt werde?

Das Ergebnis ist im Context zu sehen: **ein Knoten mit einem Attribut.**



Im Gegensatz zum Attribut ohne Knoten ist nun das Attribut unter dem Knoten verteilbar, also für das Context-Mapping vorbereitet. Du könntest mich jetzt fragen »Warum um Himmels willen?« Und ich könnte antworten: Weil das Context-Mapping nur für Knoten, aber nicht für Attribute durchgeführt werden kann. Aber du fragst ja nicht.

Kleine Pause gefällig?



[Belohnung/Lösung]

Wie wäre es mit einem Weltrekordversuch im Knoten-Attribut-Anlegen (ein Knoten mit einem Attribut): Wie viele Knoten-Attribute **schaffst du pro Minute?** (Weltrekordbedingung: Eine Hand an der Maus, in der anderen Hand den Kaffeebecher halten!) Ich lege mal vor: 12!

[Zettel]

Falls du dich nach dem Weltrekordversuch wie ein **Alpen-Ampfer** oder ein **Chinesischer Rhabarber** fühlst, ist es dir nicht zu verdenken. Beide gehören zu den **Knöterich-Gewächsen**.



Wie ist euch zumute, ihr Context-Attribute?

Ich bin noch ganz außer Atem und total verknorret. Im Vertrauen: Ich habe gesehen, dass es Eigenschaften zum Knoten und zu den Attributen gibt.

Du bist ein Hellseher, gerade wollte ich das spannende Thema anschneiden.

Im Eingabefeld **Attributname** kannst du den Namen des Attributes festlegen. Egal, wie du den Namen eingibst, ob groß, ob klein oder gemischt, das System macht daraus **Großbuchstaben**. Der Name des Attributes kann zudem nicht länger als 20 Zeichen sein.

Beim Eingabefeld **Typisierung** denkst du am besten an das deklarative ABAP-Schlüsselwort **DATA**. Denn auch hier musst du für das anzulegende Datenobjekt einen Typ vergeben, das machst du mit **TYPE** oder **TYPE REF TO**. Also legst du für das Attribut fest, ob es eine statische (**type**) oder eine Datenreferenz/Objektreferenz (**type ref to**) sein soll.

*Bei mir steht aber nicht type, sondern
0 type, wie jetzt?*

Attributname	VORNAME
Typisierung	0 type
Typ	STRING
Read-only	nein
Default-Wert	
Wertehilfe-Modus	1 automatisch
Ermittelte Werteilfe	
Typ der Werteilfe	

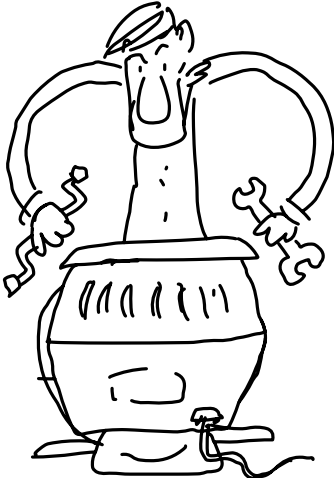
Die Eigenschaften eines Attributes beim Anlegen.

[Ablage]

Das hängt mit deinen **Einstellungen** zusammen: In der Drucktastenleiste findest du das Symbol für **lokales Layout anpassen** (Alt + F12). Und sobald du da drauf gedrückt hast, wählst du den Menüpunkt **Optionen ...** Wenn du das ausgewählt hast, erscheint das **Optionen**-Pop-up, und auf dem Karteireiter **Experte** entdeckst du, dass die Checkbox **Schlüssel in allen Drop-down-Listen anzeigen** gesetzt ist. Daran liegt's.

Das Eingabefeld **Typ** öffnet dir das Tor zu den Quellen aller ABAP-Typen. Eine kleine Auswahl: Falls du die Typisierung mit **type** gewählt hast, stehen dir elementare Typen aus dem ABAP Dictionary (Datenelemente), eingebaute Typen (**I**, **D**, **T**, **F**, **STRING** und **XSTRING**) und Komponenten von transparenten Tabellen und Strukturen (zum Beispiel die Namenskomponente **BNAME** aus der transparenten Tabelle **USR01 = USR01-BNAME**) zur Verfügung. Falls du als Typisierung **type ref to** angegeben hast, kannst du globale Klassen, Interfaces und elementare Typen verwenden.

Und was ist mit den anderen Attributen?
 Hier sind doch noch mehr
 Drop-down-Menüs, in denen ich etwas
 auswählen kann.



In Kurzform:

- Die Eigenschaft **Read-only** schützt das Attribut, sodass es nicht durch eine Eingabe des Anwenders geändert werden kann.
- Mithilfe der Eigenschaft **Default-Wert** kannst du dem Attribut einen Anfangswert zuweisen. Falls du nur für dich Entspannungspläne anlegen möchtest, wäre dein Name ein idealer Default-Wert. Mit diesem Wert wird dann das Attribut belegt, also so wie bei der **DATA**-Anweisung mit dem **VALUE**-Zusatz.
- Die Eingabe im Feld **Wertehilfe-Modus** steuert, ob und wenn ja, welche Wertehilfe dem Anwender bei einer Eingabe angeboten werden soll.

Übrigens: Falls du eine Eigenschaft gesetzt hast und diese **nachträglich ändern** möchtest, kannst du dir im Context die Eigenschaften zum Attribut anzeigen lassen und diese ändern.

Component-Controller COMPONENTCONTROLLER aktiv

Eigenschaften Context Attribute Ereignisse Methoden

Controller-Verwendung

Context COMPONENTCONTROLLER

CONTEXT

NAME

VORNAME

Eigenschaft	Wert
Attribut	
Attributname	VORNAME
Typisierung	type
Typ	STRING
Read-only	☒
Default-Wert	
Null-Wert	☐
Wertehilfe-Modus	automatisch
Ermittelte Wertehilfe	
Typ der Wertehilfe	
Formatierung	
Komprimierung der Eingabe	Standardwert

Das geht so: Zuerst öffnest du die Context-Hierarchie bis zum Attribut, und dann lässt du dir durch einen Mausklick auf den Namen des Attributes die Eigenschaften anzeigen.

[Zettel]

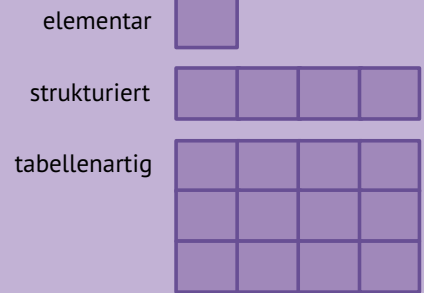
Kleine Fingerübung, damit es auch wirklich sitzt: Lege zum Knoten **NAME** ein neues Attribut **NACHNAME** vom Typ **STRING** an. Ich versuche mich derweil an einem neuen Weltrekord ...

Context-Knoten I

Jetzt kommt ein wirklich spannendes und zentrales Thema: die **Knotenattribution**.

Ich.
kann.
es.
kann.
erwarten!

Da wäre es sinnvoll, wenn du dir einen ruhigen Platz suchst, den Lotussitz des Programmiers einnimmst und nochmals das Thema der Datentypen und Datenobjekte vor deinem inneren Auge Revue passieren lässt.



Oh, großer Meister der Konzentration, das hat ja wunderbar funktioniert!



Wie konnte es
das denn jetzt sehen?
Spooky – Hoffentlich
hört es nicht an, ich,
was ich denke.

Keine Sorge, das kann ich nicht. Aber damit ich mir auch noch einmal alles gegenwärtig: **Elementare Felder** nehmen Daten für eine elementare Variable auf, **strukturierte Felder** nehmen Daten für mehrere Felder auf, und **tabellenartige Felder** nehmen Daten für mehrere Zeilen auf. Richtig, so war das. Jetzt zum Context-Knoten:

☛ Im Eingabefeld **Knotenname** legst du den Namen des Context-Knotens fest.
Achtung: Der Name darf pro Context **nur einmal vorkommen!**

☛ Kannst du dich noch an den Interface-Controller im Handy-Schaubild zur Web-Dynpro-Architektur erinnern? In diesem sind die **von außen** – also aus Sicht der Component – sichtbaren Methoden, Ereignisse und Daten der Component versammelt. Mithilfe der Eigenschaft **Interface-Knoten** kannst du festlegen, ob der Context-Knoten **von außen** sichtbar sein soll oder nicht. Ehrlich gesagt: Diese Eigenschaft verwende ich ganz, ganz, ganz selten, eigentlich nie. Zum Datenaustausch zwischen Components verwende ich Ereignisse oder Methoden.

Wie kommt's?

[Notiz]

Das war schnell gefragt, wird aber etwas Zeit beanspruchen, es zu erklären. Stell dir bitte einen **roten Rasentrimmer** vor, einen der eine aufladbare Batterie **fix eingebaut** hat. Und stell dir vor, dass du nun zumindest einen Monat lang, glücklich dahintrimmst. Und plötzlich trimmt der Trimmer nicht mehr. Du findest das Problem – es ist die Batterie! Wie sieht deine Lösung aus? Richtig: Wegwerfen, da die Batterie fest eingebaut ist. Nun stell dir bitte einen **grünen Rasentrimmer** vor, der eine **austauschbare Batterie** eingebaut hat. Und nun, trimmen, trimmen, trimmen und – Bumm! – Batterie kaputt. Was nun? Richtig, Batterie austauschen.

Wenn ich die **Trimmer-Architektur** des roten Trimmers beschreiben sollte, würde ich sagen, dass zwischen der **Trimmer-Komponente** und der **Batterie-Komponente** eine enge Bindung (**Tight Coupling**) besteht. Für den grünen Trimmer würde ich eine lose Kopplung (**Loose Coupling**) für die Beziehung zwischen Trimmer- und Batterie-Komponente verwenden.

Wenn eine Komponente Knoten mit den zugehörigen Attributen und Typen einer anderen Komponente direkt mittels Context-Mapping verwendet, führt das zu einer engen Kopplung. Wir wollen jedoch eine lose Kopplung erreichen, damit die Komponenten möglichst flexibel einsetzbar sind.

Darum, genau darum versuche ich, keine Context-Attribute in der Schnittstelle zu verwenden.

Hätte ich nur nicht gefragt ...
Übrigens, zum Thema Rasenmähen: Stehen zwei Schafe auf
der Weide. Sagt das eine: „Määäääääh!“ . Sagt das andere:
„Mäh doch selber!“

- Die Eigenschaft **Dictionary-Struktur** wiederum verwende ich fast immer. In diesem Eingabefeld kannst du den Namen eines Strukturtyps (Struktur, transparente Tabelle und View) aus dem Dictionary eingeben, um damit die Attribute des Knotens zu definieren.
- Mithilfe der Schaltfläche **Attribute aus Struktur hinzufügen** kannst du des Weiteren die Attribute aus dem Strukturtyp auswählen, die du gerne in dem Knoten hättest.

Alles, was für einen sauber gebundenen Web-Dynpro-Knoten notwendig ist.

Das mit dem Strukturtyp ist klar: Strukturtyp aus dem Dictionary auswählen, in das Eingabefeld Dictionary-Struktur eingeben und auf Attribute aus Struktur hinzufügen klicken. Aber wie kann ich dann die Attribute auswählen?

Ich beantworte deine Frage mit einer Aufgabe. Pädagogischer Schachzug. Das sogenannte Schwaiger-Manöver.

[Schwierige Aufgabe]

Lege die Struktur **ZSCH_14_TS_UI_EXERCISE** mit den Komponenten **IMAGE**, **NUMBER**, **NAME** und **FACTOR** an. Alle vom Datentyp **STRING**. Mithilfe dieser Attribute wirst du die Daten für eine Übung im Context speichern. Lege dann den Knoten **PLAN_DISPLAY** direkt unter dem Context-Wurzelknoten an, der drei Übungen speichern wird. Unterhalb des Knotens **PLAN_DISPLAY** lege die drei Knoten **EX1**, **EX2**, **EX3** an (das sind jetzt die drei Übungen), wobei bei jedem Knoten die Struktur **ZSCH_14_TS_UI_EXERCISE** zur Definition der Attribute verwendet wird. Übernimm alle Attribute aus der Struktur.

Wenn du hier angekommen bist, hast du entweder die Aufgabe gelöst – **Bravo!** – oder eben nicht. Falls nicht: Versuche es noch mal von vorne. Wenn du dann hier zum zweiten Mal angekommen bist, kannst du dir als Vergleich ansehen, wie ich diese Aufgabe gelöst habe.

Dictionary: Struktur pflegen

Struktur: ZSCH_14_TS_UI_EXERCISE aktiv

Kurzbeschreibung: Entspannungsplaner Übung

Eigenschaften **Komponenten** Eingabehilfe/-prüfung Währungs-/Mengenfelder

Komponententyp 1 / 4

Komponente	R.typ	Komponententyp	Datentyp	Länge	DezSt...	Kurzbeschreibung
IMAGE	☐		STRING			0 Bildname
NUMBER	☐		STRING			0 Übungsnummer
NAME	☐		STRING			0 Übungsname
FACTOR	☐		STRING			0 Entschleunigungsfaktor

Hier siehst du die **Struktur** für diese Aufgabe, bei der die Komponenten direkt typisiert werden. Du kannst gerne eigene Datenelemente für die Typisierung anlegen, damit für dich später während der Gestaltung der Web-Dynpro-Views die Labels, Eingabehilfen etc. **automatisch** produziert werden. Vergiss nicht, die Erweiterungskategorie einzustellen, um nicht immer die Erweiterungskategorie-Warnungen quittieren zu müssen.



[Notiz]

Ein kleiner Tipp: Entweder du siehst noch einmal in Kapitel 5 nach, oder du wählst den Menüpunkt **Zusätze • Erweiterungskategorie** und dann den Radiobutton **beliebig erweiterbar**.

[Erledigt!]

Sollten Warnungen erscheinen, dass mancher Name reserviert ist, **einfach ignorieren!**



Component-Controller COMPONENTCONTROLLER inaktiv(überarbeitet)

Eigenschaften **Context** Attribute Ereignisse Methoden

Controller-Verwendung

Context COMPONENTCONTROLLER

- CONTEXT
 - NAME
 - VORNAME
 - NACHNAME
 - PLAN_DISPLAY**

Ausgehend vom Knoten **PLAN_DISPLAY**, öffnest du den Anlagedialog per Kontextmenü.

Knoten anlegen

Knotenname:

Interface-Knoten:

Input-Element (ext.):

Dictionary-Struktur:

Kardinalität:

Selection:

Init. Lead-Selection:

Singleton:

Supply-Funktion:

☒ ☒ Attribute aus Struktur hinzufügen ☒ Weiterer Knoten ☐

Dort gibst du den Namen der Struktur **ZSCH_14_TS_UI_EXERCISE** ein und klickst auf den Button **Attribute aus Struktur hinzufügen**.

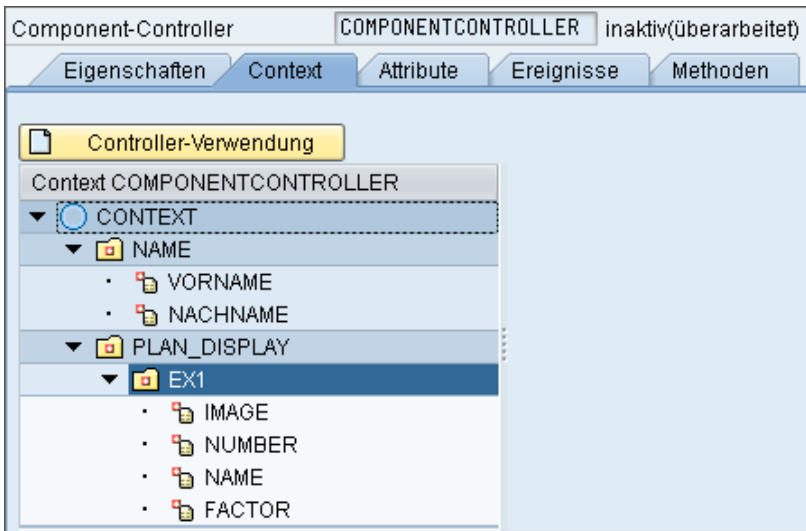
Die im nächsten Fenster erscheinenden Komponenten kannst du alle auswählen und mit dem grünen Häkchen bestätigen.

Komponenten der Struktur ZSCH_14_TS_UI_EXERCISE auswählen

1 / 4

Komponente	R.typ	Komponententyp	Datentyp	Länge	DezSt...	Kurzt...
<u>IMAGE</u>	☐		STRING	0	0 Bildn	
<u>NUMBER</u>	☐		STRING	0	0 Über	
<u>NAME</u>	☐		STRING	0	0 Über	
<u>FACTOR</u>	☐		STRING	0	0 Entsc	

☒ ☐



Der Weg der ersten Übung. So kannst du nach und nach alle Attribute anlegen.
Die anderen zwei Übungen überlasse ich deiner Kreativität.

Respekt, Schrödinger:

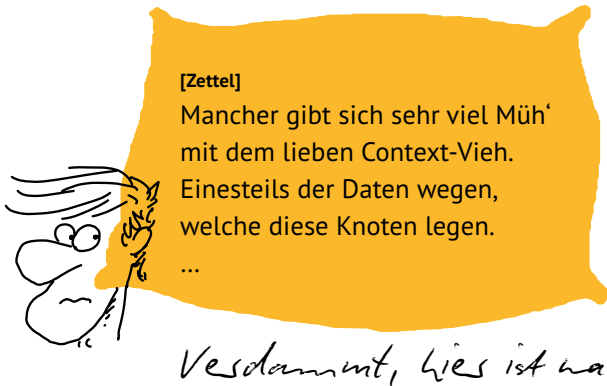
Du bist bereit für Teil II des
Context-Knoten-Epos!



Und wenn ich eine Struktur-
Komponente nicht ausgewählt
habe und sie nachträglich einfügen
möchte, dann wechsle ich in
die Eigenschaften des Knotens und
klicke auf die Eigenschaften
des Dictionary Struktur - und
schon kann ich wieder Struktur-
Komponenten hinzufügen.

Context-Knoten – Reloaded

Den ersten Teil zur Knotenerzeugung hast du hinter dich gebracht. Jetzt kommt der nächste Streich.



Zurück zum Thema. Context-Knoten – Reloaded. Bis jetzt hast du gesehen, wie du Knoten anlegen und dazu Attribute definieren kannst. Damit kannst du bereits elementare und strukturierte Variablen definieren. Sehr gut. Wie definiert man aber interne Tabellen für den Context? Ich sage nur eins: **Kardinalität!**

Component-Controller COMPONENTCONTROLLER inaktiv(überarbeitet)

Eigenschaften Context Attribute Ereignisse Methoden

Controller-Verwendung

Context COMPONENTCONTROLLER

CONTEXT

- NAME
- PLAN_DISPLAY
- PLAN_TABLE
 - IMAGE
 - NUMBER
 - NAME
 - FACTOR

Eigenschaft	Wert	Attribute übernehmen
Knoten		
Knotenname	PLAN_TABLE	
Interface-Knoten	☐	
Input-Element (Ext.)	☐	
Dictionary-Struktur	ZSCH_14_TS_UI_EXERCISE	
Kardinalität	1..1	
Selection	0..1	
Initialisierung Lead-Selection	1..1	
Singleton	0..n	
Supply-Funktion	1..n	

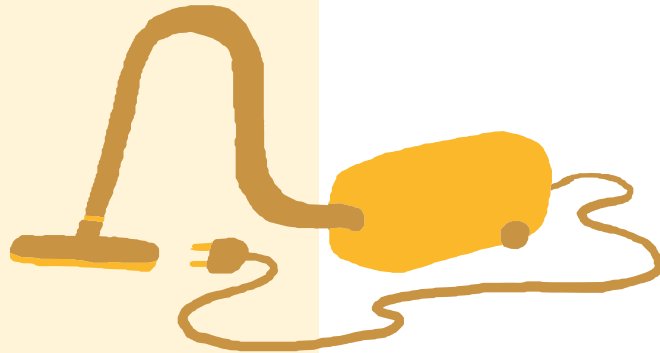
Kardinal(itäts)frage: Mithilfe der Eigenschaft **Kardinalität** legst du fest, wie viele sogenannte **Elemente** zu diesem Knoten existieren müssen und können. Du kannst dir unter einem Element einen Eintrag in einer Tabelle vorstellen. Dann wäre der Knoten die Tabelle. Eine Tabelle kann leer sein, sie kann aber auch eine beliebige Anzahl an Einträgen besitzen (naja, zumindest wenn der Speicherplatz vorhanden ist).



Falls du nun gerne hättest, dass zum Knoten genau ein Element existiert, dann stellst du die Kardinalität in ihrer **Ober- und Untergrenze** auf 1..1. Falls ein Element existieren kann oder auch nicht, dann verwendest du 0..1. Wenn du mehrere Einträge haben möchtest, verwendest du die Obergrenze n, also 0..n bzw. 1..n. Eine Sammlung von Elementen wird als **Collection** bezeichnet.

[Einfache Aufgabe]

Lege den neuen Knoten **PLAN_TABLE** an. Dort kannst du die Daten für die Entspannungsplan-Übungen sammeln. Nehme die passende Einstellung für die Kardinalität unter der Annahme vor, dass mehrere Elemente zum Knoten angelegt werden. Möglicherweise willst du die Übungen später als Tabelle darstellen?



*Ich weiß, ich weiß. Zuerst lege ich den Knoten **PLAN_TABLE** an, dann verwende ich die Struktur **ZSCH_14_TS_UI_EXERCISE** für die Definition des Attributs, und dann setze ich die Kardinalität des Knotens auf 1..n.*

Ganz genau, und noch eines: Die Eigenschaft **Initialisierung Lead-Selection** steuert, ob ein Element eines Context-Knotens automatisch ohne dein Zutun markiert ist. Die Checkbox zu dieser Eigenschaft ist im Allgemeinen gesetzt.

Nächstes Ziel auf unserer Route: Controller-Attribute!

Wo liegen die Daten, die der Controller benötigt, die den Usern aber nicht angezeigt werden sollen? **In den Attributen natürlich**, so wie es eben in der objektorientierten Programmierung ist. Zwei Attribute sind in Web Dynpro ABAP für jeden Controller vorhanden: **WD_CONTEXT** und **WD_THIS**.

- ☞ Das Attribut **WD_CONTEXT**, typisiert mit **IF_WD_CONTEXT_NODE**, ist die **Referenz auf den Context-Wurzelknoten**. Damit kannst du den Context durchsuchen und Daten lesen und setzen. Das Interface **IF_WD_CONTEXT_NODE** wird uns später noch mal begegnen.
- ☞ Das andere Attribut, **WD_THIS**, ist die Selbstreferenz **ME** in Web Dynpro. Das heißt, du zeigst mit **WD_THIS** auf den **Controller selbst**. Der Typ von **WD_THIS** ist interessant. Erstens, weil jeder Controller-Typ ein anderes Interface besitzt, und zweitens, weil sich das Interface während der Entwicklung verändert. Immer wenn du einen Knoten, ein Attribut, eine Methode anlegst, änderst oder löschst, wird das Interface geändert. **Während der Entwicklungszeit! Permanent!**

Wow! Das ist ja wirklich unfassbar, Roland!

Na hör mal: Erstens wird die Schnittstelle zum Controller **permanent angepasst**, zweitens musst du dich **darum nicht kümmern**, und drittens weißt du nun, wo du nachsehen kannst, wenn dich die **Schnittstelle des Controllers** interessiert mit all ihren Definitionen – speziell jenen, die den Context betreffen.

*Okay, das sind wirklich ein paar Argumente.
Ich bin auch begeistert.*



Geht doch.

Component-Controller		COMPONENTCONTROLLER		inaktiv
Eigenschaften	Context	Attribute	Ereignisse	Methoden
				
Attribut	Public	RefTo	Bezugstyp	Beschreibung
WD_CONTEXT	☐	☒	IF_WD_CONTEXT_NODE	Referenz auf den lokalen Controller-Context
WD_THIS	☐	☒	IF_COMPONENTCONTROLLER	Selbstreferenz auf das lokale Controller-Interface

Die Attribute nehmen alle Daten auf, die nicht für die Visualisierung gedacht sind. Vordefinierte Attribute erleichtern das Arbeiten. **WD_CONTEXT** liefert den Zugang zu den Context-Daten, und **WD_THIS** zeigt auf den Controller selbst. Du kannst natürlich die Attribute beliebig erweitern.

Da der Controller eine Klasse ist, ist es sicher ein Einfaches für dich, ein eigenes **Attribut anzulegen**. Ich hätte da eine Idee: Später werden wir den **Message Manager** verwenden, um Nachrichten an die User zu senden, wie zum Beispiel »Gute Wahl, schöne Übung«, nachdem die User eine Übung ausgewählt haben. Die Referenz werden wir im **COMPONENTCONTROLLER** ablegen. **Wie wär's, willst du das versuchen?**

Äh, was ist der Message Manager?

Das ist dein Kumpel, wenn es um das **Benachrichtigen** geht. Mithilfe des Message Managers erzeugst du **Nachrichten**, die dieser dann verwaltet und in einem **Nachrichtenbereich** ausgibt. Aber jetzt los:



[Zettel]

Lege die Interface-Referenz **GR_MM** vom Typ **IF_WD_MESSAGE_MANAGER** in den Attributen des **COMPONENTCONTROLLER** an.

Stopp, was hat eigentlich die Public-Spalte bei den Attributen zu bedeuten? Hat das etwas mit dem Interface-Controller zu tun?

Danke, das hätte ich doch beinahe übersehen. Im **COMPONENTCONTROLLER** kannst du Attribute definieren, die für alle anderen Controller sichtbar sind.

So wie eine **zentrale Datenablage**. Das **Public**-Attribut ist aber nur innerhalb der Component sichtbar und nicht nach außen. Somit hat es **nichts mit dem Interface-Controller** zu tun.

Ein **toller Zwischenstand**, den wir erreicht haben. Eine Referenzvariable für den Message Manager, auf die andere Controller zugreifen können.

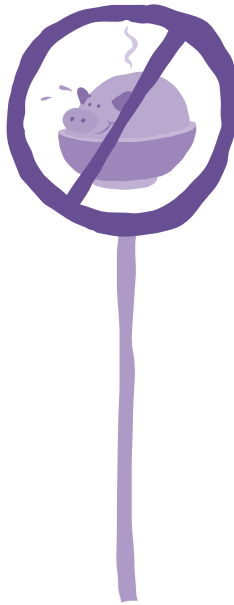


[Belohnung/Lösung]

Mach doch mal eine Aktivpause, und sieh dir den Boden ganz aus der Nähe an. Lege dich mit dem Bauch auf den Boden, Hände hinter den Kopf und Brust heben, so hoch du kannst. **Wie oft schaffst du das?**

Denk bitte bei der Übung an dich als Objekt, als Instanziierung. Du wirst diese Verinnerlichung für den nächsten Abschnitt gut gebrauchen können.

Context-Programmierung, sin carne



Eines kann ich dir sagen: Bei der nächsten Aufgabe wirst du dein geballtes objektorientiertes Wissen einsetzen können. Denn jetzt kommt die Programmierung des Contextes. Wobei: Wenn ich so recht darüber nachdenke, ist meine Formulierung nicht ganz korrekt. Neuer Versuch.

Eines kann ich dir sagen: Bei der nächsten Aufgabe wirst du dein geballtes objektorientiertes Wissen einsetzen können. **Denn jetzt kommt das Setzen und Lesen der Daten im Context!** Gefällt mir besser.

Als Beispiel möchte ich gerne einen Vorschlagswert für den Namen im Context-Attribut **VORNAME** setzen. Das könntest du zwar auch mit der Attributeigenschaft **Default-Wert** erledigen, jedoch bietet dir der implementierte Ansatz **mehr Flexibilität**. Der Name könnte zum Beispiel zuerst durch das Model von der Datenbank ermittelt und dann gesetzt werden. Da hilft kein Default-Wert, da erst während der Laufzeit der Wert ermittelt werden kann.

Verstanden, los geht's!

Da das Lesen bzw. Setzen ein Implementierungsaspekt ist, muss dieser **in einer Methode** implementiert werden. Methoden können in allen Controllern implementiert werden, wir werden diese Methode aber im **COMPONENTCONTROLLER** implementieren, da diese von allen anderen Controllern gerufen werden kann.

Preisfrage: Warum? (Tipp: Schau dir nochmals das Web-Dynpro-Architektur-Diagramm an, und achte speziell auf die Pfeile!)

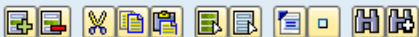
*Ich möchte lösen: Die anderen Controller haben eine Referenz auf den **COMPONENTCONTROLLER** in ihren Attributen - **WD_COMP_CONTROLLER**.*



Völlig richtig! Und hier ist dein Preis: Es geht sofort weiter!

Um eine Methode in einem Controller zu entwickeln, wechselst du auf den Karteireiter **Methoden** in der Controller-Sicht.



Component-Controller		COMPONENTCONTROLLER		inaktiv(überarbeitet)	
Eigenschaften	Context	Attribute	Ereignisse	Methoden	
					
Methode	Methoden-Typ	Interface	Beschreibung	Ereignis	
WDDOAPPLICATIONSTATECHANGE	0 Methode	☐	Behandlung bei Suspend und Resume einer		
WDDOBEFORENAVIGATION	0 Methode	☐	Fehlerbehandlung vor der Navigation durch di		
WDDOEXIT	0 Methode	☐	Cleanup-Methode des Controllers		
WDDOINIT	0 Methode	☐	Initialisierungs-Methode des Controllers		
WDDOPOSTPROCESSING	0 Methode	☐	Vorbereitung der Ausgabe		

Jeder Controller besitzt eine vordefinierte Menge an **Methoden**. Diese werden auch als **Hook-Methoden** bezeichnet und zu bestimmten Zeitpunkten aufgerufen. Die wichtigste Methode ist **wddoinit**, da sie in jedem Controller vorhanden ist und aufgerufen wird, sobald der **COMPONENTCONTROLLER** instanziiert wird. Eine weitere wichtige Methode ist **wddobeforenavigation**, die aufgerufen wird, sobald zwischen Views navigiert wird. **wddopostprocessing** wird als allerletzte Methode aufgerufen, bevor die Response zum Client zurückgesendet wird. Aber **wddoapplicationstatechange** und **wddoexit** werde ich dir nicht erklären, die sind geradezu unwichtig.

Damit du die Methode zum Setzen des Attributes **VORNAME** implementieren kannst, gehst du so wie im Class Builder (**Transaktion SE24**) vor.

[Achtung]

Lies den folgenden Text sehr sorgfältig.
Da hat sich ein Fehler eingeschlichen.

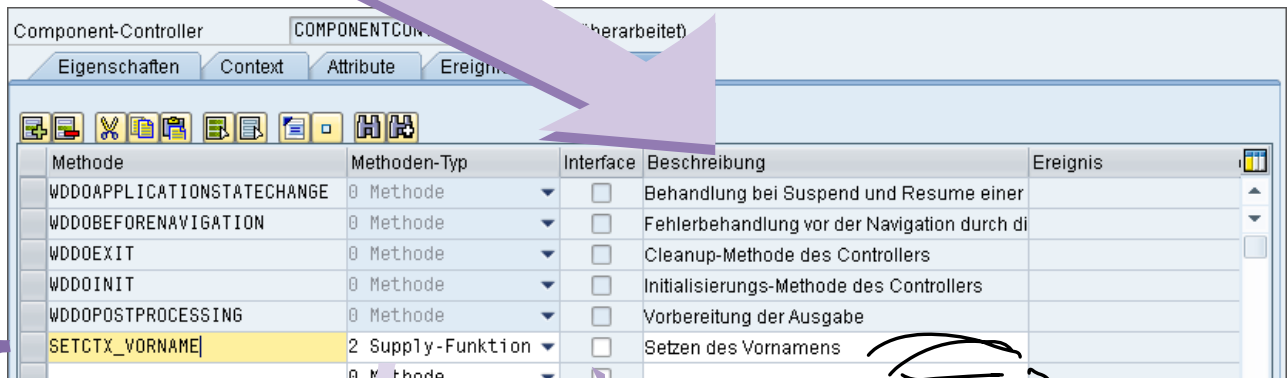
Trage in der Spalte **Methode** in einer leeren Zeile den Namen deiner Methode ein. Vorschlag: **setctx_vorname**.

Belasse den **Methoden-Typ** auf **Methode**. Das interessiert dich wahrscheinlich: Du kannst einen anderen Typ für deine Methode vergeben. Diese kann zum Beispiel ein **Ereignisbehandler** sein, so wie in der Objektorientierung, oder eine sogenannte **Supply-Funktion**, die unter gewissen Bedingungen, zum Beispiel »keine Elemente im Knoten«, automatisch von der Web-Dynpro-Runtime aufgerufen wird.

Die Checkbox in der nächsten Spalte – **Interface** – lässt du initial. Also nicht angekreuzt. Was könnte das dann bedeuten?

Das ist zum Sichtbarmachen
der Methode nach außen,
also für den Interface-Controller.
Aber viel wichtiger fände ich
zu wissen, ob der Fehler schon
vorkam ...

☛ In die Spalte **Beschreibung** kommt etwas Sinnvolles rein, das die Methode kurz beschreibt.



Methode	Methoden-Typ	Interface	Beschreibung	Ereignis
WDDOAPPLICATIONSTATECHANGE	0 Methode	☐	Behandlung bei Suspend und Resume einer	
WDDOBEFORENAVIGATION	0 Methode	☐	Fehlerbehandlung vor der Navigation durch di	
WDDOEXIT	0 Methode	☐	Cleanup-Methode des Controllers	
WDDOINIT	0 Methode	☐	Initialisierungs-Methode des Controllers	
WDDOPOSTPROCESSING	0 Methode	☐	Vorbereitung der Ausgabe	
SETCTX_VORNAME	2 Supply-Funktion	☐	Setzen des Vornamens	
	0 Methode	☐		

Deine erste Methode im **COMPONENTCONTROLLER**.

Hast du den Fehler entdeckt?

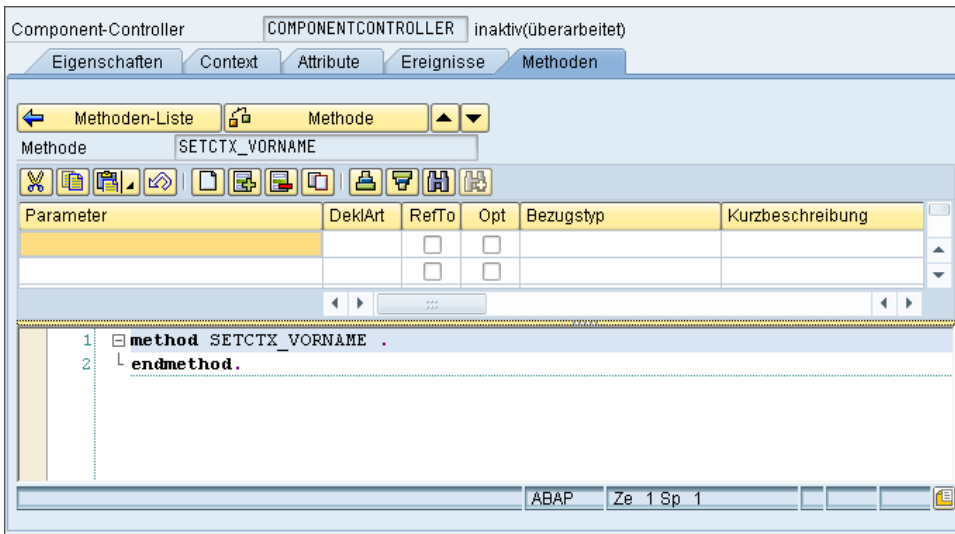
Der **Methoden-Typ** ist natürlich nicht **Supply-Funktion**, sondern **Methode**.



[Achtung]

Sobald der Typ der Methode fixiert ist, hast du **keine Möglichkeit mehr**, ihn zu ändern. Lösung: Methode löschen mit einem Klick auf den Button **Zeile löschen** und noch einmal neu anlegen, dieses Mal mit korrektem Typ.

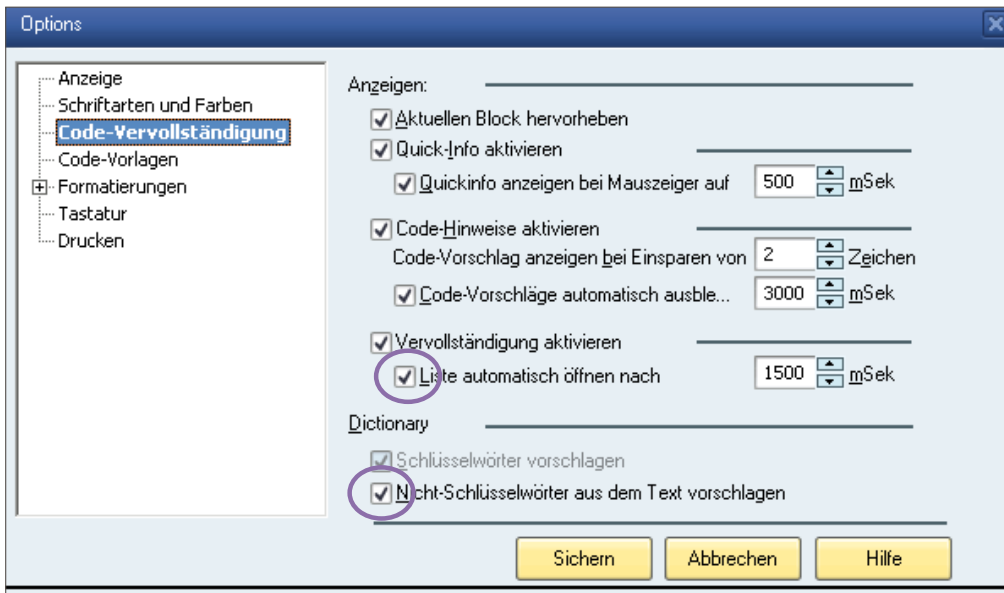
Wenn du alles mitgemacht und mitgedacht hast, kannst du nun in die Implementierung verzweigen. Am einfachsten **per Doppelklick** auf den Namen der Methode oder auch mit der Drucktaste **ABAP-Routine** oberhalb der Methoden.



Im Controller-Methoden-Editor fühlst du dich wahrscheinlich wie in deinem Wohnzimmer. Ganz oben steht der **Name der Methode**, dann kommen die Liste der **Formalparameter** und darunter der **Editor** zum Implementieren der Methode.

*Doof nur, dass wir gerade in der Werkstatt,
nicht im Wohnzimmer sind. Da hätte man doch bei
der Kapitelkonzeption ...*

Im Editor rechts unten, also ganz rechts unten, ist ein kleiner gelber Fleck. Das ist der Button ohne Tooltip und ohne Namen, über den du den **Einstellungsdialog für den Editor** öffnen kannst. Hier wollen wir jetzt Folgendes machen: Wir bringen dem System bei, uns bei der Programmierung durch **Codevervollständigung** zu unterstützen.



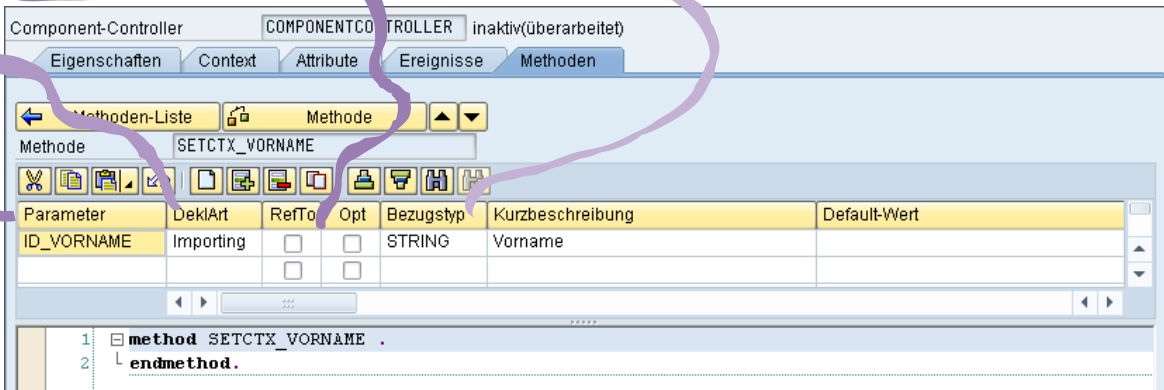
Genau das erreichst du in den Einstellungen für den Editor: Kreuze die Checkboxen **Liste automatisch öffnen nach** und **Nicht-Schlüsselwörter aus dem Text vorschlagen** an. Das bewirkt, dass du ab sofort die Codevervollständigung im Editor nutzen kannst, die dir bei der Entwicklung Texte vorschlagen wird. Diese kannst du mit der Tabulator-Taste dann einfach übernehmen.

Los geht's mit der **Implementierung**:

- ➔ Zuerst legst du den **Formalparameter ID_VORNAME** vom Typ **STRING** an, damit du später den Wert an diese Methode übergeben kannst, wenn du sie aufrufst.
 - ➔ In der Spalte **Parameter** legst du den Namen des Formalparameters fest.
 - ➔ In der Spalte **DeklArt** legst du fest, ob es sich um einen Importing-, Exporting-, Changing- oder Returning-Parameter handelt. In unserem Fall soll ein Wert an die Methode **setctx_vorname** übergeben werden, also kreuze die richtige Antwort an:
- 

[Notiz]*

 - ☐ Importing
 - ☐ Exporting
 - ☐ Changing
 - ☐ Returning
- ➔ Falls der Formalparameter ein **Referenztyp** sein soll, musst du die Checkbox in der Spalte **RefTo** ankreuzen. Und falls der Parameter optional ist, die Checkbox in der Spalte **Opt**.
 - ➔ In der Spalte **Bezugstyp** legst du den Typ des Formalparameters fest.



Parameter	DeklArt	RefTo	Opt	Bezugstyp	Kurzbeschreibung	Default-Wert
ID_VORNAME	Importing	☐	☐	STRING	Vorname	

```
1 method SETCTX_VORNAME .
2   endmethod.
```

Der Class Builder lässt grüßen: Du siehst die Liste der Parameter mit der Deklarationsart, der Art der Typisierung und dem Bezugstyp. **Empfehlenswert:** die Spalte **Kurzbeschreibung**. Damit du nach drei Monaten noch weißt, warum dein Parameter einen so eigenartigen Namen hat. Die Kombination aus optionalem Parameter und Default-Wert erzeugt schlanke Schnittstellen, da beim Aufruf der Methode der optionale Parameter nicht angegeben werden muss, wenn der Default-Wert passend ist.



Der Code-Wizard, dein Freund und Helfer

Nachdem die Schnittstelle definiert ist, geht es an die Implementierung.

Das könnten wir jetzt so richtig von der Pike auf machen. Oder: Wir nehmen eine Abkürzung. Alle dafür? Okay, dann die Abkürzung.

Die hat sogar einen Namen: **Web-Dynpro-Code-Wizard**. Wenn du Strg + F7 drückst, denk dir dazu bitte ein Geräusch, so wie **Tätärätätä**, nur viel lieblicher.

Immer, wenn du den Code-Wizard startest, hast du jetzt dieses **zuckersüße Tätärätätä** in deinem Kopf, auch das ist Zauberei.

Du findest den Zauberer in der Drucktastenleiste (Zauberstab) oder im Menü unter **Bearbeiten • Web-Dynpro-Code-Wizard**. Starte ihn doch mal.

Tätärätätä!



Mist, klappt.

Der Karteireiter **Context** bietet dir die Funktionen für den Context. Du willst den Wert des Attributes **VORNAME** setzen, daher wählst du den Radiobutton **Setzen**.

... damit der

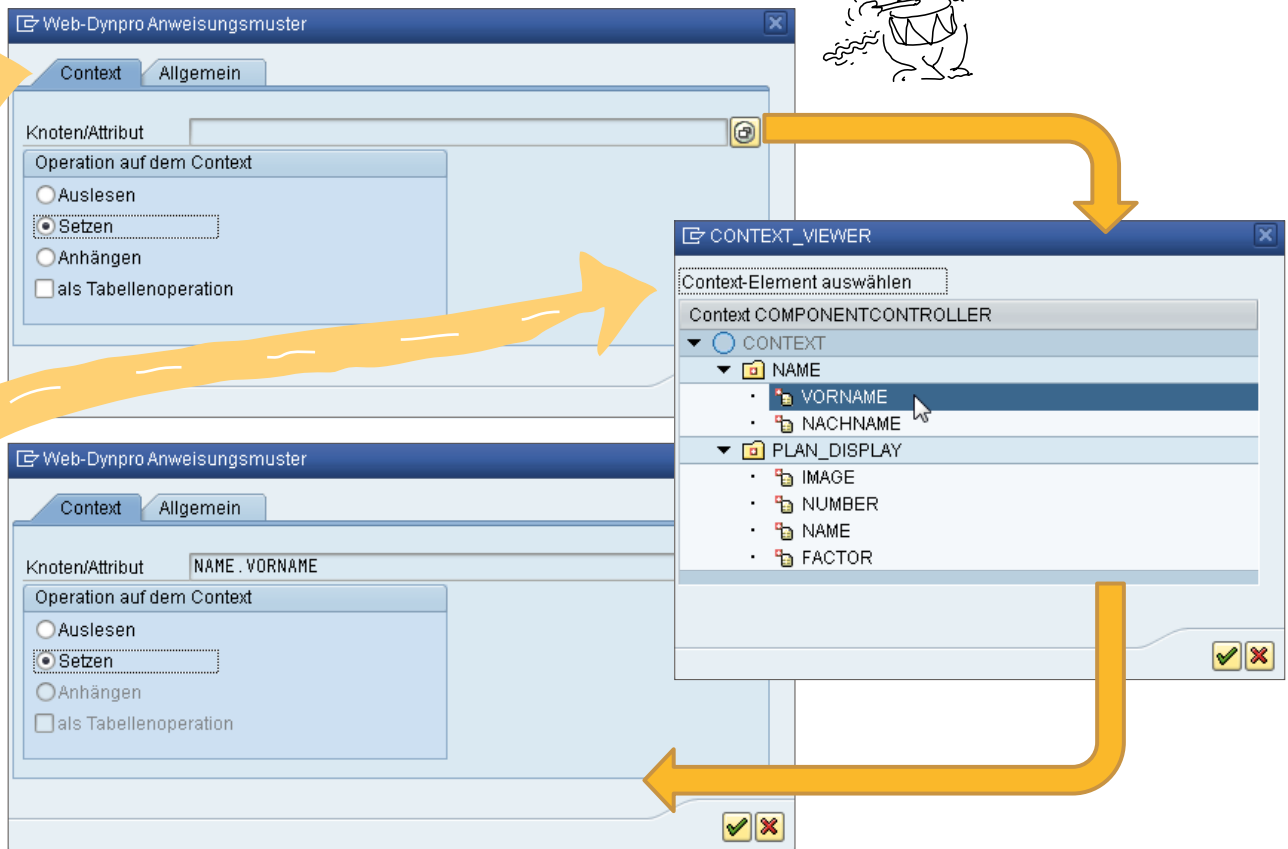
*Wert aus der Schnittstelle übernommen und im Context
abgelegt wird und dann angezeigt werden kann.
Puh, gerade noch mal die Klammer gekratzt.*

Der Knoten **NAME** besitzt die Kardinalität 1..1, daher ist es keine Tabelle, somit setzt du nicht die Checkbox **als Tabellenoperation**. Nach den vielfältigen Einstellungen drückst du die Wertheilfe-Taste des Feldes **Knoten/Attribut**, damit du das Attribut **VORNAME** als Ziel der Operation auswählen kannst. Damit ist für den Code-Wizard klar, **wohin** die Daten gestellt werden sollen.

Im folgenden Pop-up wählst du das **Ziel der Operation**, das Attribut **VORNAME**. Bestätigen nicht vergessen. Wie immer mit dem grünen Häkchen.

Nachdem du die Auswahl bestätigt hast, sind alle Einstellungen vorgenommen – die **Setzen**-Operation gesetzt und das Attribut **VORNAME** gewählt –, um den Code-Wizard seinen Job erledigen zu lassen.
Das grüne Häkchen sei mit dir. Also hau drauf! Was passiert?

Tätarätätä!



Der Code-Wizard war fleißig und hat eine Unmenge an Code für dich in der **Setter-Methode** **setctx_vorname** produziert. Da steckt alles drin, was Spaß macht und ...

***1** Das ist eine Interface-Referenz und zwar eine ziemlich wichtige. Es gibt nicht viele Interfaces, die du dir namentlich merken solltest, das hier ist aber eines davon:
IF_WD_CONTEXT_NODE. Diese Referenz stellt dir alle Methoden zur Verfügung, um mit einem Knoten zu arbeiten.

***2** Und da kommt schon die nächste Interface-Referenz, von der du träumen solltest.
Das Interface **IF_WD_CONTEXT_ELEMENT**, das alles zur Verfügung stellt, was zum Arbeiten mit Elementen, also den Datensätzen des Knotens wichtig ist.

***3** Bei der Definition eines Knotens wird zu jedem Knoten ein Strukturtyp im View-Controller angelegt. Der Name lautet **element_<Name des Knotens>**. Die Komponenten des Strukturtyps sind die Attribute des Knotens. Mit dieser Deklaration wird eine **Strukturvariable** identisch mit der **Knotendefinition** angelegt.

METHOD setctx_vorname .

DATA lo_nd_name TYPE REF TO if_wd_context_node. ***1**
DATA lo_el_name TYPE REF TO if_wd_context_element. ***2**
DATA ls_name TYPE wd_this->element_name. ***3**

```
DATA lv_vorname TYPE wd_this->element_name-vorname. *4
```

```
* navigate from <CONTEXT> to <NAME> via lead selection  
lo_nd_name = wd_context->get_child_node( name = wd_this->wdctx_name ). *5
```

```
* @TODO handle non existant child
```

```
* IF lo_nd_name IS INITIAL. *6
```

```
* ENDIF.
```

```
* get element via lead selection
```

```
lo_el_name = lo_nd_name->get_element( ). *7
```

```
* @TODO handle not set lead selection
```

```
IF lo_el_name IS INITIAL. *8
```

```
ENDIF.
```

```
* @TODO fill attribute
```

```
* lv_vorname = 1.
```

```
lv_vorname = id_vorname. *9
```

```
* set single attribute
```

```
lo_el_name->set_attribute(  
    name = `VORNAME`  
    value = lv_vorname ). *10
```

```
ENDMETHOD.
```

*7 Ist die **gültige Referenz** auf den Knoten vorhanden, kann gelesen werden. In unserem Fall lesen wir das Lead-Selection-Element.

*6 Was sollten wir tun, falls der Knoten nicht bekannt ist, also **keine gültige Referenz** vorhanden ist?

[Zettel]

Möglicherweise fragst du dich, warum die Kommentare in Englisch sind. Die werden vom Code-Wizard automatisch in Englisch produziert. Falls es für dich nicht passt, einfach ändern.

*Da hat dann jemand ziemlich gewusst und den Namen des Knotens geändert und im Code nicht angepasst. Ich wurde die Methode verlassen: **EXIT**.
Klingt auch so schön dramatisch.*

[Notieren/Üben]

Zum Lead-Selection-Element: Dem Knoten sind Elemente zugeordnet, wobei die Kardinalität des Namens Knotens 1..1 ist. Das wiederum bedeutet, genau ein Element und die Eigenschaft **Lead-Selection** sind für den Namens Knoten auch gesetzt. Und das heißt: Das eine Element zum Namens Knoten ist garantiert vorhanden und ausgewählt, und mit der Methode **get_element** bekommst du die Referenz auf das Element.

*4 Der Code-Wizard produziert eine **zusätzliche Hilfsvariable**, basierend auf der Auswahl des Attributs für die Operation.

*5 Jetzt geht's ans Eingemachte: Mithilfe des **Controller-Attributs WD_CONTEXT** und der Methode **get_child_node** aus dem Interface

IF_WD_CONTEXT_NODE wird die **Referenz** für den Knoten **NAME** ermittelt. Darüber kann mit dem Knoten kommuniziert werden.

Für das Lesen des Knotens wird eine Konstante verwendet, die automatisch bei der Anlage des Knotens definiert wurde.

Der Name der **Knotenkonstanten** lautet immer **wdctx_<Name des Knotens>**.

Übrigens könntest du den optionalen Importing-Parameter **index** der Methode **get_element** verwenden und damit auch Elemente lesen, die nicht als Lead-Selection markiert sind. Aber bei uns gibt es sowieso nur ein Element.

*8 Es kann natürlich sein, dass zu einem Knoten **keine Elemente** vorhanden sind. Und zwar genau dann, wenn die **Kardinalitätsuntergrenze gleich 0** ist. In diesem Fall wäre die Referenz **NULL**.

Wie war in unserem Beispiel die Einstellung?
Untergrenze = 1. **Damit sind wir aus dem Schneider**, da die Laufzeit automatisch ein Element anlegt. Diesen Code könntest du entfernen.

*9 Wir sind so weit. Die Hilfsvariable kann mit dem gewünschten **Wert versorgt** werden. Wir übernehmen den Wert aus der Schnittstelle der Methode.

*Wir sind so weit? Also du vielleicht.
Und der Code-Wizard auch.
Aber für mich könntest du vielleicht
noch einmal in Stichworten
erklären ...*

Okay, okay ... Ich war im Kopf schon in der Kaffeepause.
Was bisher geschah: Du hast die Referenz auf den Namensknoten ermittelt, danach die Referenz auf das einzige Element des Knotens. Der nächste Schritt ist das Setzen eines Wertes zum Attribut **VORNAME** des Elements, und dafür übernimmst du den Wert aus der Schnittstelle der Methode in die vom Code-Wizard vorgeschlagene Hilfsvariable **lv_vorname**.

*10 Nachdem wir die **Referenz auf das Element** ermittelt haben, können wir mit diesem sprechen.

*In welcher Sprache denn?
ABAP zählt nicht!*

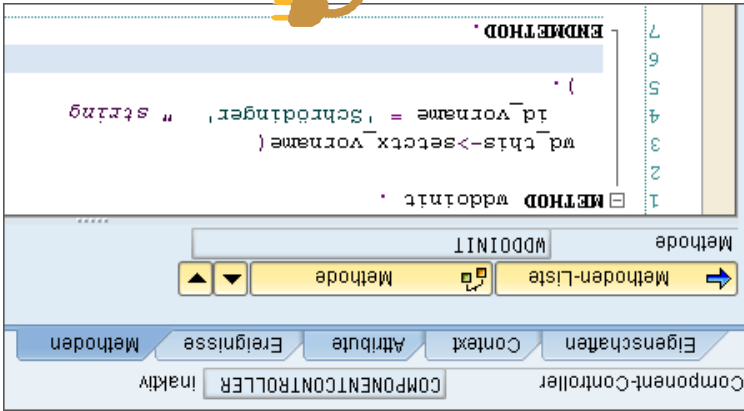
Wir können dem Element einen **neuen Wert** für eines seiner **Attribute** mitteilen, was mit der Methode **set_attribute** aus dem Interface **IF_WD_CONTEXT_ELEMENT** erledigt wird.

Geschafft. Und wenn wir schon beim Programmieren sind, so richtig mit Code und so, was bei Web Dynpro ja nicht so oft vorkommt, können wir doch auch den Aufruf der Methode **setctx_vorname** in die Hook-Methode **wddoinit** einbauen.



[Schwierige Aufgabe]

Rufe die Methode **setctx_vorname** in der Hook-Methode **wddoinit** des **COMPONENTCONTROLLER** auf!



Hey, nicht schummeln und einfach nur die Seite umdrehen! Du bist in die Methode **wddoinit** des **COMPONENTCONTROLLER** gewechselt und hast dort den Code-Wizard aufgerufen. Danach hast du den Karteteiler **Allgemein** gewählt und dort den Radiobutton für **Metho-**
denaufruf im aktuellen Controller gesetzt und mithilfe der Suchhilfe die Methode **setctx_vorname** aufgerufen.

Kleiner Tipp: Für die Lösung musst du dich **auf den Kopf stellen**! Das dient gleichzeitig der Entspannung und dem Teint.



Damit haben wir die **Context-Programmierung** durch, und du kannst jetzt die Web-Dynpro-Component aktivieren. Ganz einfach, so wie immer, verwendest du das Kontextmenü der Component in der Objektliste und wählst den Menüpunkt **Aktivieren**.

Und dann?

Dann bist du auf dem richtigen Weg. Du willst etwas sehen! Also werden wir im nächsten Abschnitt dafür sorgen, dass du etwas siehst.

Eine Frage hätte ich aber noch: Wo ist jetzt das weiße Kaninchen?

Das hatte ich zu Mittag.

Robaaaaaaard!



Window, View und der Rest

So langsam wird es Zeit, an der Visualisierung des Entspannungsplaners zu arbeiten.
Noch mal das Grundprinzip: Nach außen sichtbar ist der **Interface-View**, der eigentlich ein **Window** ist. Dieses wiederum bettet einen oder mehrere **Views** in sich ein.
Das ist schon das Prinzip.

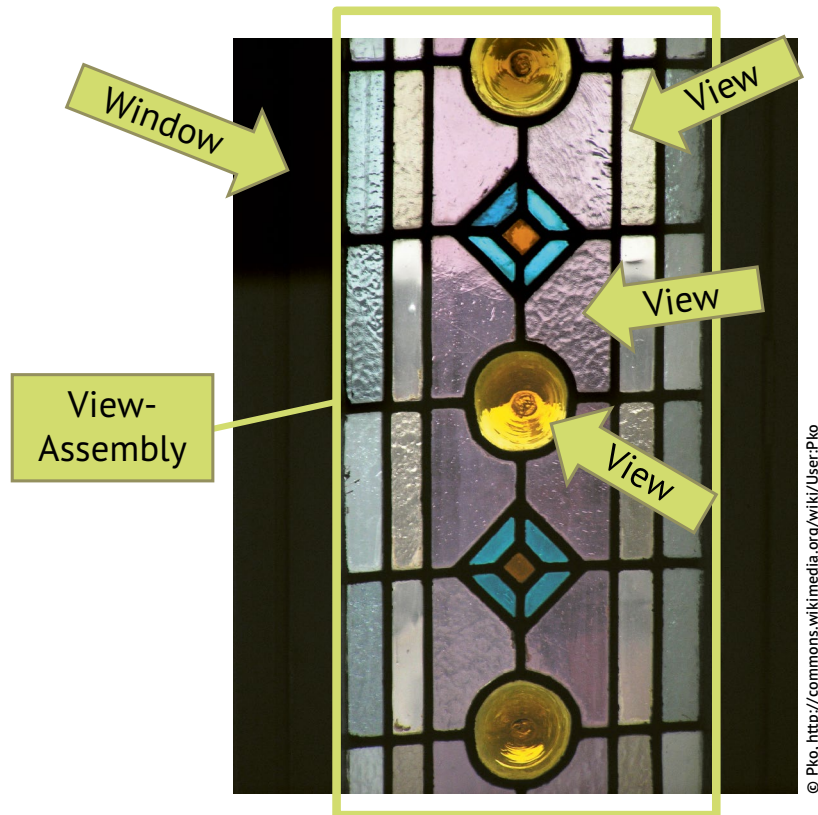
Und in verständlich?

Hmm, nehmen wir mal an, du wärst begeisterter Hinterglasmaler.

Und hoffen wir außerdem noch, dass jetzt ein gutes Beispiel kommt.

Die Glasscheibe, auf die du malst, ist der **View**.

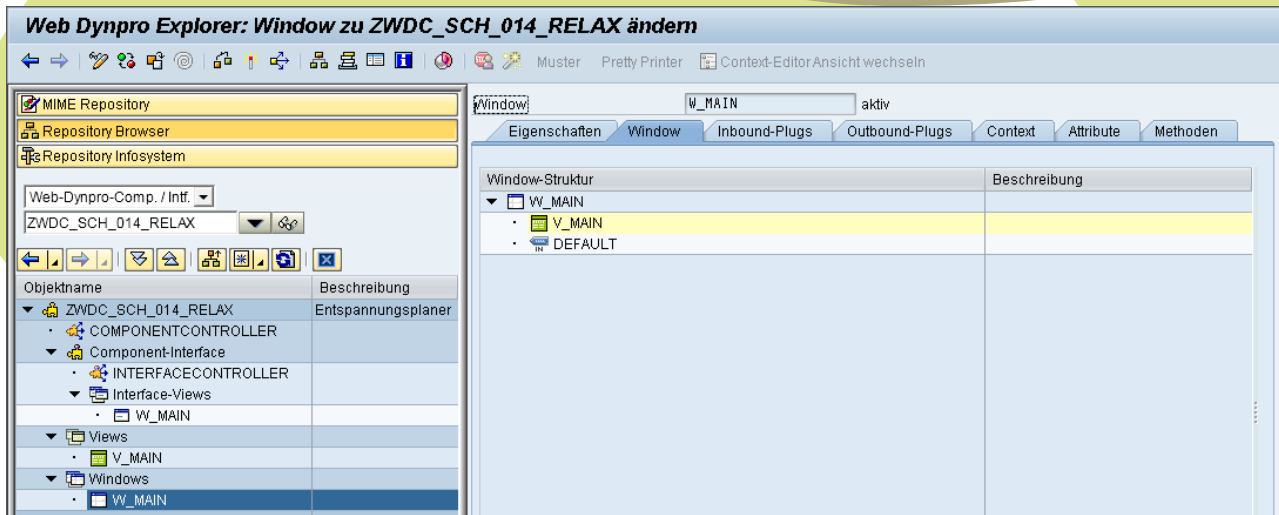
Und der wird ja in einen Rahmen gesteckt, das ist das **Window**. Also ein Interface-View. Falls du mehrere Scheiben in den Rahmen hineinsteckst (Butzenscheibe), würde das in Web-Dynproisch als **View-Assembly** bezeichnet.



Eine wunderschöne Butzenscheibe, die visuelle Schnittstelle von innen nach außen und von außen nach innen.

Wie du dich erinnerst, haben wir beim Anlegen der Component bereits **einen View und ein Window** angelegt. Du kannst natürlich so viele Windows und Views anlegen, wie du möchtest. Einfach das Kontextmenü auf der Component öffnen und dann den passenden Menüpfad auswählen: **Anlegen • View** bzw. **Anlegen • Window**. Die Zuordnung des Views zum Window wurde beim Anlegen ebenfalls automatisch durchgeführt. Das ist aber ziemlich einfach. Du musst nur in die Sicht des Windows und dort auf den Karteireiter **Window** wechseln.

Ordner Windows öffnen und Doppelklick auf das gewünschte Window, ganz klar.



Wenn du die Window-Struktur öffnest, siehst du das Window **W_MAIN** und den eingebetteten View **V_MAIN**. Und wenn du genau hinschaust, siehst du noch etwas: einen sogenannten **Inbound-Plug**, ganz in Blau gehalten. Wenn du noch mal deinen Blick über die Karteireiter schweifen lässt, wirst du feststellen, dass es einen Karteireiter für **Inbound-Plugs** und einen für **Outbound-Plugs** gibt.

*Reingeh-Stecker,
Rausgeh-Stecker,
wo ist das Kabel
dazwischen?*

Zwischen einem Outbound-Plug und einem Inbound-Plug kannst du einen **Navigationsübergang** definieren, damit sind Outbound- und Inbound-Plug miteinander verbunden. Die Plugs dienen dazu, eine Navigation, also einen Übergang, **von einem View zu einem anderen View** zu definieren. Das kann direkt zwischen den Views oder über das Window geschehen. Das sollte mal reichen für den Anfang.





„Hallo, Inbound-Plug?!“

Ach ja, und zum **DEFAULT**-Plug des Windows: Dieser Inbound-Plug kann beim Aufruf der Component über eine Web-Dynpro-Anwendung aufgerufen werden. Da kannst du zum Beispiel **Daten beim Aufruf mitgeben**.

Und was für Daten könnten das zum Beispiel sein?

Alles was du einem elementaren Feld zuweisen kannst. Das kann ein Datum sein oder die Nummer deines Strafzettels oder die Länge deines rechten Fußes.

[Zettel]

Eine Möglichkeit, um Daten an eine Web-Dynpro-Anwendung zu übergeben, sind die **URL-Parameter**. Dazu muss die URL der Anwendung erweitert werden:

&<name>=<wert>

Zum Beispiel:

&nummer=12345.

[Zettel]

Lege den Inbound-Plug

IN und den Outbound-

Plug **OUT** für den View

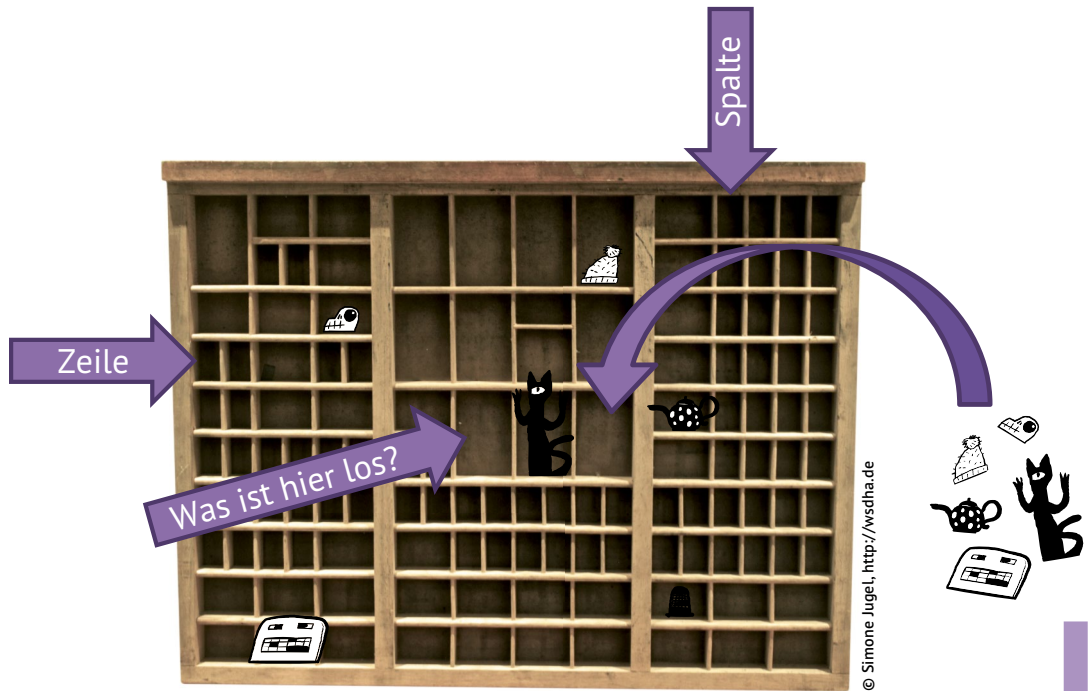
V_MAIN an.

Hmm, zuerst wechsle ich in den View **V_MAIN**,
dann auf den Karteireiter Inbound-Plugs,
dort schreibe ich in der ersten Spalte (Plug-Name) **IN**
und ergänze noch eine nette Beschreibung,
so wie „Hier geht's rein“.

Dannach wechsle ich auf den Karteireiter Outbound-Plugs
und schreibe wieder in der ersten Zeile/Spalte den Namen
OUT mit Beschreibung „Raus hier“.

Ein Setzkasten für einen View

Wo du es jetzt schon bis in den View geschafft hast, machen wir hier weiter – und zwar beim **View-Layout**, also der Gestaltung der visuellen Schnittstelle deiner Component. Mir hilft da immer die Vorstellung eines **Setzkastens**.



Denn das **Layout einer Web-Dynpro-Anwendung** ist ein wenig so wie ein Setzkasten: In jedes Kästchen kommen deine Sammelfiguren, Bilder von Ex-Partner*innen, Fingerhüte, was immer du dir wünschst. Die Kästchen im Setzkasten sind nach **Zeilen und Spalten** angeordnet. Doch ab und zu gibt es einen Ausreißer, der sich nicht an diese Regel hält. Der benötigt zum Beispiel mehr Spalten oder Zeilen.

*Also wirklich, es gehört sich nicht, so über Partner*innen zu sprechen! Auch nicht, wenn es Ex-Partner*innen sind!
Und auch nicht, wenn es stimmt,
dass sie mehr Platz benötigen.*

[Belohnung]

Eine kleine Belohnung: Hier kannst du alle Klischees zu Ex-Partner*innen eintragen. Es wurde bewusst nicht die ganze Seite zur Verfügung gestellt, damit bei dir kein Trauma entsteht.

Klischee 1: _____

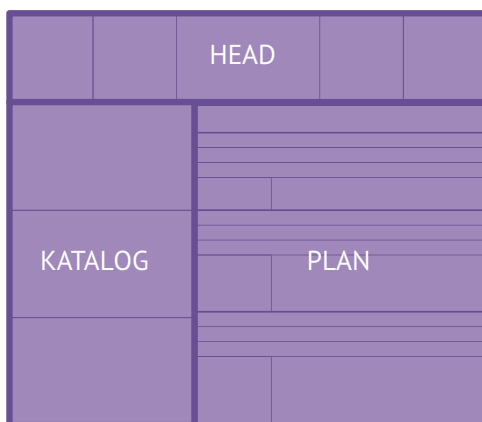
Klischee 2: _____

Klischee 3: _____

...



Ich habe mir zur **View-Gestaltung** unserer Component auch schon einige Gedanken gemacht. Also, wie die Entspannungsplaner-Anwendung aussehen soll. Schau mal:



In der **ersten Zeile**, der **HEAD-Zeile**, sollten einige Bilder erscheinen, in der Mitte eine Überschrift und eine Eingabemöglichkeit für den Namen des oder der Entspannungssuchenden. Darunter in einer **zweiten Zeile** mit zwei Spalten der Entspannungsübungs-KATALOG und rechts davon die gefühlvoll zusammengestellten drei Übungen für einen bestimmten Tag, also der Entspannungs-PLAN.

Die Anordnung in Zeilen und Spalten, das heißt das Layout, übernimmt ein **Layoutmanager**. Web Dynpro stellt dir eine Reihe solcher Layouts zu Verfügung, wie zum Beispiel ein **FlowLayout**, **RowLayout**, **MatrixLayout**, **GridLayout** und **FormLayout**.

Jedes dieser Layouts bietet Spezialitäten an:

- ☛ Das **FlowLayout** («das Demokratische») ordnet die View-Elemente nebeneinander an, und sie bekommen so viel Platz, wie sie benötigen. Falls die Fensterbreite vom Webbrowser nicht ausreicht, werden die View-Elemente einfach in die nächste Zeile verschoben. Quasi ein automatischer Zeilenumbruch.
- ☛ Das **RowLayout** («das Diktatorische») ist wie das FlowLayout, es kennt nur keinen Zeilenumbruch. Die View-Elemente werden beinhart an der Fenstergrenze des Webbrowsers abgeschnitten.
- ☛ Das **MatrixLayout** («der Superstar») denkt in Zeilen und Spalten. Du bestimmst, wann eine neue Zeile beginnt, und das Layout errechnet die benötigten Zeilen und Spalten. Da das MatrixLayout so dynamisch ist, wird es sehr, sehr oft eingesetzt.
- ☛ Das **GridLayout** («der Buchhalter») erwartet von dir, dass du die Anzahl der Spalten kennst, und ist entsprechend unflexibel. Falls die Anzahl der Spalten erreicht ist, wird umgebrochen, und eine neue Zeile beginnt. Punkt.
- ☛ Das **FormLayout** («der rasende Reporter») orientiert sich am Aufbau einer Zeitungsseite in Spalten. Dieses Layout ist das jüngste Geschwisterkind in der Layoutfamilie.

Und wo kann ich das alles einstellen?

Damit das Layout angewendet werden kann, benötigst du zunächst einmal einen sogenannten **Container**. Das ist auch ein View-Element, das aber auf das **Gruppieren von View-Elementen** spezialisiert ist. Und diesem Container teilst du mit, welches Layout du verwenden möchtest, also, wie die View-Elemente im Container anzuordnen sind.

[Notiz]

Der Begriff **UI-Element** bezeichnet ein eigenständiges Oberflächenelement. Dagegen bezeichnet **View-Element** auch die Teilelemente von zusammengesetzten Oberflächenelementen. Ich werde zu Oberflächenelementen immer **View-Elemente** sagen.

Beispiel gefällig? Falls du noch nicht dort bist, wechsele ins Layout von **V_MAIN**.

Verflucht, wo ist
dein nm schon wieder
dieses Layout-Karsteiner?



Web Dynpro Explorer: View zu ZWDC_SCH_014_RELAX anzeigen

Navigation: Zurück, Vorwärts, Suchen, etc.

Muster Pretty Printer Layout-Vorschau ein-/ausblenden Context-Editor Ansicht wechseln

MIME Repository

- Repository Browser
- Repository Infosystem

Web-Dynpro-Comp. / Intf. ZWDC_SCH_014_RELAX

Objektname	Beschreibung
ZWDC_SCH_014_RELAX	Entspannungsplaner
COMPONENTCONTROLLER	
Component-Interface	
INTERFACECONTROLLER	
Interface-Views	
V_MAIN	
Views	
V_MAIN	
Windows	
V_MAIN	

View V_MAIN aktiv

Eigenschaften Layout Inbound-Plugins Outbound-Plugins Context Attribute Aktionen Methoden

Favorites

- CONTEXT_MENU
- ROOTUIELEMENTCONTAINER

Eigenschaft	Wert	Binding
Eigenschaften (TransparentContainer)		
Id	ROOTUIELEMENTCONTAINER	
Layout	FlowLayout	
accessibilityDescription		
contextMenuBehaviour	inherit	
contextMenuId		
defaultButtonId		
enabled	☒	
handleHotkeys	☐	
height		
isLayoutContainer	☒	
labeledBy		
scrollingMode	none	
tooltip		
visible	visible	
width		
Layout (FlowLayout)		
wrapping	☒	

Du landest dann im **View-Editor**: mit der Toolbar zu den View-Elementen links, dem What-you-see-is (nearly)-what-you-get-View-Designer (WYSInWYG) in der Mitte, der View-Element-Hierarchie oben rechts und den Eigenschaften des View-Elementes unten rechts. Die Eigenschaften können bei dir zu Hause unterschiedlich sein, je nachdem, **welches SAP-Release** du verwendest.

Kurze Kaffeepause?



Sorgen für Ambiente: View-Elemente



[Zettel]

Ganz ehrlich: Du kannst zwar die View-Elemente aus der Toolbar per Drag & Drop in den View-Designer hineinziehen, jedoch **kenne ich niemanden**, der das tut. Das Arbeiten

Die View-Elemente werden zur **Gestaltung der Oberfläche** eingesetzt. Es gibt ganz einfache, elementare View-Elemente, wie zum Beispiel Eingabefelder (**InputField**) oder Drucktas-
ten (**Button**). Andere View-Elemente sind dafür gedacht, eine Menge von View-Elementen zu gruppieren (zum Beispiel **TransparentContainer**) – diese werden als **Container** bezeichnet –, und wieder andere View-Elemente sind wiederum aus mehreren anderen View-Elementen zusammengesetzt (auch dazu hab ich ein Beispiel: **Table**). In der View-Element-Hierarchie kannst du dir die Anordnung der Elemente **aus hierarchischer Sicht** ansehen.

mit der View-Element-Hierarchie **ist viel einfacher**.

Unterhalb der Hierarchie findest du die Eigenschaften des in der Hierarchie ausgewählten View-Elementes, das eingestellte Layout, die Layoutdaten und – falls es ein interaktives View-Element ist – auch mögliche Ereignisse.

Der Knoten **ROOTUIELEMENTCONTAINER** in der View-Element-Hierarchie ist immer vorhanden. Das ist das Pendant zum Context-Wurzelknoten, nur eben für den View. Die Eigenschaften sind in **Blöcke** unterteilt, die eine passende Überschrift besitzen.

Einer dieser Blöcke ist zum Beispiel **Eigenschaften (TransparentContainer)**. Dadurch erkennst du, dass der Typ des View-Elementes ein **TransparentContainer** ist. Dieser dient dazu, Elemente anzuordnen, besitzt selbst aber keine optische Repräsentation. Er erledigt seine Arbeit im Verborgenen.

• ☐ CONTEXT_MENUS	
• ☒ ROOTUIELEMENTCONTAINER	

Eigenschaft	Wert	Binding
<u>Eigenschaften (TransparentContainer)</u>		
Id	ROOTUIELEMENTCONTAINER	
Layout	MatrixLayout	
accessibilityDescription		
contextMenuBehaviour	inherit	
contextMenuId		
defaultButtonId		
enabled	☒	
handleHotkeys	☐	
height		
isLayoutContainer	☒	
labeledBy		
scrollingMode	none	
tooltip		
visible	visible	
width		
<u>Layout (MatrixLayout)</u>		
stretchedHorizontally	☒	
stretchedVertically	☐	

Die erste relevante Eigenschaft im Abschnitt **Eigenschaften (TransparentContainer)** ist die **Id**, die innerhalb des Views eindeutig sein muss. Im Fall des **ROOTUIELEMENTCONTAINER** kannst du diese nicht ändern, sonst schon.

Der **ROOTUIELEMENTCONTAINER** nimmt sich ja ganz schön was raus ...



Die Eigenschaft **Layout** bietet dir die wichtige Möglichkeit, das Layout für eben dieses View-Element festzulegen. Ich habe hier **MatrixLayout** eingestellt und damit die Anordnung in Zeilen und Spalten für den gesamten View festgelegt. Damit garantiere ich, dass die Elemente, die später eingefügt werden, immer die von mir gewünschte Position bekommen und das in Reih und Glied, ääh, Zeile und Spalte. Somit werden alle View-Elemente, die dem **ROOTUIELEMENTCONTAINER** zugeordnet werden, **in Zeilen und Spalten** angeordnet.

Die weiteren Eigenschaften sind einstweilen nicht so interessant, jedoch verrät der Name der Eigenschaft bereits die Auswirkung.

[Einfache Aufgabe]

Lies dir die Eigenschaften durch, und überlege, wozu die einzelnen Eigenschaften wohl verwendet werden.



Überlegen als einfache Aufgabe? Puh ...

Der nächste Abschnitt im Bereich Eigenschaften, **Layout (MatrixLayout)**, dient dazu, Einstellungen zum Layout vorzunehmen. Für das **MatrixLayout** stehen zwei Eigenschaften zur Verfügung: **Vertikales und horizontales Strecken** bewirkt, dass die Elemente, die dem Container zugeordnet werden, über die gesamte Breite und Höhe verteilt werden. Außerdem passen solche Streckübungen unheimlich gut zum Entspannungsplaner ... Weitere Abschnitte, die Layoutdaten und die Ereignisse, werden wir später besprechen.

[Einfache Aufgabe]

Setze für den **ROOTUIELEMENTCONTAINER** die Eigenschaft **Layout** auf den Wert **MatrixLayout**, und wähle die Eigenschaft **stretchedHorizontally** über die passende Checkbox aus. Dann muss ich nachher nicht mehr daran denken.

Was will es? Die View-Elemente alle horizontal strecken? Kann nicht sein, das schaut ja eigenartig aus. Ach, vielleicht die Elemente horizontal über die gesamte Breite verteilen! Ha, das ist es!

View-Elemente in Rehearsal (diese Überschrift ergibt erst nach der folgenden einen Sinn)

Welches View-Element kennst du, um weitere
Elemente anzuordnen?

[Zettel]

Caipiroshka

- * 3 Teile Wodka
- * 1 Teil Zuckersirup
- * 4 Limettenviertel

Die Limetten und den Zuckersirup in ein
gekühltes Glas geben, mit zerstoßenem
Eis auffüllen und den Wodka zugeben.
Und dann: Rühren und genießen!

*Persönlich kenne ich den
TransparentContainer. Und
vom Hörensagen die **Group**.
Ach ja, das **Tray**-Element ist
gestern auf der Web-Dyn-
pro-Geek-Cocktailparty er-
wähnt worden. Die **Caipi-
roshka** war wirklich lecker.*

Da kennst du ja schon einige **Container**, ich ergänze noch das **Panel**-Element. Und
dann gibt es ja noch diese **Quasi-Container**, in die nur bestimmte View-Elemente einge-
fügt werden dürfen. Da ist aus meiner Sicht das **ViewContainerUIElement** das
wichtigste, da du damit einen kompletten View einbinden kannst.

Element anlegen

Id	tc_head
Typ	TRANSPARENT_CONTAINER Trans

✓ ✗

Wir werden diesen Dialog ungefähr 99 Mal
aufrufen. Jetzt, beim hundertsten Mal (rückwärts
gezählt), trägst du die **Id** des neuen Elementes
ein (**TC_HEAD**, das wird der head = _____,
bitte selbst einfügen!) und wählst den Typ
TransparentContainer aus. Übrigens ist es
irrelevant, ob du die **Id** in Groß-, Klein- oder
CamelCase-Buchstaben schreibst. Das System
setzt es automatisch in Großbuchstaben um.

Jetzt wird angebaut: **Der Setzkasten beginnt zu
wachsen**. Zuerst eine obere Zeile für den Titel des Pla-
ners und ein paar Grafiken. Und dann eine untere Zeile für
die Übungsliste und den Plan über die ganze Breite,
die wiederum weitere Elemente beinhalten soll. Man
könnte auch sagen: eine Spalte, zwei Zeilen. Verwende das
Kontextmenü des **ROOTUIELEMENTCONTAINER** und
den Menüeintrag **Element einfügen**, um den Dialog
namens **Element anlegen** zu öffnen.

*Und lass mich raten: Das mache ich,
um den Kopfbereich zu gestalten?*

Ganz genau. Mit dem grünen Häkchen bestätigst du deine Eingabe, und das neue **View-Element** erscheint in der View-Element-Hierarchie unterhalb des **ROOTUIELEMENTCONTAINER**.

Bei genauerer Betrachtung fällt dir bestimmt auf, dass unterhalb der Eigenschaft **Layout** die Eigenschaft **Layoutdaten** angezeigt wird. Diese Eigenschaft bezieht sich auf den darüber liegenden Container und dessen Layouteinstellung. Nachdem im **ROOTUIELEMENTCONTAINER** das **MatrixLayout** eingestellt ist, legst du mit der Eigenschaft **Layoutdaten** fest, ob mit dem aktuellen View-Element eine neue Zeile beginnt (Wert = **MatrixHeadData**) oder das Element ohne Zeilenwechsel (Wert = **MatrixData**) in einer neuen Spalte anzuordnen ist.

Der Abschnitt **Layoutdaten (Matrix-HeadData)** dient dir dazu, die Zelle einzustellen, in der sich das aktuelle View-Element befindet. Zum Beispiel könntest du den Hintergrund der Zelle mit der Eigenschaft **cellBackgroundDesign** und dem Wert **plain** mit Weiß einfärben.

Preisfrage: Was musst du bei dieser Eigenschaft einstellen, um einen weißen Hintergrund zu bekommen?

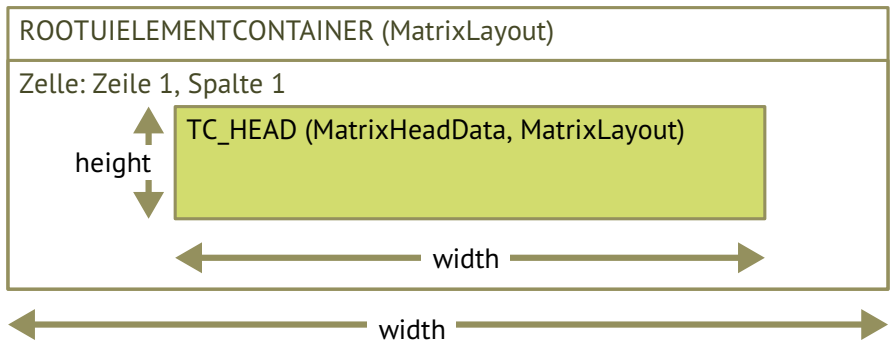
[illegible]

Das neue View-Element ist wieder ein Container. Ein Container im Container. Und eine Rose ist eine Rose ist eine Rose.

So, ich habe geduldig
zugehört. Jetzt hätte
ich aber gerne eine
kurze Zusammen-
fassung.



height

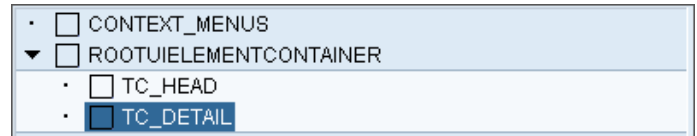


Na klar, kein Problem: Bis jetzt hast du für den **ROOTUIELEMENTCONTAINER** festgelegt, dass das **MatrixLayout** für die Anordnung der darunter liegenden View-Elemente zu verwenden ist. Dann hast du das View-Element **TC_HEAD** vom Typ **TransparentContainer** angelegt und für diesen ebenfalls die **MatrixLayout**-Anordnung festgelegt. Die Eigenschaften **height** und **width** habe ich herausgehoben, damit du den Unterschied zwischen Zelle und View-Element erkennst: Das View-Element hat eine Breite und eine Höhe, die unter den Eigenschaften zu finden sind. Unter den Layoutdaten, also den Daten der umgebenden Zelle, finden sich ebenfalls Anpassungsmöglichkeiten für **width** und **height**.

[Einfache Aufgabe]

Versuch es doch mal selbst:

Wir benötigen eine zweite Zeile, unterhalb von **TC_HEAD**, in der du wieder einen **TransparentContainer** platzierst. Gib diesem den Namen **TC_DETAIL**, hier sollen später der Entspannungskatalog und der Entspannungsplan dargestellt werden.

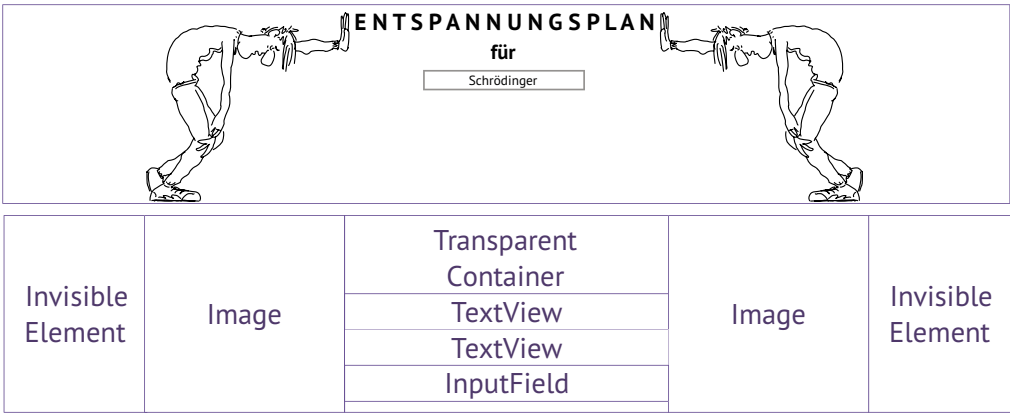


Wenn du das gemacht hast, haben wir schon **zwei View-Elemente**. Zweite Zeile, aber immer noch erste Spalte.

View-Elemente in Concert (wie angekündigt)

Ich vermute, dass die folgende Erklärung nun wie ein Schock kommt, aber so ist manchmal die Realität. Ich zeige dir nun die Gestaltung des Kopfbereiches. Du wirst dich fragen: »Warum tut er mir das an?« Ganz einfach, weil du wahrscheinlich in der Realität auch so vorgehen wirst. Frei nach dem Ansatz: **Divide and Conquer**. Zuerst die grobe Einteilung im View schaffen und dann nach und nach die Detailbereiche strukturieren.

Nun denn, die Kopfzeile will gestaltet werden. Dazu der folgende Plan:



Oben siehst du, wie die erste Zeile im Entspannungsplaner **später aussehen** soll. Unten, wie du das **technisch erzeugen** wirst. Die Bezeichnungen in den Kästchen entsprechen den **realen Namen** der View-Elemente.



Roland, alter Vorher-Nachher-König! Das ist ja mal einleuchtend! Aber noch einmal die Frage: Woher hast du die Bilder von mir??

Links und rechts habe ich **zwei unsichtbare Elemente** platziert, die dafür sorgen, dass die anderen Elemente zentral positioniert werden. Darauf folgend, die zwei völlig willkürlich gewählten Bilder.

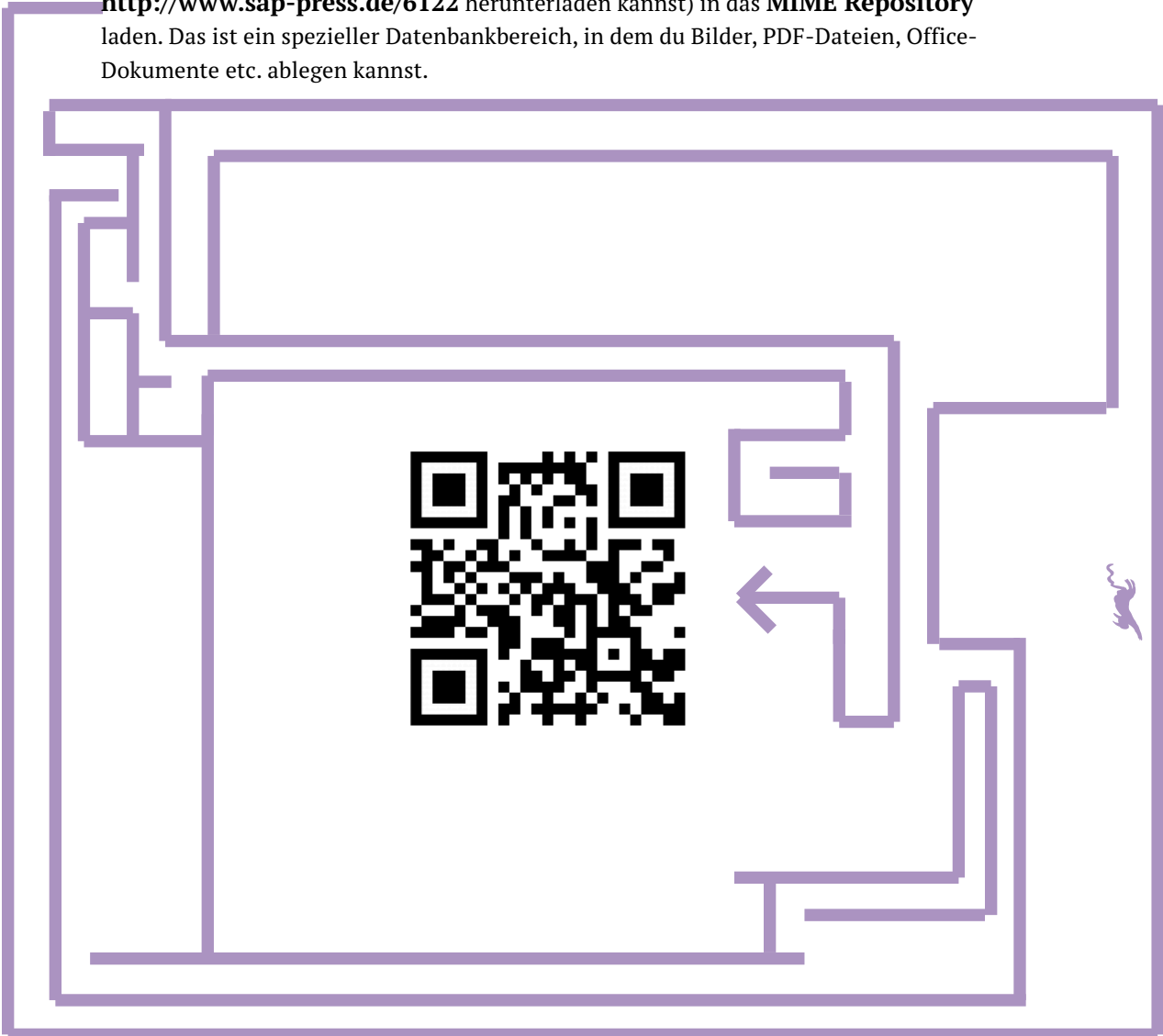
Pah!

Dann kommt das Herzstück des Headers: ein **TransparentContainer**, in dem die Elemente zentriert untereinander platziert werden und dir die Möglichkeit geben, einen Namen einzugeben.

- ☛ Das erste View-Element platzierst du so, wie du schon die **TransparentContainer** zuvor angelegt hast: Kontextmenü des **TransparentContainer (TC_HEAD)** öffnen und den Menüpunkt **Element einfügen** auswählen. Verwende **INV_LEFT** als **Id** und **InvisibleElement** als **Typ**. In den Eigenschaften von **Layoutdaten (MatrixHeadData)** stellst du bei **width** 50% ein. Ist das die Breite des View-Elementes oder der Zelle?

Natürlich der Zelle!

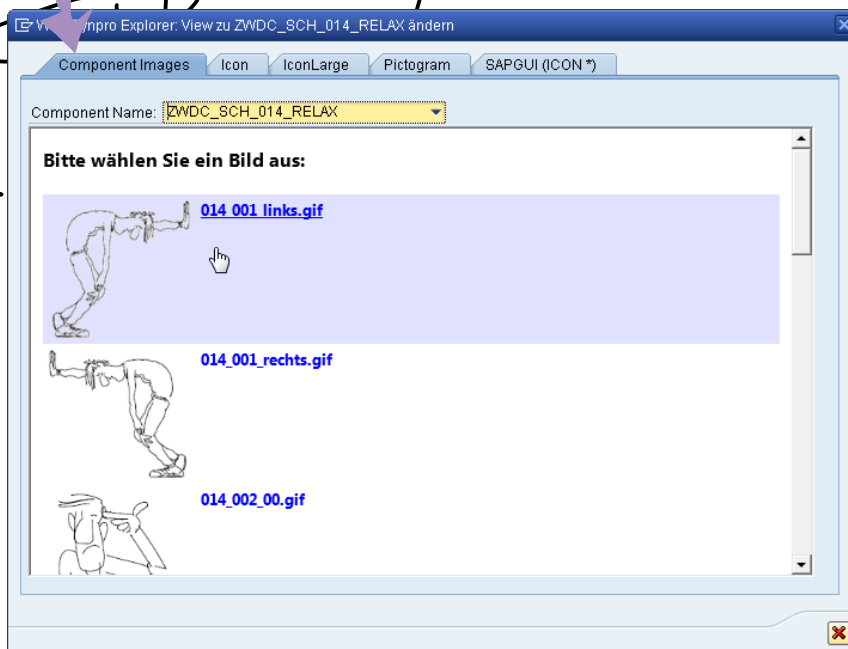
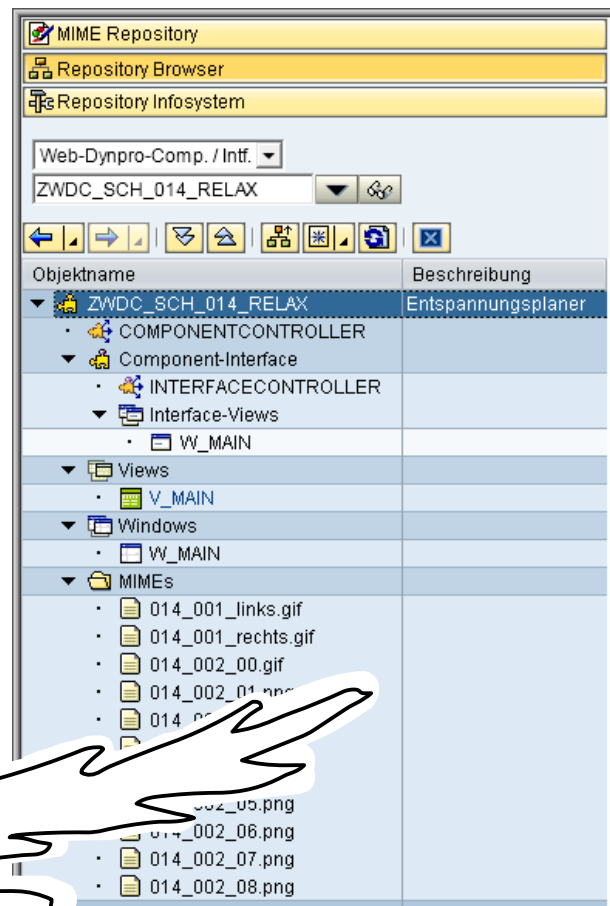
- ☛ Das nächste View-Element wird ein Bild darstellen. Dazu musst du noch Abbildungen (zum Beispiel aus dem Bildpaket, das du in den Materialien zu diesem Buch unter <http://www.sap-press.de/6122> herunterladen kannst) in das **MIME Repository** laden. Das ist ein spezieller Datenbankbereich, in dem du Bilder, PDF-Dateien, Office-Dokumente etc. ablegen kannst.



Verwende zum Hochladen der Bilder das Kontextmenü der Component **ZWDC_SCH_014_RELAX** und dort den Punkt **Anlegen • Mime-Objekt • Importieren**. Sobald das System die Bilder hochgeladen hat, erscheint ein neuer Ordner mit dem Namen **MIMES** und den hochgeladenen Dateien.

Weiter im Programm: Lege das View-Element **IMG_LEFT** vom Typ **Image** an. Setze die Eigenschaft **source** auf **014_001_links.gif**. Dazu verwendest du am besten die Eingabehilfe zur Eigenschaft. Sobald du darauf geklickt hast, erscheint der Auswahldialog für die Bilderquellen. Wähle den Karteireiter

Component Images und das erste Bild aus. Das, auf dem ein dir bekannter WoW-Sportler seine Erschöpfung nach rechts auslebt.



Weitere Eigenschaften, die du einstellen bzw. überprüfen kannst, sind zum Beispiel:

- ➡ **Eigenschaften(Image).Layoutdaten = MatrixData**
damit das Bild in derselben Zeile wie das vorhergehende Bild angeordnet wird
- ➡ **Layoutdaten(MatrixData).cellDesign = IrNoPad** um links und rechts vom Bild
keinen Abstand einzufügen
- ➡ **Layoutdaten(MatrixData).hAlign = endOfLine** um das Bild am Ende der Zeile
zu platzieren

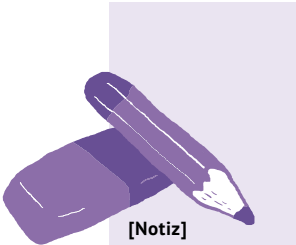
*Was ist denn das für eine Schreibweise, mit dem Punkt
und dem Gleichheitszeichen? Ist das Web Dynpro ABAP?*

Da hat wieder meine Faulheit zugeschlagen: Anstatt zu schreiben: »Im Abschnitt **Eigenschaften(Image)** setzt du den Wert der Eigenschaft **Layoutdaten** auf den Wert **MatrixData**«, verwende ich die Syntax **<Abschnitt>.<Name Eigenschaft>=<Wert>**. Spart unheimlich viel Zeit und Seiten!
Nun kommt das Kernstück der Kopfzeile: **Titel und Eingabefeld**. Für die Anordnung der View-Elemente benötigen wir einen Container. Lege dazu einen **TransparentContainer** mit **TC_HEAD_AREA** als **Id** und den Eigenschaften aus folgender Tabelle an (mit Kurzschreibweise):

Eigenschaft	Wert
Eigenschaften(TransparentContainer).Layout	MatrixLayout
Eigenschaften(TransparentContainer).Layoutdaten	MatrixData
Layout(MatrixLayout).stretchedHorizontally	true
Layoutdaten(MatrixData).cellBackgroundDesign	plain
Layoutdaten(MatrixData).cellDesign	IrNoPad

Im **TransparentContainer** namens **TC_HEAD_AREA** platzierst du ein View-Element **TextView** mit der **Id TV_HEADER1** und folgenden Eigenschaften:

Eigenschaft	Wert
Eigenschaften(TransparentContainer).Layoutdaten	MatrixHeadData
Eigenschaften(TransparentContainer).design	header1
Eigenschaften(TransparentContainer).text	ENTSPANNUNGS PLAN
Layoutdaten(MatrixHeadData).cellDesign	IrNoPad
Layoutdaten(MatrixHeadData).hAlign	center



[Notiz]

Da ist noch eine kleine Unschönheit: die Sprache des Textes. Egal, in welcher Sprache sich die User anmelden, da steht immer ENTSPANNUNGSPLAN auf Deutsch. Das ist möglicherweise für englischsprachige User nicht gewünscht. Merke dir mal das Thema **Online Text Repository**. Damit lassen sich Texte übersetzungsrelevant anlegen.



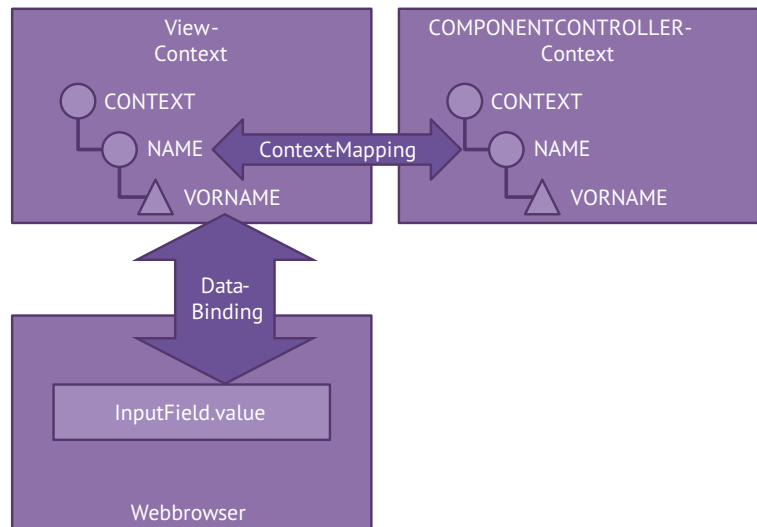


[Einfache Aufgabe]

Lege ein weiteres View-Element **TextView** mit der **Id TV_HEADER2** an.
Verwende die gleichen Eigenschaften wie bei **TV_HEADER1**, ersetze nur den Text durch »für«.

Das dritte View-Element in **TC_HEAD_AREA** ist vom Typ **InputField**. Das besitzt eine besondere Eigenschaft: **value**. Woher kommt dieser Wert? Einerseits kann er der User-Eingabe entstammen, und andererseits, wie wir bereits gesehen haben, kann er in einer Methode produziert werden. Damit die Datenverteilung in beide Richtungen (User – Component, Component – User) laufen kann, benötigen wir zwei Konzepte: **Context-Mapping** und **Data-Binding**.

Ich habe da mal etwas vorbereitet!

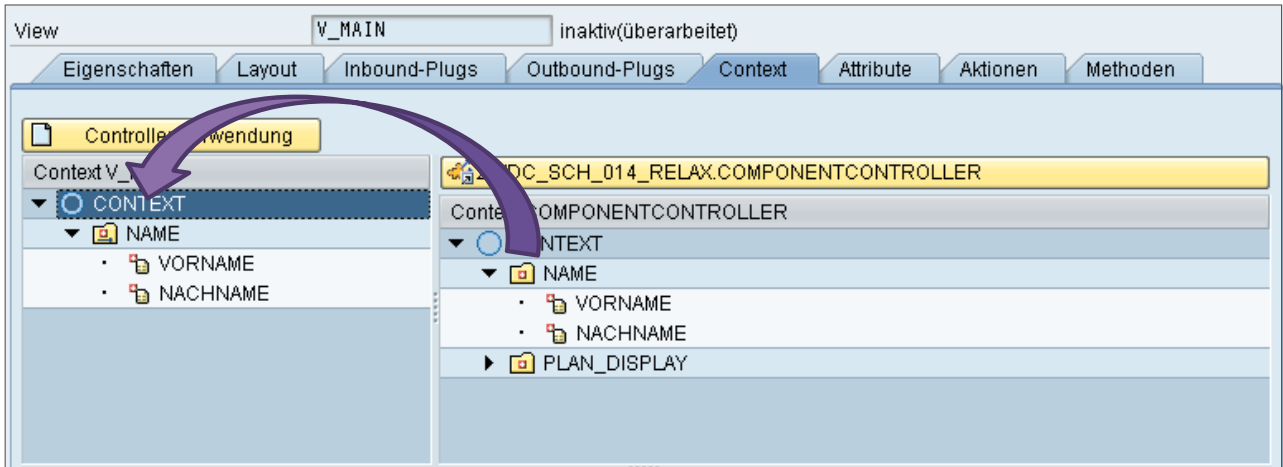


So ähnlich: Ich dachte beim Context-Mapping/Data-Binding weniger an Marionetten, sondern eher an **deine eigene Datenautobahn** in der Web-Dynpro-Component. Die Idee ist die folgende: Beim Context-Mapping werden Daten, die in einem Controller-Context definiert werden, anderen Controllern zur Verfügung gestellt. Durch das Context-Mapping werden die **Sichtbarkeiten beeinflusst** und nicht doppelte Definitionen oder Kopien angelegt. Sofern das Context-Mapping definiert ist, »fließen« die Daten in beide Richtungen, also vom einen zum anderen Controller-Context.

Und beim Data-Binding?

Gleiches Prinzip, nur dass die Eigenschaften von View-Elementen an den Context gebunden werden und der Anwender mit im Spiel ist. Die Datenänderungen kommen von den und/oder werden an die User gesendet.

Das Context-Mapping für unseren Entspannungsplaner ist ganz einfach durchzuführen: Lass dir den **Context** des Ziel-Controllers anzeigen (das ist bei uns der View-Controller, da im **COMPONENTCONTROLLER** der Context bereits definiert ist). Dann ziehst du den gewünschten Knoten **vom Quell-Context zum Ziel-Context**, also bei uns vom **COMPONENTCONTROLLER** zum View-Controller.



Einfach per Drag & Drop – schon sind die Daten im View-Context sichtbar!



[Achtung]

Falls sich nachträglich die Definitionen im Quell-Context ändern, kannst du die Veränderungen ganz einfach mithilfe des Kontextmenüs des gemappten Knotens und dem Menüpunkt **Aktualisieren Mapping** nachziehen.

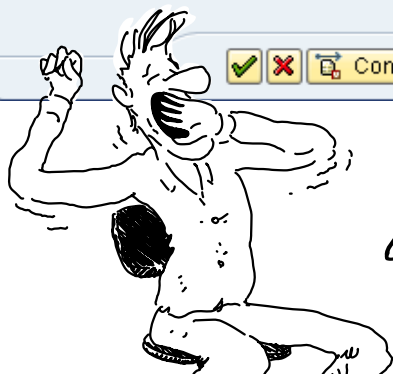
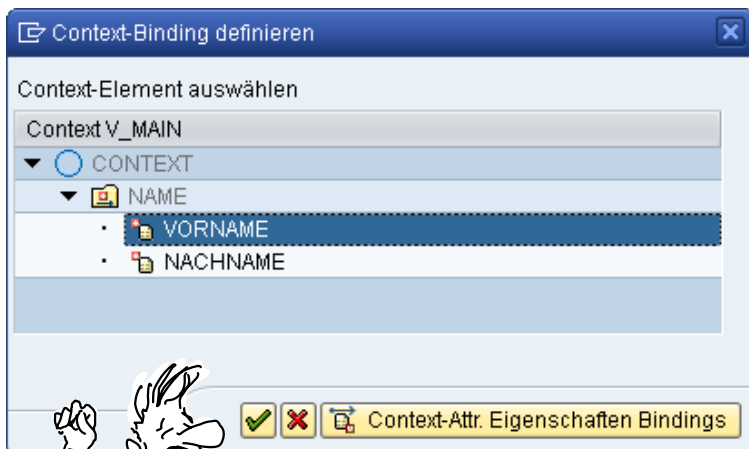
Beim **Data-Binding** werden Eigenschaftswerte von View-Elementen an Context-Knoten bzw. an Context-Attribute gebunden.

Ähnlich, wo ist denn da der Unterschied?

Ob das Data-Binding an einem **Knoten** oder an einem **Attribut** durchgeführt wird, hängt davon ab, was sich das View-Element wünscht. Zum Beispiel benötigt eine Tabelle einen Knoten als Datenquelle, ein Eingabefeld aber nur ein Attribut.

Zurück zum Data-Binding: Auch dies lässt sich durch einen einfachen Vorgang realisieren.

Verwende die **Eingabehilfe** zur View-Element-Eigenschaft im View-Editor, zum Beispiel bei **InputField** die Eigenschaft **value**. Dir wird der gesamte Context angeboten, und du wählst **den passenden Knoten** oder **das passende Attribut** aus. Ob du Knoten oder Attribut wählst, hängt natürlich von der Definition der Eigenschaft ab.



mit der
Lamhann
!edlin os

Nachdem du die Auswahl bestätigst hast (Du weißt schon: Mit dem **grünen Häkchen**, aber das muss ich inzwischen wirklich nicht mehr und so, oder?), ist die Datenbindung abgeschlossen. Hier: auf das Attribut.



[Achtung]

Data-Binding bedeutet nicht nur, dass Daten aus dem Context angezeigt werden. Nein, du kannst damit auch Eigenschaftswerte beeinflussen, wie zum Beispiel **enabled**. Damit kannst du während der Laufzeit den Aktivierungszustand des View-Elementes steuern. Sehr praktisch, wenn du einen Anzeige- und Änderungsmodus implementieren möchtest. Einmal sollte **enabled = 'x'** und einmal **enabled = ' '** sein. Übrigens: Kannst du Binding rückwärts sagen?

So, Genosse Gnidnib, nächste Aufgabe: Im **TransparentContainer TC_HEAD_AREA** platzierst du ein View-Element **InputField** mit der **Id IF_VORNAME** und den Eigenschaften aus folgender Tabelle:

Eigenschaft	Wert
Eigenschaften(TransparentContainer).Layoutdaten	MatrixHeadData
Eigenschaften(TransparentContainer).alignment	center
Eigenschaften(TransparentContainer).value	Data-Binding auf NAME.VORNAME
Layoutdaten(MatrixHeadData).cellDesign	lrNoPad
Layoutdaten(MatrixHeadData).hAlign	center

Das letzte Stück: Nur noch ein paar Meter, und die erste Zeile ist fertig. Da muss ich jetzt nichts mehr erklären. Das kannst du alleine.

[Einfache Aufgabe]

Lege **IMG_RIGHT** und **INV_RIGHT** analog zu **IMG_LEFT** und **INV_LEFT** an. Wenn du möchtest, kannst du dabei mit den 50 % bei der Breitenangabe **width** experimentieren, die für **InvisibleElement** eingestellt sind.

[Belohnung]

Kleines Püschchen? Dann lass uns mal eine Runde um den Block drehen! 50 % von einer Stunde sollten genügen, um deinen Akku wieder aufzuladen.

Und 50% Akkuleistung reichen wiederum, um gepflegt eine Runde zu rocken. Perfekt!

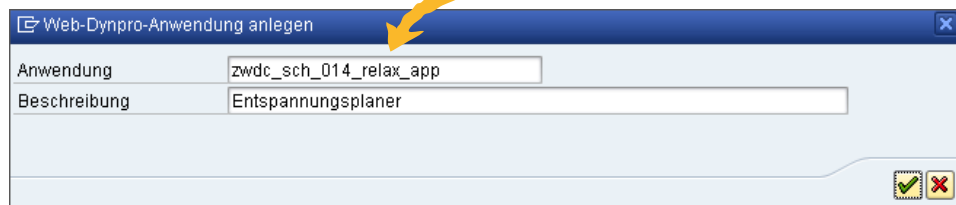


Web-Dynpro-Anwendung und Web-Dynpro-Boot-Camp

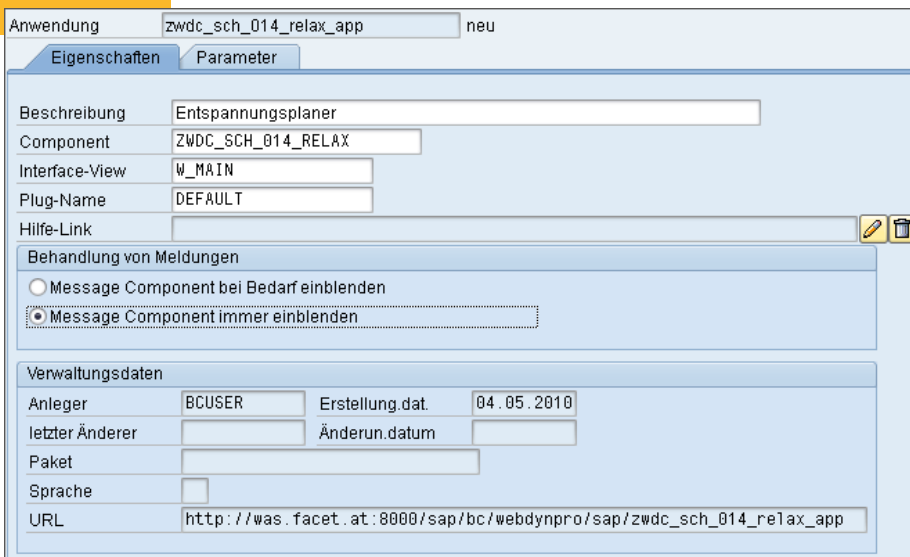
Es wird Zeit, **die Component zu testen**. Direkt kannst du die Component nicht aufrufen, dazu benötigst du eine **Web-Dynpro-Anwendung**. Die ist aber – wie immer – ganz einfach anzulegen. Verwende das Kontextmenü der Component, und wähle den Menüpunkt **Anlegen • Web-Dynpro-Anwendung**. Aber pass auf, dass sie sich nicht mit dir anlegt!

Ja, ja, und Opa Roland war schon in Mailboxen unterwegs, als das noch keine App beantwortet waren.

Im Eingabefeld **Anwendung** kannst du den Namen der Web-Dynpro-Anwendung festlegen. Ich verwende meistens den Namen der Component und hänge ein **_APP** an, oder eben ein **_APP0**, **_APP1** etc., wenn ich mehrere Anwendungen anlegen möchte. **Das habe ich übrigens schon gemacht**, bevor Apps mit kleinen bunten Icons in Verbindung gebracht wurden.



Für den Namen der Anwendung kannst du den Namen der Component verwenden und **_app** anhängen. Du kannst natürlich einen ganz anderen Namen verwenden, du musst jedoch darauf achten, dass der Name im Kundennamensraum liegt, also mit **Z** oder **Y** beginnt, oder in deinem reservierten Namensraum.



Nachdem du eine passende (= aussagekräftige) **Beschreibung** vergeben und alles mit dem kleinen grünen Häkchen bestätigt hast, erscheinen die Details zur Anwendung.

Die Details zur Anwendung mit der Component, dem Start-Interface-View und dem Start-(Inbound-)Plug-Namen. Die Anzeige des Meldungsbereichs stellst du mit den Radiobuttons in der Gruppe **Behandlung von Meldungen** ein.



Die Anwendung bezieht sich auf deine Component, daher wird der Name deiner Component vorgeschlagen. Dann kommt der **Interface-View**.

Also ein Window?

Schrödinger, Schrödinger, ziemlich beeindruckend!
Die Pause hat dir wohl gut getan!

Die Basis des Interface-Views ist das Window **W_MAIN** – und das sollte beim **Starten der Anwendung** aufgerufen werden. Da du beim Aufrufen des Windows Daten mithilfe des Inbound-Plugs übergeben kannst, musst du auch den Plug angeben, der in der Anwendung einzusetzen ist. Da gibt es noch ein paar **spannende Sachen** zu entdecken.

- ☛ Zuerst einmal die Einstellungsmöglichkeit, ob der **Nachrichtenbereich** (da werden die **Messages** angezeigt, zeig ich dir später) immer oder nur im Bedarfsfall anzuzeigen ist. Ich zeige den Bereich immer an, da das runter- und raufspringen der View-Elemente bei den Usern möglicherweise Schwindelgefühle auslöst.

Verstehe ich nicht: Was oder was hüpf hier?

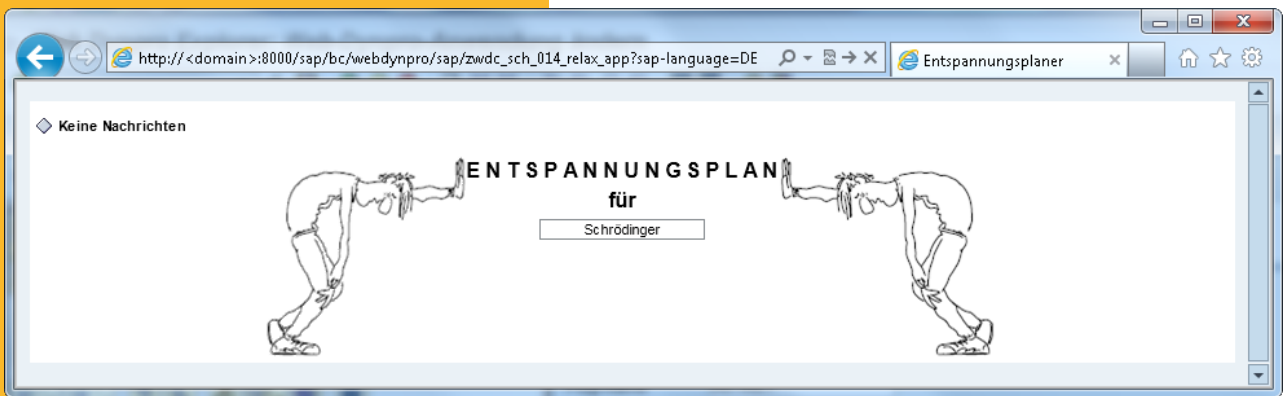
- ☛ Falls der Nachrichtenbereich nur bei Bedarf eingeblendet wird, erscheinen die Meldungen oben. Der Platz, der dafür benötigt wird, wird oberhalb der bereits angezeigten View-Elemente geschaffen, das heißt, diese View-Elemente müssen nach unten wandern – hü-hüpf.
- ☛ Eine sehr zentrale Info steckt in der **URL der Anwendung**. Das ist die Domäne plus Port plus Pfad plus Web-Dynpro-Anwendungsname plus Query-String. **Die URL wird automatisch vom System zusammengestellt**. Domäne und Port sind Systemeinstellungen. Der Pfad ergibt sich aufgrund der Ablage der Web-Dynpro-Component, im Allgemeinen ist das **/sap/bc/webdynpro/sap/<Web-Dynpro-Component>**. Und für diesen Pfad mit Service gibt es auch eine Transaktion namens **SICF**, wobei ICF für **Internet Communication Framework** steht.

[Zettel]

Hatten wir das nicht schon in Kapitel 4?

Also tippe ich, um den Entspannungsplaner zu starten, nachher in die Adresszeile vom Webbrowser was ein?

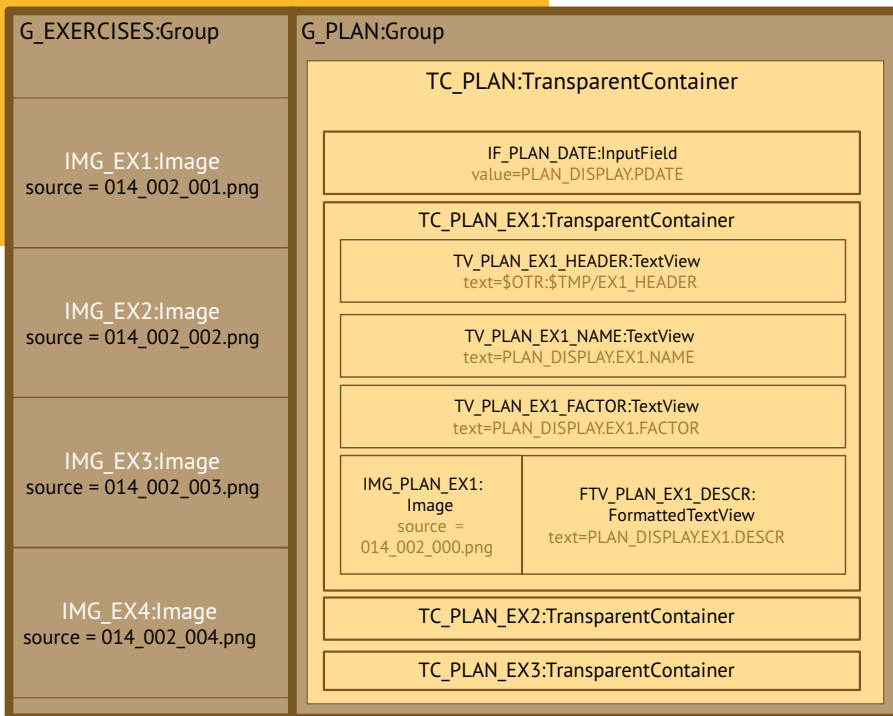
Faul sein! Nichts tippen, sondern die **Testfunktion** verwenden. Die Anwendung ist fertig angelegt, und du kannst sie testen. Verwende dazu entweder das **Kontextmenü der Anwendung** (du findest die Anwendung in der Objektliste zu deiner Component) und den Menüpunkt **Testen** oder die Drucktastenleiste mit dem **Testen-Button (F8)**.



Test, Test, Taaadaaaaaaaa!

Bravo, Bravo! Ist für heute Schluss?

**Wir kennen uns wohl noch nicht lange genug.
Jetzt geht es erst richtig los!**



Ich habe dir einen Plan zusammengestellt, den du ganz locker mit dem bisher gesammelten Wissen umsetzen kannst. Ich werde dir so wenig Hilfestellung wie möglich geben. Fühl dich also wie im Web-Dynpro-Boot-Camp für schwer trainierbare Web-Dynpro-Entwicklerinnen und -Entwickler.

*Schrödinger = Kasper.
Roland = das böse Tier.
SEID IHR ALLE DA UND
NOCH GANZ BEI
TROST?*

Ich habe die Anordnung, Namen, Typen und Context-Attribute für das Data-Binding der View-Elemente in die Grafik eingefügt. Die **TransparentContainer TC_PLAN_EX2** und **TC_PLAN_EX3** sollten genauso wie **TC_PLAN_EX1** aufgebaut sein.



Bevor du aber loslegst, musst du noch das **Context-Mapping** für den Knoten **PLAN_DISPLAY** durchführen, damit du die **Daten im View-Controller** für den Plan zur Verfügung hast. Wenn du fertig bist, melde dich. Ich gehe auf einen Kaffee (Betonung auf dem Doppel-e) – und falls du nicht weiterkommst, einfach schreiben.



Hallo, Roland, bist du schon wieder da? Ich stecke fest! Du hast mir ein paar Fallen eingebaut! Erstens finde ich das Attribut **DESCR** nicht, zweitens fehlt das Attribut **PDATE** ebenfalls, und drittens kennt das System den Text **\$OTR:\$TMP/EX1_HEADER** nicht!

Bei deinem ersten Problem kann ich dir schnell helfen. Da hast du wohl eine **vermeintlich bewusste Inkompetenz**, jedoch ist es einfach zu wenig nachgedacht.

Ich glaube, dass das eine Beleidigung war, aber beweisen kann ich es nicht. Noch nicht!

Erinnere dich noch mal an das **Anlegen einer Struktur im ABAP Dictionary** und die Schritte, die dann folgten, um die Attribute zu definieren und das Mapping durchzuführen.

Das war so: Komponente in der Struktur anlegen, Struktur aktivieren, Context-Knoten-Attribut hinzufügen, Mapping in View-Controller aktualisieren. Ha! Das war ja gar nicht so schwer.



Korrekt, damit solltest du auch deine zweite Frage beantworten können.

Also einfach ein Attribut direkt unter dem Knoten **PLAN_DISPLAY** anlegen.

Ich würde dir vorschlagen, dass du dir ein eigenes Datenelement anlegst, **wegen der Beschriftung des Eingabefeldes**. Der Name könnte zum Beispiel **ZSCH_DE_014_PDATE** lauten und vom Typ **DATS** sein. Und als **Feldtext** verwendest du »Plandatum«.

[Zettel]

Natürlich kannst du noch mal in Kapitel 5 nachschlagen, falls das **Anlegen der Datenelemente** zu lange her ist.

Bei deiner dritten Frage muss ich etwas ausholen, da wir das noch nicht besprochen haben.

Nachrichten und Internationalisierung



Bei der Programmierung des Selektionsbildes und der Liste hatten wir das Thema **Übersetzung** – manche sagen auch, hier geht es um **Internationalization** oder **I18N**.

[Hintergrundinfo]

Die Programmierung des Selektionsbildes wurde in Kapitel 12 besprochen!

I18N?

Ein I am Anfang, dann 18 Buchstaben dazwischen (internationalizatio) und ein N am Ende: I18N.

Danke, R4D. Du willst mir jetzt aber nicht sagen, dass R2-D2 in der Schweiz lebt und Rudolf heißt: Ruedli.

Nein, lieber S9R, denn R2-D2 ist eine griechische Mittelmeerinsel, hinten sogar mit Betriebssystem.

Ach, ich liebe seinen H3R!

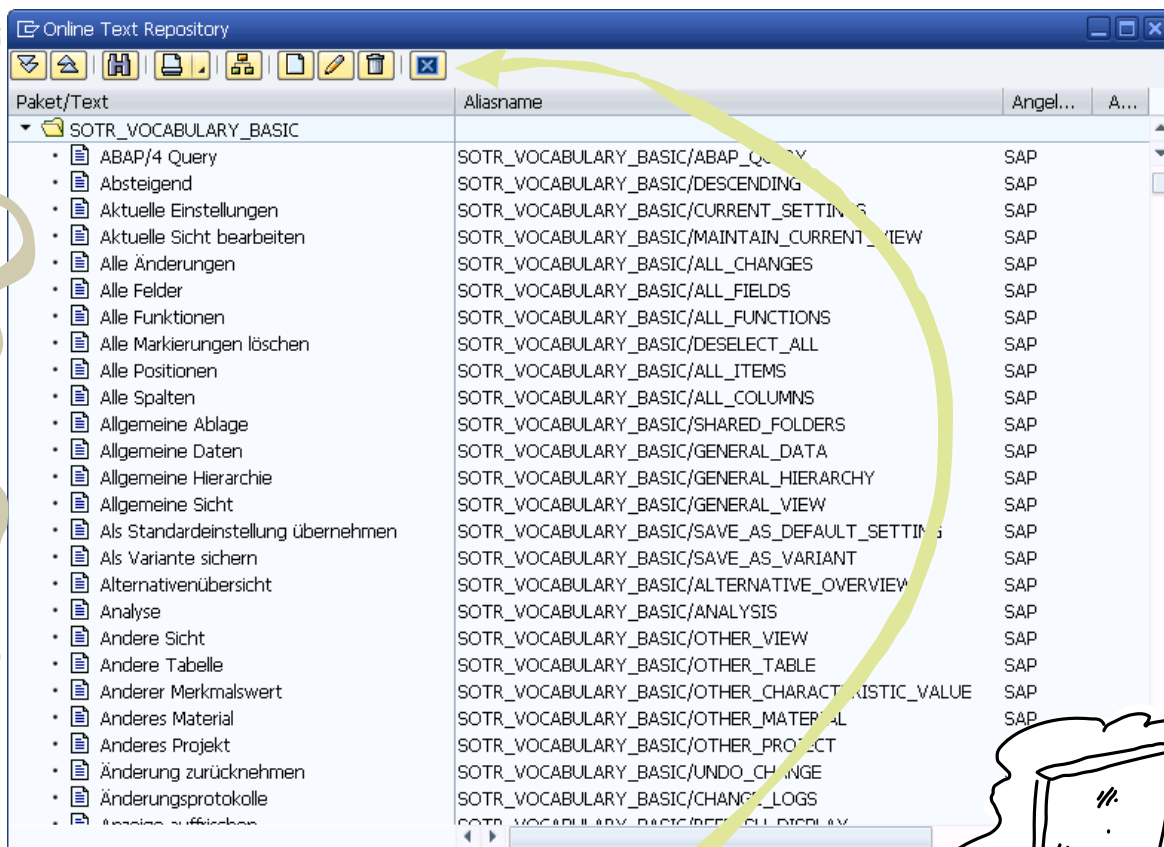


Im Selektionsbild sind die **Selektionstexte** und in der Liste die **Textelemente** und die **Listenüberschrift** übersetzbar. Die Übersetzbarkeit der Selektionstexte kommt durch die **Datenelemente**, die du für die Definition von Parametern und Select-Options verwendet hast.

Und hier kommt jetzt das **Online Text Repository** (abgekürzt OTR) ins Spiel, das du dir vorhin merken solltest. Das Online Text Repository ist eine Ablage für Texte, die in einer Web-Dynpro-Component verwendet werden können: einerseits direkt durch **das Binden an View-Element-Eigenschaften**, andererseits durch **objektorientierte Klassen**, die das Online Text Repository auslesen können (Methode

CL_WD_UTILITIES=>get_otr_text_by_alias).

Das Online Text Repository ist über den Menüpunkt
Springen • Online Text Repository Browser erreichbar.



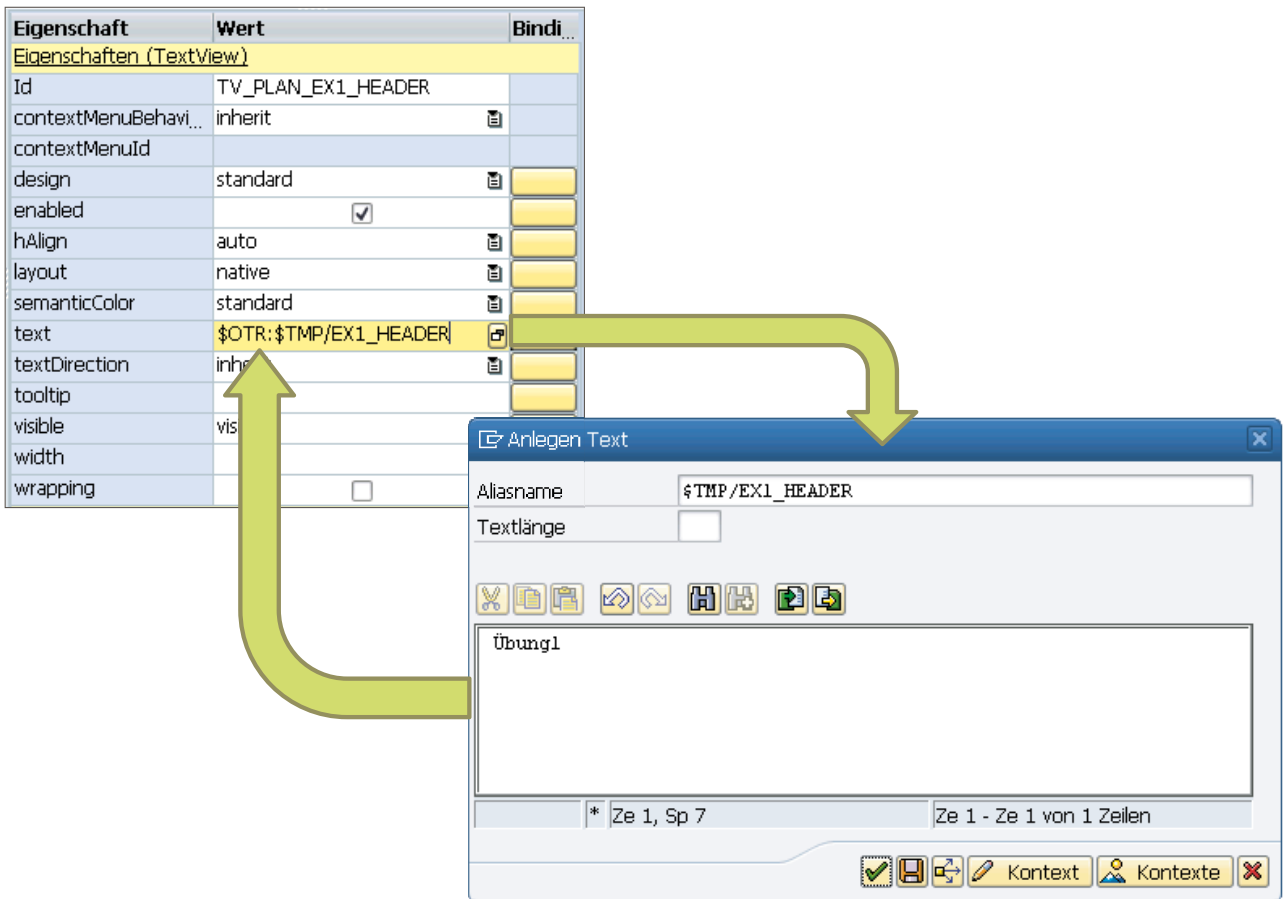
So sieht das Online Text Repository mit dem SAP-Standardvokabular **SOTR_VOCABULARY_BASIC** aus. In der **Toolbar** findest du alle Funktionen zum Anlegen, Ändern und Löschen von OTR-Aliassen und deren Texten.

Die Anlage eines neuen OTR-Eintrages (Web-Dynproisch: **OTR-Alias**) kannst du vom Online Text Repository Browser aus oder – während der Entwicklung am einfachsten – direkt von einer Eigenschaft eines View-Elementes aus starten.

*Also auch von der Eigenschaft **text** des **TextView**-Elementes?*

Ja, ganz genau.





In das Eingabefeld der Eigenschaft gibst du **\$OTR** ein, gefolgt von einem Doppelpunkt und dem Alias. Der beginnt mit dem Namen des Paketes, in dem die Component liegt. In unserem Fall ist die Component im lokalen Paket, also **\$TMP**; dann folgt ein Schrägstrich (/) und der konkrete Name des OTR-Textes, zum Beispiel **EX1_HEADER**. Das ergibt insgesamt **\$OTR:\$TMP/EX1_HEADER**. Nachdem du den Namen eingetippt hast, klickst du doppelt darauf und öffnest damit das Anlagefenster des OTR-Textes. Dort gibst du den Text ein, der im View erscheinen soll, speicherst den Text – und fertig ist der OTR-Text.

Und das noch Schöner daran ist, dass du den OTR-Text wiederverwenden kannst. Ganz einfach, indem du bei einer **View-Element-Eigenschaft die Eingabehilfe verwendest**, die dir dann das Online Text Repository mit dem SAP-Standardvokabular und deine eigenen OTR-Texte anzeigt.

Wo sind meine eigenen OTR-Texte? Ich finde sie nicht!

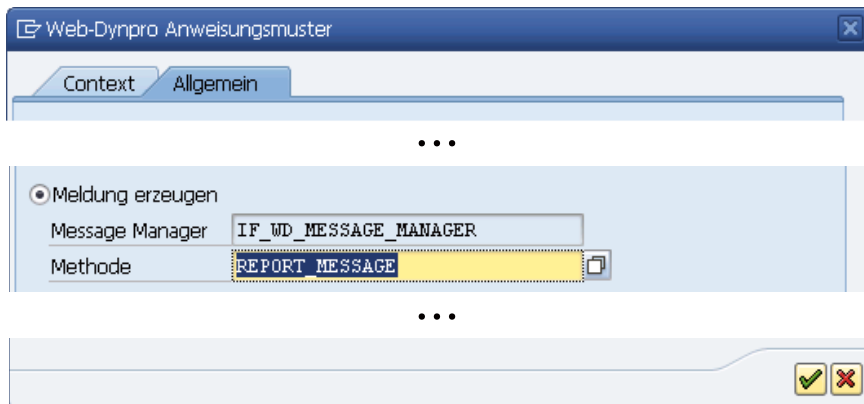
Blättere einfach mal **ganz nach unten** oder **ganz nach oben**, je nachdem, welches SAP-Release du verwendest. Da stehen sie. Falls du einen passenden Text gefunden hast,

einfach (also wirklich einfach) anklicken und mit dem namenlosen grünen Häkchen bestätigen. Daraufhin erscheint der OTR-Alias **als Wert der View-Element-Eigenschaft**.

Da wir schon beim Thema Internationalisierung sind, kann ich dir auch gleich etwas über den **Message Manager** erzählen, der dazu dient, Nachrichten auszugeben.

*Wow, ein Manager, der mal etwas für mich macht!
Und mir auch noch einen ausgibt!*

Genau, einfaches Senden einer Nachricht leicht gemacht! Der Message Manager kann Texte aus der Datenbanktabelle T100, in der die Nachrichten gespeichert sind, Texte von Ausnahmen (die kennst du aus dem Feigling-Kapitel 11) und allgemeine Texte an den Anwender senden. Allgemeine Texte sind ziemlich allgemein, im Speziellen können das Texte aus der **Assistenzklasse** sein, die wir im übernächsten Abschnitt besprechen. Kurz gesagt, ist eine **Assistenzklasse** ein objektorientierter Helfer für die Component, um dort zum Beispiel **Konstanten, Methoden**, aber auch **Textsymbole** anlegen zu können.



Um den Message Manager zu nutzen, verwendest du den Code-Wizard – und zwar den Karteireiter **Allgemein**.

Tä-tä-tä-tä!

Dort findest du den Radiobutton zu **Meldung erzeugen**. Wenn du die Eingabehilfe für die Methoden aufrufst, siehst du eine Menge von Möglichkeiten, um **Nachrichten zu versenden**: von Informationstexten bis hin zu Fehlermeldungen, mit und ohne Bezug zum Context. Den Message Manager werden wir später bei der Zusammenstellung des Entspannungsplans aufrufen.

[Zettel]

Damit wir später auf einfache Art und Weise den Message Manager nutzen können, könntest du jetzt schon dafür Vorbereitungen treffen. Im Sinn von »Was du heute kannst besorgen, ...«. Du erstellst eine Methode, die die Referenz auf den Message Manager ermittelt und daraufhin in den Attributen des **COMPONENTCONTROLLER** setzt.

1. Erstelle die Methode **init_mm**, in der du den Message Manager setzt.
2. Rufe die Methode **init_mm** in der Methode **wddoinit** auf, um die Initialisierung durchzuführen.

Ich fühle mich geehrt, dass du mit mir schon wie mit einem alten ABAP-Hasen sprichst, aber ich fühle mich noch etwas frischer. Bitte gaaaaaaanz gaaaaaaanz gaaaaaaanz!

Kein Problem. Ich denke, das Einzige, was dir Probleme bereitet – zumindest fachlich –, ist die **Ermittlung der Referenz** auf den Message Manager und das Setzen der Referenz. Darum hab ich die Methode **init_mm** für dich vorbereitet.

```
METHOD init_mm .  
*   Message Manager holen  
DATA lo_api_controller TYPE REF TO if_wd_controller. *1  
*   API ermitteln  
lo_api_controller ?= wd_this->wd_get_api( ). *2  
*   Referenz lesen und in den Attributen setzen  
CALL METHOD lo_api_controller->get_message_manager  
    RECEIVING  
        message_manager = wd_this->gr_mm. *3  
ENDMETHOD.
```

*1 Zur Ermittlung der Referenz auf den **Message Manager** musst du die Referenz auf die **Schnittstelle des COMPONENTCONTROLLER** ermitteln. Der passende Typ für die Referenz ist **IF_WD_CONTROLLER**.

*2 Mit der Methode **wd_get_api**, die für alle Controller vordefiniert ist – hier ist also nichts für dich zu tun –, wird dir die Referenz auf den Controller geliefert.

*3 Dann bleibt nur mehr der Aufruf der Methode **get_message_manager**, um die Referenz auf den Message Manager zu ermitteln. Ich habe das Ergebnis der Methode sofort im Attribut **gr_mm** des **COMPONENT-CONTROLLER** abgelegt.

Den Aufruf der Methode **init_mm** in der Methode **wddoinit** kann ich getrost dir überlassen. So alt bist du schon! Einen Punkt habe ich aber noch: die **Assistenzklasse**. Die Assistenzklasse zur Unterstützung der Component.

*Darf ich mich vorstellen:
Assistenzklasse mein Name,
stets zu Diensten!*

Na ja, eigentlich kann die Assistenzklasse vor allem für die **Bereitstellung von allgemeinen Texten** verwendet werden. Feucht durchwischen musst du also weiterhin selbst. Anlegen kannst du eine Assistenzklasse, die übrigens eine ganz »normale« objektorientierte Klasse ist, indem du zur Component-Information wechselst.



Doppelklick auf den Namen der Component in der Objektliste?

Genau. Und dort gibst du im Eingabefeld **Assistance-Klasse** den Namen der anzulegenden Klasse ein, zum Beispiel **ZCL_SCH_14_A**.

Web-Dynpro-Component	ZWDC_SCH_014_RELAX	inaktiv/überarbeitet	
Beschreibung	Entspannungsplaner		
Assistance-Klasse	ZCL_SCH_14_A 		
Anleger	BCUSER	Erstellungsdatum	23.04.2012
letzter Änderer	BCUSER	Änderungsdatum	23.04.2012
Originalsprache	DE	Paket	§TMP
☒ Barrierefreiheit-Prüfungen aktiv			

Jetzt kommt wieder der Doppelklick – der kommt immer irgendwie vor –, und schon bist du im **Anlagedialog der Assistenzklasse**. Wenn dieser abgeschlossen ist, kannst du mit nochmaligem Doppelklick – zwei zum Preis von einem – in die **Pflege der Assistenzklasse** springen.

*Schaut alles ganz normal aus. Außer diese Methode **get_text** ...*

Ja, eben! Der Punkt, um den es mir geht, sind die Textelemente. Wenn du den Menüpunkt **Springen • Textelement** auswählst, gelangst du direkt in die Textpflege. Und das sollte dir nun wirklich bekannt vorkommen. **Das ist wie bei der Pflege der Texte für die Liste in Kapitel 12.**

*Dreistelliges Kürzel vergeben, Text pflegen und mit Transaktion SE63 oder **Springen • Übersetzung** übersetzen. Jaaaa, da stannst du!*

Nachdem du die Texte angelegt hast, kannst du zurück zur Component navigieren. In eine Methode. Dort mit dem **Code-Wizard** auf die Texte in der Assistenzklasse zugreifen, du weißt schon: Karteireiter **Allgemein** und den Radiobutton **Textsymbol-Zugriff** auswählen. **Hast du es bemerkt?**

*Bemerkt? Was hab ich bemerkt?
Immer dieses Leistungsdruck!*

Durch die Anlage der Assistenzklasse wurde das Attribut **WD_ASSIST** in jedem Controller angelegt.

Und wie hätte ich das bemerken sollen, bitte schön?

Dein Blick schweift doch immer über die Attribute der Controller, dort ist es angelegt. Der Funktionsgewinn: Du kannst nun von deinen Controller-Methoden aus direkt auf die **Assistenzklasse zugreifen**. Das werden wir gleich benötigen. Da wirst du die Assistenzklasse für das Lesen von Texten und zur Ermittlung von Übungsdetails verwenden.

[Belohnung/Lösung]

Du hast unglaublich lange durchgehalten!

Möglicherweise ist das der Beginn
einer wunderschönen Zeit als ABAP-
Programmierer, wer weiß ...

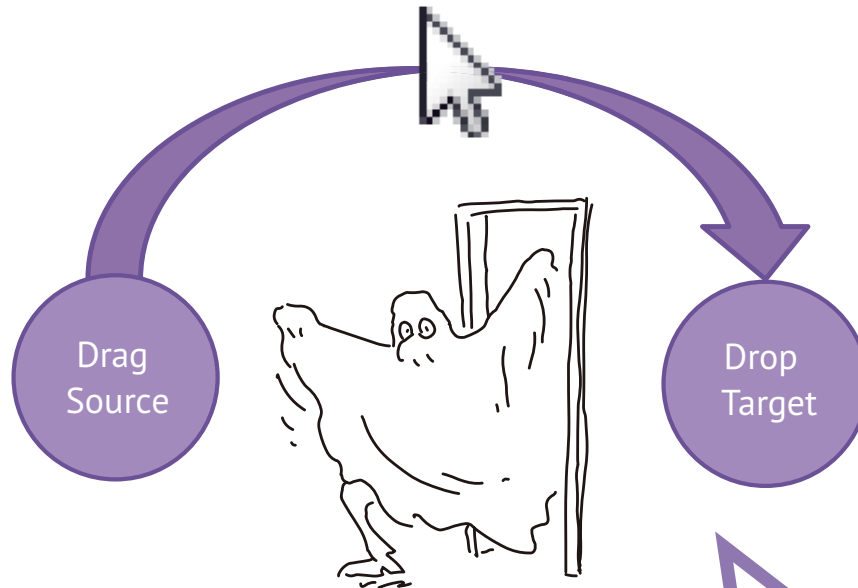
Gönn dir doch – und deiner Freundin –
eine Portion **Salzburger Nockerln**:
Eiweiß fürs Hirn und Zucker für die
schnelle Energiezufuhr. Mmmh!



Fallengelassene Drachen

Von NockerIn besänftigt, wird dein Herz jetzt bestimmt auch nichts dagegen haben, wenn wir die User des Entspannungsplaners ein bisschen verwöhnen.

Dazu bauen wir nun eine Funktion ein, dank der man eine Übung aus dem Entspanningskatalog anklicken und in den Entspannungsplan ziehen kann, um sie dort zu verankern. **Drag & Drop** eben: kein Tippen, nur Klicken.



Beim Drag & Drop sind zwei Rollen zu implementieren: Die **Drag-Rolle** (auch **Drag Source** genannt) stellt die **Quelle** und die **Drop-Rolle** (auch als **Drop Target** bekannt) das Ziel der Operation dar. Während der Drag-&-Drop-Operation wird in Web Dynpro ein Schattenbild (**Ghost Image**) angezeigt, direkt beim Mauszeiger.

Fangen wir mit der **Drag-Source-Rolle** an. Jedes der Bilder aus dem Entspanningskatalog sollte für den Entspannungsplan verwendbar sein, also per Drag gezogen werden können. Daher musst du zu jedem Bild eine **DragSourceInfo** anlegen.

Und das funktioniert so, zumindest für das erste Bild:

1. In der View-Hierarchie lässt du dir das View-Element zum Bild **IMG_EX1** anzeigen.
2. Dann rufst du das Kontextmenü des Images auf und wählst den Menüeintrag **Drag Source einfügen** aus.
3. Als **Id** verwendest du **DSI_EX1** (die werden wir noch einsetzen).
4. Set the data to **EX1** (that data may be transported from the drag source to the drop target)
5. Set the **mimeType** to **image/png** (that's just for the ghost image).
6. And finally set the tags to **EX1** (that's a string pattern for comparison purposes).

DA,
DA,
DA!



[Achtung]

Lieber Herr Schwaiger, wo haben Sie denn diese Anleitung ... ähem ... entliehen?
Bitte zumindest eindeutschen! Viele Grüße, Ihr Lektorat.

Liebes Lektorat, das hat Sie jetzt aus der Bahn
geworfen, ja? Dann hat es so geklappt, wie ich
wollte. Damit Sie genauer lesen. Denn das ist
mein eigener englischer Text – und nicht
geklaut! Immer für klare Worte zu haben,
Ihr Roland Schwaiger.

[Einfache Aufgabe]

Definiere für alle anderen Images im
Entspannungskatalog eine Drag Source,
analog zu **EX1**. Am Ende sollten
zu allen View-Elementen namens
IMG_EX<nummer> die verschiedenen
DragSourceInfo **DSI_EX<nummer>**
definiert sein.

Nächste Rolle: **Drop Target**. Es gibt zwei Möglichkeiten, das Drop Target zu definieren.

- ☛ Das View-Element, auf das der Anwender etwas fallen lässt, bietet die **DropTarget-Info** direkt an.
- ☛ Oder du benutzt das View-Element **DropTarget**, um das View-Element zu »verschalen«, das die **DropTargetInfo** nicht direkt anbietet.

Verstehe ich nicht. Wieso verschalen?

IMG_EX1		
• DSI_EX1 [DragSourceInfo]		
Eigenschaft		
Wert		
Bindi...		
Eigenschaften (DragSourceInfo)		
Id	DSI_EX1	
data	EX1	
enabled	☒	
mimeType	image/png	
scope	componentInstance	
tags	EX1	

Damit ist die Definition der Drag Source abgeschlossen. It's your turn.

Mit Verschalen meine ich, dass das View-
Element ein Untererelement des View-
Elementes **DropTarget** wird und so die
Eigenschaften für das **Fallenlassen**
(Droppen) übernimmt. Wenn ich darüber
nachdenke, gefällt mir der Begriff **Erben**
besser als Verschalen.

DE
DUDUDU
DE
DADADA

[Einfache Aufgabe]

Baue meine Erklärung zum View-Element **DropTarget** so um, dass das Wort (ver-)erben anstatt verschalen verwendet wird.

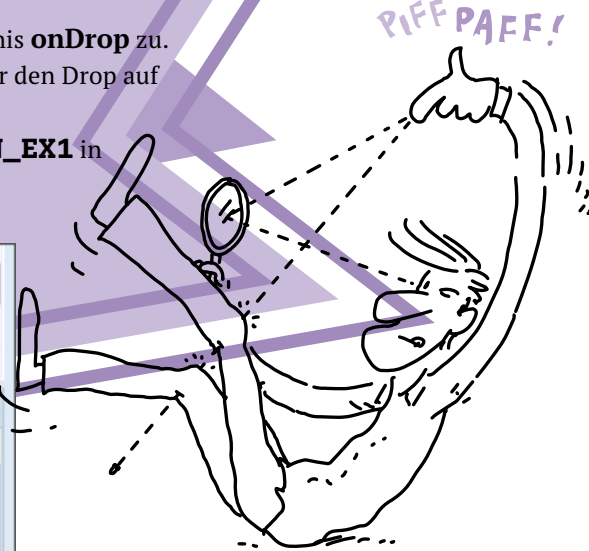
Die zweite Möglichkeit ist deshalb der beste Weg, um dem Image beizubringen, dass es sich als **Drop Target** verhalten soll. Ich beschreibe wieder den Weg für die erste Entspannungsübung im Plan:

1. Wechsle in der **View-Element-Hierarchie** vor das Image für die erste Übung.
2. Füge das neue View-Element **DropTarget** ein, und setze die **Id** auf **DT_PLAN_EX1**.
3. Lege die Aktion **DROP** an, und ordne sie dem Ereignis **onDrop** zu. Die Aktion wird dann ausgelöst, wenn der Anwender den Drop auf das Element ausführt.
4. Verschiebe per Drag & Drop das Image **IMG_PLAN_EX1** in das **DropTarget** namens **DT_PLAN_EX1**.

DT_PLAN_EX1

- IMG_PLAN_EX1 [Content]
- DTI_PLAN_EX1 [DropTargetInfo]

Eigenschaft	Wert	Bindi...
Eigenschaften (DropTarget)		
Id	DT_PLAN_EX1	
Layoutdaten	MatrixData	
contextMenuBehaviour	inherit	
contextMenuId		
enabled	☒	
tooltip		
visible	visible	
Ereignisse		
onDrop	DROP	



Du erklärst hier aber jetzt gerade nicht Drag & Drop mit DropTarget?!
Das glaube ich jetzt nicht.

5. Lege per Kontextmenü (Menüeintrag **Drop Target einfügen**) des View-Elementes **DropTarget** die **DropTargetInfo** mit einer **Id** namens **DTI_PLAN_EX1** an, und setze die Eigenschaft **tags** auf **EX***.

Da ist er wieder, der Asterisk!

Vielleicht noch ein paar Worte zu den Tags.

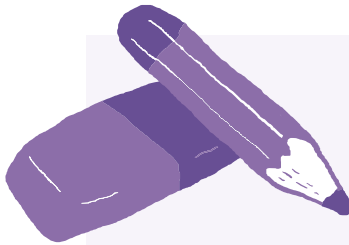
- Die **Drag Source** kann eine **Menge von Tags** besitzen, die in der Eigenschaft **tags** durch Leerzeichen getrennt aufgelistet werden.
- Beim **Drop Target** wird in der Eigenschaft **tags** möglicherweise eine Liste mit Tags angegeben, wobei diese Platzhalter (*) beinhalten können.
- Die **Drop Operation** ist dann möglich, wenn eines der Tags aus der Drag Source mit einem der Tags aus dem Drop Target **per Mustervergleich** übereinstimmt.

- DT_PLAN_EX1 - IMG_PLAN_EX1 [Content] - DTI_PLAN_EX1 [DropTargetInfo]		
Eigenschaft	Wert	Bindi...
Eigenschaften (DropTargetInfo)		
Id	DTI_PLAN_EX1	
enabled	☒	
name		
scope	componentInstance	
tags	EX*	



Könntest du das an einem Beispiel ...

Kein Beispiel, selbst erkennen! Auf dem folgenden Zettel sind Aussagen zu den Drag-Source-Drop-Target-Tags zu finden. Streiche die falschen Aussagen durch, und kreuze die richtigen an. Du darfst für die Lösung auch KI verwenden.



[Notiz]

- ☐ **EX1** passt zu **EX1**
- ☐ **EX1** passt zu **EX***
- ☐ **EX1** passt zu **E***
- ☐ **EX1 EX2 EX3** passt nicht zu **EX***

Die Lösung zur Aufgabe lautet im Morsecode:

• - - - - • -
 • - - - - • -
 • - - - - • -
 - • • • • •

Dein persönlicher Assistent

Wow! Erst ein hilfreiches Manager,
jetzt ein Assistent! Web Dynpro gefällt mir
immer besser!

Jetzt fehlt noch die Implementierung der **Drop-Aktion**, also der Methode **onactiondrop**. Beim Drop sollten die Daten zur »gedragten« Entspannungsübung im Plan angezeigt werden.

Ich mache kein schlechtes Wortspiel mit „Drag Queen“.
Ich mache kein schlechtes Wortspiel mit „Drag Queen“.
Ich mache kein schlechtes Wortspiel mit „Drag Queen“.
Ich mache ...

Die Drop-Aktion hast du schon angelegt. **Jetzt schnüren wir den Braten und servieren dazu noch eine kleine Beilage.**

Beilage zuerst! Du kannst Texte im Online Text Repository, in der Tabelle T100 und in den Textelementen der Assistenzklasse definieren. Das hatten wir ja schon. Diese Möglichkeiten der Textanlage sind allesamt längenbeschränkt. Für längere Texte kannst du

Transaktion SE61 (Dokumentenpflege) verwenden, sie dient zur Ablage von Dokumenten unterschiedlicher Art, wie zum Beispiel **allgemeinen Textdokumenten**.



Beim Einstieg in Transaktion SE61 legst du die Dokumentenklasse fest. Wir benötigen einen **allgemeinen Text**. Dann die Sprache auswählen und zu guter Letzt den Namen des Dokuments (Vorschlag: **ZSCH_014_EX1_DESCR**).

Ähem, du hast dich verippt:

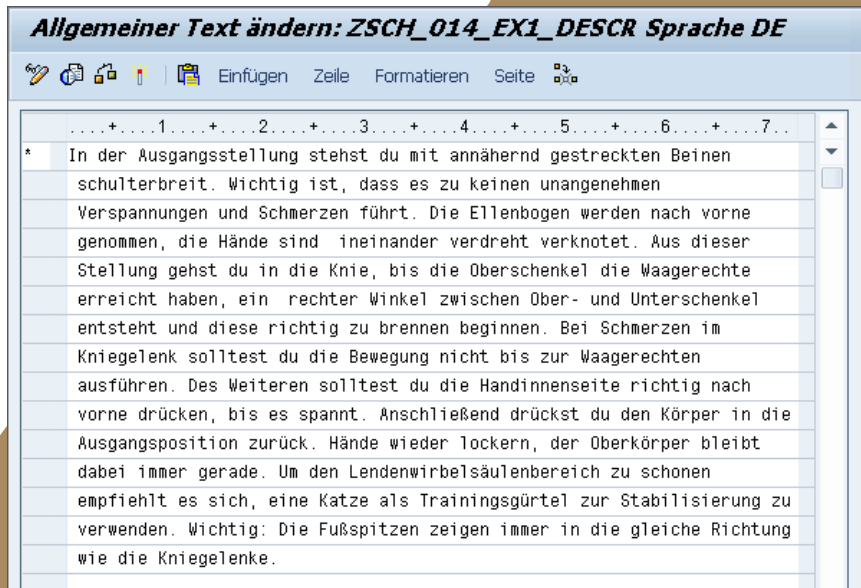
Da steht UE1, gesagt hast du aber EX1!

Gut aufgepasst. Nein, das hat nichts zu bedeuten.

Das war ein »Ist-Schrödinger-noch-anwesend-Test«.

EX1 ist richtig.

Mithilfe des **Anlegen**-Buttons kommst du in die Pflegeansicht, und dort kannst du deine Beschreibungen zu den Übungen pflegen.



Eine etwas antiquarische Detail-sicht zur Textpflege: Ich habe den »alten« zeilenbasierten Texteditor verwendet. Wir wollen ja nur Texte ablegen, da kann man ruhig in Nostalgie schwelgen. Das Ausfüllen der anderen Beschreibungen überlasse ich deiner Kreativität. Falls du den neuen Editor verwenden möchtest, musst du im Menüpunkt **Springen • Editor** die Checkbox **Grafischer PC-Editor** setzen und damit den »neuen« Texteditor einstellen.

Nun ist alles bereit für den letzten Anstieg. Wadln noch massieren und los. Sozusagen der **Großglockner** der Web-Dynpro-Programmierung: Wechsle **ungedopt** in die **Assistentenklasse**! Dort wirst du das Bereitstellen der Übungsdaten implementieren.

Was nun dort?

Die Assistentenklasse ist **von allen Controllern aus erreichbar** – also auch vom View-Controller aus, in dem die Aktionshandler-Methode **onactiondrop** angelegt ist. Von dieser aus werden wir die Assistentenklasse bzw. ihre Methode später aufrufen.

Sobald du in der Assistentenklasse gelandet bist, wirst du eine Methode anlegen, die die Beschreibung zu einer Übung zurückliefert. Die Methode wirst du später verwenden, um die Gesamtinformationen zu einer Übung zusammenzustellen und im View abzubilden.

Also lege die Methode **get_descr_4_ex** mit dem **Importing-Parameter ID_DOC_NAME** und **Returning-Parameter RD_DESCR** jeweils vom Typ **STRING** an. Im Parameter **ID_DOC_NAME** wird der Name des Transaktion-SE61-Dokuments stehen. Das erwartet die Methode. Dem Parameter **RD_DESCR** wird in der Methode der Text zugewiesen.

METHOD get_descr_4_ex.

* Texttabelle

```
DATA: lt_lines TYPE tline_tab,*1  
      ls_line LIKE LINE OF lt_lines,
```

* Typkonvertierung für Funktionsbaustein

```
ld_name TYPE dokhl-object.*2
```

* Typkonvertierung

```
ld_name = id_doc_name.*3
```

* Text ermitteln

```
CALL FUNCTION 'DOCU_GET'*4
```

```
EXPORTING
```

```
id = 'TX'
```

```
langu = sy-langu
```

```
object = ld_name
```

```
TABLES
```

```
line = lt_lines
```

```
EXCEPTIONS
```

```
no_docu_on_screen = 1
```

```
no_docu_self_def = 2
```

```
no_docu_temp = 3
```

```
ret_code = 4
```

```
OTHERS = 5.
```

```
IF sy-subrc <> 0.*5
```

* Implement suitable error handling here

```
ELSE.
```

* Rückgabewert erzeugen

```
LOOP AT lt_lines INTO ls_line.*6
```

```
CONCATENATE rd_descr ls_line-tdline INTO rd_descr.
```

```
ENDLOOP.
```

```
ENDIF.
```

```
ENDMETHOD.
```

*1 Der SAP-Standard-Funktionsbaustein **DOCU_GET** ermittelt die Texte zu einem Dokument. Die interne Tabelle **lt_lines** übernimmt

die Textdaten vom Funktionsbaustein.

Ach ja, und eine Struktur benötigen wir noch für das **Abarbeiten** der Texttabelle.

*2 Funktionsbausteine sind ziemlich sensibel bezüglich der Typen, die für die Aktualparameter verwendet werden.

Funktionsbausteine sind also weiblich ...

*3 Merken: **Typkonvertierungen** sind einfach!

*4 Das ist der Aufruf des Funktionsbausteins **DOCU_GET**, natürlich mithilfe von Mustern produziert. **Gute Entwicklerinnen und Entwickler sind faul!** Das ist beinahe deren wichtigste Eigenschaft – außer vielleicht die Fähigkeit, Kaffee in Programme umzuwandeln.

*5 Falls der Text **nicht gefunden wird**, sollte nichts passieren ...

*6 ... aber **falls er gefunden wird**, folgt das Abarbeiten der Texttabelle und das Konkatenieren (Hintereinanderhängen) der Textzeilen.



Wunderbar, das Lesen der Übungsbeschreibungstexte aus Transaktion SE61 ist damit implementiert. Und jetzt kommen wir noch zu der Methode, um die gesamten Daten für die Übungen zusammenzustellen.

[Einfache Aufgabe]

Lege die Methode **get_details_ex** mit dem Importing-Parameter **ID_NUMBER** vom Typ **STRING** und dem Returning-Parameter **RS_EX** vom Typ **ZSCH_14_TS_UI_EXCERCISE** an.

```
METHOD get_details_ex.
```

```
  CASE id_number.*1
```

```
  * Übung 1
```

```
    WHEN 'EX1'.
```

```
      rs_ex-image = '014_002_001.png'.
```

```
      rs_ex-factor = 'Entschleunigt um 50 %'(e01). *2
```

```
      rs_ex-descr =
```

```
        me->get_descr_4_ex( 'ZSCH_014_EX1_DESCR' ). *3
```

```
      rs_ex-name = 'Kniebeuge mit Fingerkrachen'(e02).
```

```
  * Übung 2
```

```
    WHEN 'EX2'.
```

```
      rs_ex-image = '014_002_002.png'.
```

```
      rs_ex-factor = 'Entschleunigt um 30 %'(e03).
```

```
      rs_ex-descr =
```

```
        me->get_descr_4_ex( 'ZSCH_014_EX2_DESCR' ).
```

```
      rs_ex-name = 'Radieschen von unten'(e04).
```

```
  * Übung 3
```

```
    WHEN 'EX3'.
```

```
      rs_ex-image = '014_002_003.png'.
```

```
      rs_ex-factor = 'Entschleunigt um 70 %'(e05).
```

```
      rs_ex-descr =
```

```
        me->get_descr_4_ex( 'ZSCH_014_EX3_DESCR' ).
```

```
      rs_ex-name = 'Entspannungskuscheln ohne Partner'(e06).
```

```
  * Und weitere Übungen
```

```
  ENDCASE.
```

```
ENDMETHOD.
```

*1 Aufgrund der angefragten Übung **stellst du die Daten zusammen**.

*2 Kannst du dich noch an Kapitel 12 erinnern, wo wir **Textsymbole** in einer Liste ausgegeben haben? Wir haben direkt hinter dem Text ein dreistelliges Kürzel in Klammern platziert, so wie hier **(e01)**. Durch einen Doppelklick auf das Kürzel kommst du in den Anlagedialog des Textsymbols, und es wird in den **Textelementen** der Assistenzklasse abgelegt. Genau dieses Vorgehen verwendest du hier, damit wir dann später wieder auf diesen Text zugreifen können.

*3 Hier kommt die **Verwendung der Methode get_descr_4_ex**, die du zuvor angelegt hast.

Das war's auch schon, ziemlich unspektakulär.

Kleine Pause, bevor wir alle Puzzleteile zusammenstecken?

[Belohnung/Lösung]

Gehe auf deinen Speicher und kontrolliere, ob deine gesammelten Hugh-Hefner-Lifestyle-Magazin-Ausgaben komplett sind.

Integration

Und, das eine oder andere Interview war doch bestimmt spannend noch einmal zu lesen? Egal, kurz zur Wiederholung.

Wir haben die **Assistenzklasse** verwendet, um dort **Hilfsmethoden zur Ermittlung von Daten** zu Entspannungsübungen zu implementieren. Und wir zäumen das Pferd von hinten auf, implementieren die notwendige **COMPONENTCONTROLLER**-Methode zum Lesen der Übungsdaten und verwenden diese am Ende im Aktionsbehandler **onactiondrop**.

Schritt 1: Wechsle in den **COMPONENTCONTROLLER**, und lege dort die folgende Methode **setctx_ex1** an. Wie der Name schon sagt, dient sie zum **Setzen der Daten** der ersten Übung im Context.

```
METHOD setctx_ex1 .*1
    DATA lo_nd_ex1 TYPE REF TO if_wd_context_node.
    DATA lo_el_ex1 TYPE REF TO if_wd_context_element.
    DATA ls_ex1 TYPE wd_this->element_ex1.
    * navigate from <CONTEXT> to <EX1> via lead selection
    lo_nd_ex1 = wd_context->path_get_node( path =
`PLAN_DISPLAY.EX1` ).
    * @TODO handle non existant child
    * IF lo_nd_ex1 IS INITIAL.
    * ENDIF.
    * get element via lead selection
    lo_el_ex1 = lo_nd_ex1->get_element( ).
    * @TODO handle not set lead selection
    IF lo_el_ex1 IS INITIAL.
    ENDIF.
    * fill static attributes
    ls_ex1 = wd_assist->get_details_ex(`EX1` ).*2
    * set all declared attributes
    lo_el_ex1->set_static_attributes(
        static_attributes = ls_ex1 ).
ENDMETHOD.
```

^{*1} Wie du nicht zuletzt an den englischen Kommentaren sehen kannst, habe ich mir diese Methode fast **vollständig** durch den **Code-Wizard** erzeugen lassen.

Tä-tä-rä-tä-tä!

Beziehe dich auf den Knoten **PLAN_DISPLAY.EX1**. Der Punkt zwischen **PLAN_DISPLAY** und **EX1** markiert den Pfad vom Knoten **PLAN_DISPLAY** zum Knoten **EX1**.

^{*2} Na ja, eine Kleinigkeit hast du doch zu erledigen.

Rufe die Methode **get_details_ex** aus der Assistenzklasse auf. Da du die Daten der Übung **EX1** benötigst, übergibst du **EX1** an die Schnittstelle.

Kleiner Tipp: Verwende dazu die Muster zu ABAP Objects.

In der Drucktastenleiste findest du den Druckknopf **Muster (Strg + F6)**. Drücke diesen, wähle danach den Radiobutton **Muster zu ABAP Objects** aus, und bestätige deine Eingabe mit dem grünen Häkchen. Im Abschnitt **Call Method** gibst du im Eingabefeld **Instanz wd_assist** und für **Klasse/Interface ZCL_SCH_14_A** ein.

Die Methode kannst du dann mit der Suchhilfe ermitteln.

Bestätigen mit dem grünen Häkchen nicht vergessen!

Die Methode **setctx_ex1** ist fertig implementiert und setzt die Werte für die Übung im Context.
 Und jetzt der krönende Abschluss: **die Implementierung des Drop-Ereignisses!**
 Könnte auch ein Filmtitel sein. Wie bei jedem guten Film brauchen wir auch noch lautstarkes Marketing.
 Der **Message Manager** sollte die User mit einer Nachricht für die gute Auswahl loben.

```
METHOD onactiondrop .
* Die Nachricht *1
  DATA lv_text_l01 TYPE string.
* Übungsname
  DATA lv_text_e01 TYPE string.
* Je nach ID das Lesen der Daten durchführen
  CASE id.*2
    WHEN 'DT_PLAN_EX1'.
* Die Daten im Plan setzen
      CALL METHOD wd_comp_controller->setctx_ex1( ).*3
* Nachricht von der Assistenzklasse holen *4
* Übungsname
lv_text_e01 =
  wd_assist->if_wd_component_assistance~get_text( 'E01' ).
* Lobeshymne
  lv_text_l01 =
    wd_assist->if_wd_component_assistance~get_text(
      key   = 'L01'
      para1 = lv_text_e01 ).
* Ausgeben mit dem Message Manager
  CALL METHOD wd_comp_controller->gr_mm->report_success
    EXPORTING
      message_text = lv_text_l01.
  WHEN 'DT_PLAN_EX2'.
* ...
  WHEN 'DT_PLAN_EX3'.
* Und mehr ...
  ENDCASE.
ENDMETHOD.
```

*1 Die zwei Datendeklarationen
lv_text_l01 und **lv_text_e01** sind
 für die Ausgabe der Nachricht mit dem Message
 Manager deklariert.

*2 Die Aktion **DROP** liefert dir die **Id** des
 ausgewählten Elementes.

*3 Dies ist der **Aufruf**
 der Methode aus dem
COMPONENTCONTROLLER, um
 die Daten zu setzen.

*4 Um die User bei Laune
 zu halten, werden wir sie noch loben.
 Dazu wird die folgende Nachricht
L01 in der **Assistenzklasse** angelegt
 und ausgelesen:
**»Die Übung »&PARA1&«
 ist tatsächlich eine gute Wahl –
 Bravo!«**
 Wie du siehst, habe ich den Namen
 der Übung ebenfalls aus der
 Assistenzklasse geholt,
 und zwar für das Textsymbol **E01** (Noch
 mal zurückblättern?).
 Diesen Text kannst du verwenden,
 um **&PARA1&** aus **L01** damit
 zu ersetzen.
 Ganz einfach in der Methode
get_text als Parameter mitgeben.
 Die Nachricht sendest du dann mittels
 Message Manager an die User. Wir waren
 ja so schlau und haben
 die **Message-Manager-Referenz** im
COMPONENTCONTROLLER angelegt
 und bereits initialisiert.
 Ich verwende eine ganz einfache Methode
report_success,
 um den Text auszugeben.

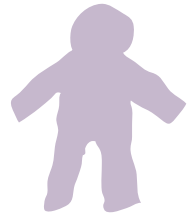
**Geschafft, es ist
 eingerichtet.** Und

wie überprüfen wir das Entspannungs-
 kunstwerk jetzt?

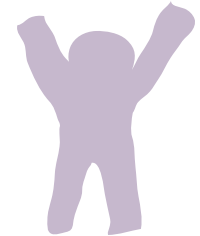
*Wie immer: Alles aktivieren
 und dann testen!*

Und dieses Mal sollte der Drag

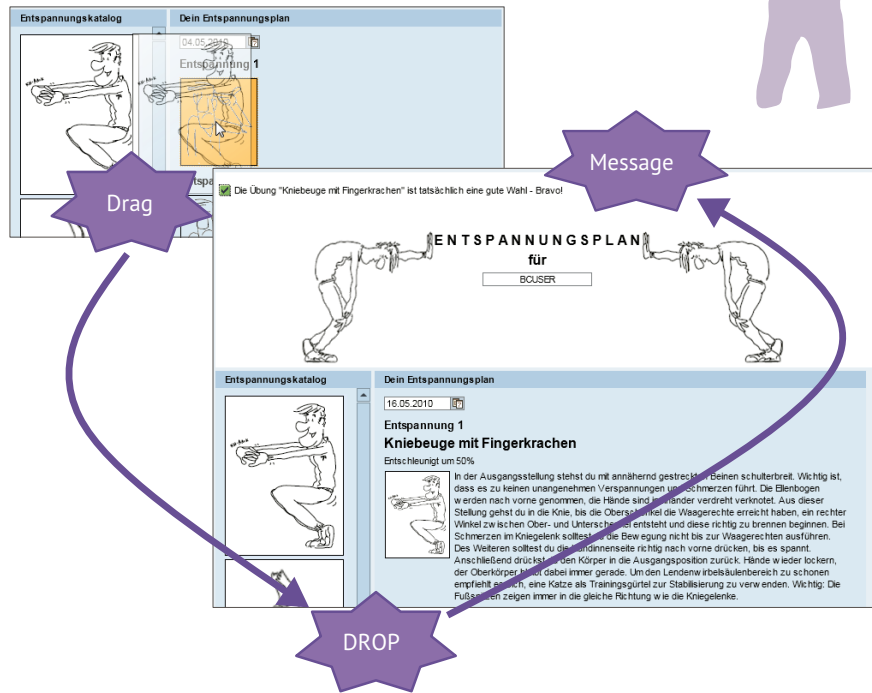
vom ersten Bild im Entspannungskatalog
zum Drop auf das erste Platzhalter-
bild im Plan funktionieren!
Und ich will gelobt werden!



Genauso geht es. Beim **Drag** aus dem Entspannungskatalog siehst du noch das Ghost Image mitwandern, und beim **Drop** – Flupp! – erscheinen alle Informationen zur Übung an den dafür vorgesehenen Stellen. Der Message Manager liefert noch das Tüpfelchen auf dem I und lobt und motiviert.



Der Entspannungsplaner
ist perfekt.



Für dich bleiben noch einige
Fingerübungen, wie zum
Beispiel das Drag & Drop **für
die weiteren Übungen**
zu implementieren oder mehr
Übungen zu ergänzen oder Aus-
nahmen für Entschleunigungen
von über 100 % einzubauen oder,
oder, oder.
Da steckt natürlich noch viel
mehr in Web Dynpro ABAP ...



[Belohnung]

Lies die ganzen Entwicklungsschritte noch einmal, aber dieses
Mal **in verkehrter Reihenfolge**. Beginne hier und blättere
rückwärts.

Und die Belohnung besteht dann darin, ...

... dass kurz vor dem Ende, also am Anfang, ein Bierchen mit Doug
McIlroy auf dich wartet!

