MyTISM - Ein Datenbank- und Anwendungs-Framework

Inhaltsverzeichnis

MyTISM: Das 3-Tier-Framework für Ihre Anwendung . . . 2
Vorstellung von MyTISM . . . 3
- Was ist MyTISM? . . . 4
- Warum MyTISM? . . . 5
- Historie . . . 6
- Zukunft? . . . 9

Schema . . . 10
- Funktionsweise . . . 11
- Vorteile . . . 12
- Schema-Definition . . . 13
  - Attribute . . . 13
  - Unter-Elemente des Schema-Elements . . . 13
  - Include . . . 14
  - Folder . . . 14
  - ModuleProvider . . . 15
  - ModuleIntegrator . . . 15
  - Module . . . 15
  - Generator . . . 15
  - Type . . . 16
  - Interface . . . 16
  - GDPR* . . . 16
    - GDPRDataCategory . . . 16
    - GDPRBusinessInterest . . . 16
    - GDPRProcessingPurpose . . . 17
    - GDPRProcessingLegalBasis . . . 17
    - GDPRLaw . . . 17
    - GDPRRetentionPurpose . . . 17
  - Entity . . . 18
    - Unter-Element "gdpr" von Entity . . . 19
    - Unter-Element "ui" von Entity . . . 19
    - Unter-Element "lookup" von Entity . . . 21
    - Unter-Element "code" von Entity . . . 22
    - Unter-Element "db" von Entity . . . 23
    - Unter-Element "report" von Entity . . . 24
    - Unter-Element "export" von Entity . . . 24
    - Beispiel für eine Entity-Definition . . . 25
    - Attribut . . . 25
- Vordefinierte Datentypen für Attribute . . . 34

Timespan . . . 34
Duration . . . 34
Schemapflege / Datenbankupdates . . . 35
Liste der durch den UpdateHandler zur Verfügung gestellten Hilfsmethoden . . . 35
Coredata-Generator . . . 36
Zusätzliche, vorgebaute Strukturelemente . . . 36
Der Array Datentyp . . . 38
Vordefinierte Arrays von Skalaren . . . 39
Verwendung als Attributtyp . . . 40
Definition von neuen Arraytypen . . . 41
Limitiere die Komponenten . . . 41
Parameter . . . 41
Vererbung . . . 42
Tabellenansicht in der GUI . . . 43
Das "AsRelation" Postfix von Array Attributen . . . 43
Die virtuelle Entität als virtueller Namensraum . . . 44
Zeilen um virtuelle Properties anreichern . . . 45
Selbstdefinierte Tabellenansicht in der GUI . . . 46
Ansicht in Automatikformularen als String . . . 48
ArrayZeilenDelegate / Wrapper . . . 49
Verwendung in OQL Queries . . . 50
Umgang mit Arrays im Code . . . 51
Persistente Array Attribute . . . 52
Verfügbare Methoden . . . 52
Hinweise . . . 52
Hinweise bei v-attrs: . . . 53
Persistenzschicht von MyTISM . . . 54
Löschen von Daten in MyTISM . . . 55
Soft Delete . . . 55
Wiederherstellung von "Soft Deleted" Objekten . . . 55
Hard Delete (Purge) . . . 55
Unterschiede zwischen Soft und Hard Delete im Überblick . . . 56
Zusammenfassung . . . 56
Sprachunterstützung und Internationalisierung . . . 57
Einführung . . . 58
Wo wird Mehrsprachigkeit unterstützt und wie benutze ich sie? . . . 58
Wie wird die konkrete Zeichenkette für einen Schlüssel gefunden? . . . 58
Welche L10nPacks gibt es und wie sind diese organisiert? Wie wird bestimmt, welche L10nPacks nach Texten durchsucht werden? 59
Web . . . 59
Welches sind die "beteiligten bzw. relevanten Objekte"? . . . 59

Wo kommen die (Daten der) L10nPacks her? . . . 60
L10n und das Anführungszeichen bzw. Apostroph . . . 61
Wichtige Klassen . . . 62
Eingabe von L10n-Daten . . . 63
Die Formularengine des Solstice Clients . . . 64
de.ipcon.form . . . 65
Hintergrund . . . 65
Das Formular-Objekt . . . 65
Eigenschaften . . . 65
Auswahl . . . 66
Definition . . . 66
Fehler und Ursachen . . . 67
Compiler-Meldung "Object cannot be null" . . . 67
bi-Tabelle kann nicht erstellt werden (nachdem die Datenbank gedropped und recreated wurde) 67
Compiler-Meldung "Object bla is null but shouldn’t" (sic) . . . 67
Synchronisation der Strukturelemente . . . 68
Das Formular "DateiSystemSync" . . . 69
Volltextsuche . . . 70
Konfiguration im Schema . . . 71
Berücksichtigte Daten . . . 71
Berücksichtigte Entitäten . . . 71
Berücksichtigte Attribute . . . 71
Weitere Einstellungen im Schema . . . 72
analyzed . . . 72
boost . . . 73
Formularelemente . . . 74
Action . . . 75
availableOn . . . 76
enabledOn . . . 76
initialState . . . 76
longDescription . . . 76
onAction . . . 77
BooleanInputComponent . . . 78
Border . . . 79
Button . . . 83
Canvas . . . 85
Chart . . . 86
onClick . . . 89
CheckBox . . . 91
ComboBox . . . 93

DateChooser . . . 95
Editor . . . 97
Element . . . 98
Email . . . 101
Image . . . 104
Label . . . 105
FPanel (abstrakt) . . . 109
Skriptvariablen . . . 109
onAfterSelectValue . . . 110
editableIf . . . 110
visibleIf . . . 110
OnDrop . . . 111
onFocusGained . . . 111
onFocusLost . . . 112
onRefresh . . . 112
onSync . . . 112
FInputPanel (abstrakt) . . . 113
alsoMandatoryIf . . . 113
PDFViewer . . . 114
Popup . . . 116
Scheduler . . . 122
SimpleTimespanChooser . . . 132
Tab . . . 133
TabbedView . . . 136
Table . . . 139
Column . . . 145
headerRenderer und renderer . . . 148
DetailView . . . 148
MultipleChoiceFilterGUI . . . 150
Text . . . 151
TimeSelector . . . 153
ToggleButton . . . 154
Tree . . . 156
Uri . . . 157
View . . . 158
Datenaustausch . . . 161
Import . . . 162
Export . . . 163
Excel . . . 163
Lokale autoritative sowie synchronisierende Instanzen zum Entwickeln aufsetzen . . . 164

MyTISM ist ein leistungsstarkes Framework zur Entwicklung und Verwaltung von Datenbankanwendungen. Es ist plattformunabhängig, objektorientiert, dezentral, multiuserfähig, individuell anpassbar und quelloffen. Mit MyTISM erstellen Sie effizient komplexe Anwendungen, dank einer umfassenden Sammlung von Tools und Funktionen, inklusive GUI und Web-Application-Server. MyTISM wird entwickelt und betreut von der OAshi S.à r.l.

Diese Dokumentation richtet sich an Entwickler und interessierte Anwender, die tief in die Interna und den Aufbau von MyTISM eintauchen möchten.

Voraussetzungen: Um diese Dokumentation optimal nutzen zu können, sollten Sie über gute bis sehr gute Kenntnisse in folgenden Bereichen verfügen:

- Java
- Groovy
- NetRexx
- Relationale und objektorientierte Datenbanken

Idealerweise haben Sie bereits erste Erfahrungen mit der Bedienung von MyTISM gesammelt.

> Diese Dokumentation befindet sich noch im Aufbau. Wir arbeiten kontinuierlich an der Vervollständigung und Verbesserung der Inhalte.

Bei Fragen, Problemen oder Anregungen zu MyTISM oder dieser Dokumentation kontaktieren Sie uns gerne über https://www.mytism.de/#contact.

MyTISM: Das 3-Tier-Framework für Ihre Anwendung

MyTISM ist ein robustes und flexibles 3-Tier-Framework, das die Entwicklung skalierbarer und wartbarer Anwendungen deutlich vereinfacht. Durch die strikte Trennung von Präsentation, Anwendungslogik und Datenhaltung ermöglicht MyTISM effiziente Entwicklungsprozesse und bietet maximale Flexibilität bei der Anpassung an zukünftige Anforderungen.

Die drei Schichten in MyTISM:

1. **Präsentationsschicht (Frontend):** MyTISM stellt Ihnen Werkzeuge und Komponenten zur Verfügung, um Benutzeroberflächen nach Ihren Bedürfnissen zu gestalten.
2. **Anwendungsschicht (Middleware):** In der MyTISM Middleware implementieren Sie die Geschäftslogik Ihrer Anwendung, verarbeiten Daten und greifen auf die Datenbank zu.
3. **Datenbankschicht (Backend):** MyTISM abstrahiert und vereinfacht den Zugriff auf die Datenbank durch die integrierte Object Query Language (OQL). OQL ermöglicht einen objektorientierten, einfachen und performanten Zugriff auf die zugrundeliegende relationale Datenbank.

Die 3-Tier-Architektur sorgt für eine übersichtliche und leicht verständliche Codebasis. Änderungen an einer Schicht haben minimale Auswirkungen auf die anderen Schichten. Diese Modularität erleichtert die Wartung und Aktualisierung der Anwendung und erhöht ihre Skalierbarkeit.

MyTISM bietet Ihnen ein solides Fundament für die Entwicklung Ihrer Anwendung. Durch die konsequente Anwendung des 3-Tier-Modells profitieren Sie von einer strukturierten, flexiblen und zukunftssicheren Architektur.

Vorstellung von MyTISM

Was ist MyTISM?

MyTISM ist ein Java-basiertes Anwendungsframework mit integrierter Datenbankunterstützung. Es besteht aus einem oder mehreren miteinander verbundenen Servern (inkl. PostgreSQL-Datenbank) und Clients, die über das Netzwerk darauf zugreifen. Der Hauptclient, Solstice, bietet eine grafische Benutzeroberfläche mit umfangreichen Konfigurationsmöglichkeiten.

MyTISM ermöglicht die Entwicklung von Webanwendungen, die auf das MyTISM-System zugreifen, und bietet Funktionen zur Erstellung von Berichten, zur Verwaltung von Benutzerrechten, zur Versendung von Benachrichtigungen, zur Reaktion auf Ereignisse mittels seines Alarmsystems und zur Automatisierung von Aufgaben via eigener Dienste.

MyTISM entstand aus der Vision, ein Framework zu schaffen, das die Lücken bestehender Lösungen schließt und eine wirklich integrierte und effiziente Entwicklungsumgebung bietet. MyTISM wurde aus der Notwendigkeit heraus geboren, komplexe Datenbankanwendungen zu vereinfachen und zu beschleunigen. Es ist das Ergebnis jahrzehntelanger Erfahrung und Entwicklung und bietet eine einzigartige Kombination von Funktionen und Flexibilität.

Warum MyTISM?

Eine gute Frage. Warum tut man sich heutzutage noch den Aufwand an, ein Datenbank-Framework und all das Drumherum bis zum Applikationsserver von Grund auf neu zu entwickeln, wenn es die entsprechenden Werkzeuge am Markt doch in Hülle und Fülle bereits gibt? Oder ist es nur der Wunsch eines jeden Programmierers, "sein" Framework zu bauen und zu verwenden?

Die Wahrheit ist: Als ich mir auf der Suche nach den passenden Puzzleteilen die verschiedenen Frameworks angeschaut habe (wohlgemerkt Stand Mitte 2001) musste ich feststellen, daß es zwar immer wieder gute Teile gab, aber nichts, was zusammengepaßt hätte. Osage, XwingML, verschiedene Wrapper für native GUIs, RAD-Tools zum Erstellen von Swing-Code etc; ganz zu schweigen von Standards wie J2EE mit ihren CMP, BMP und weiß Gott was sonst noch so alles. Aber es paßte einfach nicht wirklich ins Bild. Immer, wenn ich einen Prototypen zusammensetzte, blieb ich entweder an einer Lizenz-Ecke hängen oder es hakte einfach technisch an der Möglichkeit der Umsetzung.

Noch lange nicht am Ende dieses Weges angelangt, kann ich nach inzwischen drei Jahren Entwicklung (diese Zeilen entstanden im September 2004) sagen, daß es, und das ist das schönste Lob von allen, immer wieder Programmierer gibt, die einen Blick auf das entstandene Werk werfen und feststellen, daß es sich substantiell von allem unterscheidet, was sie bislang gesehen haben. Das kann natürlich auch an der Unwissenheit oder Unerfahrenheit dieser Programmierer liegen, aber trotzdem interpretiere ich es als ein großes Kompliment für unsere Firma und auch persönlich für mich.

Historie

Den Anfang nahm alles im August 2000 (das Wochenende vom 26.-27., um genau zu sein), als ich anfing, ein neues Projekt für einen Industriekunden zu durchdenken, und mir, nachdem ich schon monatelang um OODBMSe mit einem befreundeten Programmierer diskutiert hatte, der Paradigmenbruch eines Java-Programmierers, der Datenbankanwendungen schreibt, wirklich bewußt wurde.

Irgendwie ist es fast wie Autofahren: Am Anfang freut man sich, daß man das Vehikel wenigstens schadfrei über die Straßen bewegen kann und freut sich, heil anzukommen. Doch je mehr Routine man erlangt, umso mehr fallen Unzulänglichkeiten wie schlechtes Fahrwerk, Sitze, leistungsschwache Motoren oder auch einfach das unbehende und unvorausschauende Treiben anderer Verkehrsteilnehmer auf - nicht, daß sie vorher nicht dagewesen sind, man war möglicherweise sogar ein Teil davon. Aber mit der Routine kommt auch der Drang, besser zu werden, die Prozesse zu optimieren, schneller voran zu kommen, die Resourcen sinnvoller zu nutzen.

So ist man am Anfang froh, überhaupt aus einer DBMS irgendwie Daten zu bekommen, bestaunte jede abgesetzte Query und war froh, die Daten an Ort und Stelle irgendwie laden und wieder zurückspeichern zu können. Auch war es anfangs keine Mühe, sondern eher fast magisch, an dem Schema der SQL-Datenbank herumzuschrauben und Trigger zu setzen, kleine SQL-Scripts zu schreiben etc. Alles war ein großer Spaß und ich war stolz, eine SQL-Datenbank "zu beherrschen". Wie naiv man doch manchmal ist…

Aber mit der Routine kam auch die Redundanz. Schon wieder eine Tabelle ändern, die Query erweitern um eine Spalte, ein bißchen Javacode anpassen, in der GUI das Feld dazubauen, den Serializer erweitern und so weiter. Alles Jobs, um die man sich als Programmierer nicht gerade reißt. Zumindest ich reiße mich nicht darum. Dazu kam, daß man ständig das Verhalten der Anwendung teilweise in der Datenbank, teilweise in der Persistenz und zu guter Letzt auch noch in der GUI bestimmte. Wo wurde nochmal diese zehnstellige Zahl abgefragt? Welche Klasse prüft die Artikelnummer? Die Arbeit mit Datenbankanwendungen wurde schnell zu einer leidigen Pflichtübung und hatte mit Design nur noch wenig zu tun. Ich drückte mich um jedes geänderte Feld, um jede veränderte Definition, weil sie sich mehr oder weniger durch den gesamten Code zog und nach jedem weiteren Programmteil noch aufwändiger wurde. Mal abgesehen davon, daß der Wechsel der SQL-Servers ein reiner Albtraum gewesen wäre, aber dazu kam es Gott sei Dank nie.

Die erste Verbesserung kam dann in jenem August 2000, als ich konkret daran ging, die Anwendung dieses Kunden zu entwerfen und nach einigen Tests mit Osage und anderen ORM’s, deren Namen ich heute nicht mehr weiß, einen Test mit der Castor-API machte. Dieser Test wurde dann schnell zum Prototypen und bald hatte ich ein Problem weniger: Der Persistenz-Code war jetzt an einer Stelle zu finden und viele Aspekte der Business-Logik waren nun in Objekten, die persistiert wurden, gekapselt. Außerdem konnte man mit einem Subset vom OQL Abfragen aus objektorientierter Sicht erstellen, die die API in SQL-Queries abhängig vom verwendeten SQL-Server verwandelte. Sie schirmte einen sozusagen vom SQL-Code völlig ab - eine völlig faszinierende Sache aus Sicht des Programmierers.

Leider war der Castor noch nicht fertig (er ist es heute noch nicht), und seine Unzulänglichkeiten zwangen mich zeitweise wochenlang in die Fehlersuche und haben mich wahrscheinlich um Jahre

altern lassen. Aber er ließ Dinge geschehen, die für mich magisch waren und deren Mechanismen ich im Detail erst viel später verstand. Das einzige, an dem ich immer noch basteln musste, waren die Mapping-Files, die eigentlichen BO-Klassen und die Datenbank an sich. Wenn diese drei ordentlich synchron gepflegt wurden, lief alles reibungslos. Wenn nicht, dann…

Dieses erste Projekt mit einem ORM war ein voller Erfolg; es lief vom ersten Tag fast störungsfrei und bewältigte eine Aufgabe, die recht komplex viele Datenströme zur rechten Zeit zum richtigen Ort schaffen musste und als Schalt- und Regelzentrale eine anspruchsvolle und verantwortungsvolle Stellung in der Firma hatte. Einige der Probleme, die vorher bestanden und die durch eine vorherige Lösung bereits längst erledigt sein sollten, wurden nun endlich erledigt. Einige Folgeaufträge stand das nächste neue Projekt an, ein Vertriebssystem.

Ein altes chinesisches Sprichwort sagt: Für einen Mann mit einem Hammer sieht alles wie ein Nagel aus. Wie wahr. Da hatten wir nun dieses Projekt mit dem Castor ORM und einen Kunden, dessen zugegeben etwas hoch gesteckte Ziel so nicht kauffertig zu erwerben war. Was lag näher, als auch ihm eine individuelle Lösung zu bauen - 60% davon sind ja schon fertig. Nun, zumindest schien das so, und nach einer kurzen Preisverhandlung sollte 3-4 Monate später das Programm fertig sein. Das war im März 2001. Im August 2001 hatte ich eine eher wackelige Fassade dessen, was der Kunde eigentlich wollte (wobei die Frage im Raum steht, welcher Kunde schon am Anfang eines solchen Projekts weiß, was er wirklich will). Synchronisation war noch in weiter Ferne, das Transportprotokoll RMI und die Geschwindigkeit insgesamt lausig. Aber mit wenigen Testdaten blieb alles im RAM und so fiel es nicht weiter auf, daß das eigentliche Produkt, so wie ich es im Kopf hatte, noch viel Arbeit kosten würde.

So wurde zunächst die Formularengine grob zusammengestoppelt und die damals recht junge JNLP-Spezifikation zur Installation der Clients fertiggestellt. Eine erste Version des Schema-Generators erlaubte im Compile-Zyklus eine Anpassung der Datenbank, generierte Sourcecode und erstellte statische Mapping-Files für den späteren Server-Start. Das Projektverzeichnis war ein einziger Wust von kleinen Dateien, in denen kleine Informationsbröckchen lagen, die Auskunft über die Konfiguration einzelner Teile der Software gab. Ein Apache auf der gleichen Maschine lieferte eine kleine Website mit den JNLP-Deskriptoren samt jars aus. Somit war ein zumindest installationsfähiger Client samt Server zusammengestellt, der die anfänglichen Anforderungen des Kunden, der langsam aber sicher auch etwas ungeduldig wurde, bediente (November 2001).

Zur gleichen Zeit hatte ich einen Kunden, der ein altes PHP-Framework von uns um Mehrsprachigkeit und Skins erweitern wollte. Mir war klar, daß in diesem Code-Moloch eigentlich ein Neubau die einzige mögliche Strategie der Weiterentwicklung lag. Aber warum jetzt ein PHP-Framework bauen, wenn gerade ein Java-ORM Framework entstand? Die Idee zu Equinox entstand und wurde von mir bereits im Oktober 2001 gegenüber diesem Kunden über den grünen Klee gelobt. Eigentlich hatte dieser Programmteil als erster seinen Namen; dieser Kunde brauchte einen Namen für ein Produkt, also nannte Thorsten es Equinox.

Die Formularengine bekam alsbald eine Script-Schnittstelle, der Schema-Generator wurde zerteilt und erzeugte im Compile-Stage nur noch den Quellcode der persistenten Klassen und der Server brauchte für die Initialisierung nur noch eine ini und die Log-Konfiguration. Daran hat sich bis heute auch nicht mehr viel verändert, wenngleich der Inhalt der ini inzwischen leicht gewachsen ist. Außerdem entstand in den Weihnachstagen eine erste Version des Synchronisationsmechanismus, der auf Basis der Zeitstempel an den Objekten funktionierte, der sogenannte Statesync. Er funktionierte mit einer nicht allzu großen Menge an Objekten gemessen

an der Komplexität erstaunlich gut. Aber er tat dies weder vollständig noch in einer annehmbaren Geschwindigkeit. Ich hatte die Geschwindigkeit des Gesamtsystems einfach grob überschätzt. Außerdem brauchte er Unmengen an RAM. Eine Alternative musste her, und die war nur im Logsync zu finden - diese Variante ist heute die einzig Mögliche in MyTISM.

Im Lauf der ersten Monate von 2002, inzwischen arbeiteten drei Programmierer aktiv an der Entwicklung von MyTISM, wie es inzwischen hieß, wurden das Logging verfeinert. Erst jetzt wurde der Float-Datentyp in MyTISM eingebaut. Die Formularengine konnte inzwischen Scripts nach allen Regeln der Kunst, ein BX-Objekt (eine nicht persistente Entität, die jedoch per Schema zugreifbar und damit funktionell identisch mit anderen BOs) wurde eingeführt. Ende April gab es eine erste Version der GUI des Hotelvermittlers, der zweite Kunde für MyTISM überhaupt. Anfang Mai wurden dann die ersten wirklich lauffähigen Versionen von Equinox gesichtet; der Equinox-Kunde wollte Juni 2002 live gehen mit der neuen Website, aber leider hatte er durch viele Updates an seine alten Site derartig Resourcen gebunden, daß es uns nicht möglich war, mit der nötigen Intensität an Equinox weiterzubauen. Außerdem waren der Designer und der ERP-Supplier nicht unbedingt unseren Anforderungen gewachsen, so daß wir viele Arbeiten, die eigentlich nicht unser Job waren, mit erledigen mussten. Aber alles in allem ging die Entwicklung von Equinox voran, und die ersten Seiten krabbelten aus dem Web, frisch aus MyTISM, in XML gewandelt, per XSLT transformiert und dann ausgegeben…

Die Logschreiberei war im April soweit beendet, daß ein weiterer Mechanismus gebaut werden konnte: der Export-Handler, der auf Basis der Logs Daten an Fremdsysteme ausliefern konnte - per Timetravel sogar mit echten "Snapshots" der Daten zum jeweiligen Zeitpunkt.

Zukunft?

MyTISM hat sich seit seiner Entstehung zu einem robusten und flexiblen Framework entwickelt, das sich in verschiedenen Anwendungsbereichen bewährt hat. Die Zukunft von MyTISM sieht vielversprechend aus und bietet spannende Möglichkeiten für weitere Innovationen und Verbesserungen.

To be continued…

Schema

Das Schema ist das Herzstück jeder MyTISM-Anwendung. Es definiert die Struktur der Datenobjekte und bildet die Grundlage für die gesamte Anwendungslogik.

Funktionsweise

- **Datenmodellierung:** Im Schema definieren Sie die Entitäten (z. B. Kunde, Produkt, Auftrag) und deren Attribute (z. B. Name, Preis, Datum) sowie die Beziehungen zwischen den Entitäten.
- **Codegenerierung:** MyTISM generiert automatisch den Java-Quellcode für die benötigten Klassen basierend auf Ihrem Schema.
- **Datenbankintegration:** Die Datenbanktabellen werden anhand des Schemas angelegt und verwaltet.
- **UI-Generierung:** Für Solstice (die MyTISM-GUI) werden automatisch Formulare, Schablonen und Lesezeichen generiert, die auf dem Schema basieren.

Vorteile

- **Effizienz:** Schnelle und einfache Datenmodellierung ohne manuelle Codeerstellung.
- **Konsistenz:** Das Schema stellt sicher, dass Datenstruktur und Anwendungslogik synchron bleiben.
- **Flexibilität:** Änderungen am Schema werden automatisch in Code, Datenbank und UI übernommen.

Die Schema-Definition erfolgt in der deklarativen Sprache `XML`. Eine detaillierte Beschreibung der Syntax und der verfügbaren Elemente finden Sie im folgenden Kapitel "Schema-Definition".

Schema-Definition

Das `<Schema>`-Element definiert das Schema für Ihr MyTISM-System. Es enthält alle Informationen über die Entitäten, Attribute, Beziehungen und andere Elemente, die Ihr Datenmodell ausmachen.

Einleitender Tag: `<Schema BEFEHLE>`

Schliessender Tag: `</Schema>`

Attribute

Attribut	Beschreibung	Beispiel
`version`	Gibt die Version des Schemas an. Enthält Platzhalter wie `@BUILT@` (wird beim Kompilieren durch die Versionsnummer ersetzt) und `@ProjectName@`.	`<Schema version="@ProjectName@ Schema built @BUILT@">`
`defaultPackage`	Definiert das Basis-Package für Entitäten und Attribute. Ermöglicht die Vermeidung von redundanten Package-Angaben bei `extends` und `type` Befehlen.	`<Schema version="@ProjectName@ Schema built @BUILT@" defaultPackage="@BOPACK@">`
`defaultFolder`	Definiert den Standardordner im Navigationsbaum für die im Schema definierten Entitäten.	`<Schema version="@ProjectName@ Schema built @BUILT@" defaultPackage="@BOPACK@" defaultFolder="Stammdaten/Konten">`

Unter-Elemente des Schema-Elements

Das `<Schema>`-Element kann die folgenden Unter-Elemente enthalten:

- **`Entity`:** Definiert eine Entität, die ein Datenobjekt innerhalb des MyTISM-Systems repräsentiert (z. B. Kunde, Produkt, Auftrag).
  - Entitäten besitzen Attribute und können Beziehungen zu anderen Entitäten haben.
- **`Include`:** Ermöglicht das Einbinden externer Schema-Definitionen aus anderen Dateien, um das Schema modular zu organisieren.
- **`ModuleProvider`:** Definiert einen Anbieter für Module, die zusätzliche Funktionen oder Erweiterungen für MyTISM bereitstellen.
- **`ModuleIntegrator`:** Definiert einen Integrator für Module, der die Integration von Modulen in das MyTISM-System übernimmt.
- **`Module`:** Definiert ein spezifisches Modul, das im MyTISM-System verwendet werden soll.
  - Module werden von ModuleProvidern bereitgestellt.
- **`Generator`:** Definiert einen Generator, der Code, Datenbankstrukturen oder andere Artefakte

basierend auf dem Schema generiert.

- `Folder`: Definiert einen Ordner im Navigationsbaum der MyTISM-Benutzeroberfläche, um Entitäten und andere Elemente zu organisieren.
- `Type`: Definiert einen benutzerdefinierten Datentyp, der in Attributen verwendet werden kann.
- `Interface`: Definiert ein Interface, das von Entitäten implementiert werden kann.
- `GDPR*`: Definiert GDPR-relevante Elemente, die Informationen zur Einhaltung der Datenschutz-Grundverordnung (DSGVO) enthalten.
  - `GDPRDataCategory`: Definiert eine Kategorie von personenbezogenen Daten (z. B. Vertragsdaten, Gesundheitsdaten).
  - `GDPRBusinessInterest`: Definiert ein berechtigtes Interesse des Unternehmens an der Verarbeitung personenbezogener Daten.
  - `GDPRProcessingPurpose`: Definiert den Zweck der Verarbeitung personenbezogener Daten (z. B. Vertragserfüllung, Werbung).
  - `GDPRProcessingLegalBasis`: Definiert die Rechtsgrundlage für die Verarbeitung personenbezogener Daten (z. B. Einwilligung, Vertragserfüllung).
  - `GDPRLaw`: Definiert ein relevantes Gesetz im Zusammenhang mit der DSGVO (z. B. Artikel 6 DSGVO).
  - `GDPRRetentionPurpose`: Definiert den Zweck der Aufbewahrung personenbezogener Daten.

Include

Mit dem `<Include>`-Element können Sie, ähnlich wie in Programmiersprachen, eine bestehende Schema-Definition aus einer anderen Datei einbinden. Dies ermöglicht die modulare Organisation des Schemas.

Tag: `<Include BEFEHLE />`

Attribut	Beschreibung	Beispiel
`file`	Name der einzubindenden Datei.	`<Include file="/de/ipcon/schema/schema-core.xml">`
`child`	Gibt an, ob das eingebundene Schema ein Kindschema ist.	`<Include file="/de/ipcon/schema/schema-core.xml" child="true">`

Folder

Legt einen Ordner im Navigationsbaum an, in dem die automatisch generierten Formulare und anderen Strukturelemente für die nachfolgend definierten Entitäten einsortiert werden.

Tag: `<Folder Foldername />`

Attribut	Beschreibung	Beispiel
`path`	Name des Ordners.	`<Folder path="Quertabellen">`

ModuleProvider

Definiert einen ModuleProvider.

Attribut	Beschreibung	Beispiel
`name`	Name des ModuleProviders.	`<ModuleProvider name="oashi" path="/com/oashi"/>`
`path`	Pfad des ModuleProviders.	`<ModuleProvider name="oashi" path="/com/oashi"/>`

ModuleIntegrator

Definiert einen ModuleIntegrator.

Attribut	Beschreibung	Beispiel
`class`	Name der Klasse.	`<ModuleIntegrator class="de.ipcon.schema.ModuleIntegrator"/>`

Module

Definiert ein Modul.

Attribut	Beschreibung	Beispiel
`name`	Name des Moduls.	`<Module name="core" provider="oashi">`
`provider`	Name des ModuleProviders.	`<Module name="core" provider="oashi"/>`

Generator

Definiert einen Generator.

Attribut	Beschreibung	Beispiel
`class`	Name der Klasse.	`<Generator class="de.ipcon.schema.generators.PersistenceCode"/>`
`scope`	Scope des Generators. Alle Werte außer "local" werden als globaler Scope interpretiert.	`<Generator class="de.ipcon.schema.generators.PersistenceCode" scope="global"/>`

Type

Definiert einen benutzerdefinierten Datentyp.

Attribut	Beschreibung	Beispiel
`name`	Name des Typs.	`<Type name="Prozent" class="java.math.BigDecimal" extends="Decimal"/>`
`class`	Name der Klasse.	`<Type name="Prozent" class="java.math.BigDecimal" extends="Decimal"/>`
`extends`	Basistyp.	`<Type name="Prozent" class="java.math.BigDecimal" extends="Decimal"/>`

Interface

Definiert ein Interface.

Attribut	Beschreibung	Beispiel
`name`	Name des Interfaces.	`<Interface name="GDPRRelevantI"/>`

GDPR*

Definiert GDPR-relevante Elemente.

GDPRDataCategory

Definiert eine GDPR-Datenkategorie.

Attribut	Beschreibung	Beispiel
`id`	ID der Datenkategorie.	`<GDPRDataCategory id="ContractData">`
`title`	Titel der Datenkategorie (optional).	`<GDPRDataCategory id="ContractData" title="Contract Data">`

GDPRBusinessInterest

Definiert ein GDPR-Geschäftsinteresse.

Attribut	Beschreibung	Beispiel
`id`	ID des Geschäftsinteresses.	`<GDPRBusinessInterest id="LegalCompliance">`

GDPRProcessingPurpose

Definiert einen GDPR-Verarbeitungszweck.

Attribut	Beschreibung	Beispiel
`id`	ID des Verarbeitungszwecks.	`<GDPRProcessingPurpose id="ContractualPerformance">`

GDPRProcessingLegalBasis

Definiert eine GDPR-Rechtsgrundlage.

Attribut	Beschreibung	Beispiel
`id`	ID der Rechtsgrundlage.	`<GDPRProcessingLegalBasis id="Consent">`

GDPRLaw

Definiert ein GDPR-Gesetz.

Attribut	Beschreibung	Beispiel
`id`	ID des Gesetzes.	`<GDPRLaw id="gdpr_art_6_para_1_lit_b" paragraph="Art. 6 para. 1 lit. b GDPR">`
`paragraph`	Paragraph des Gesetzes (z. B. "Art. 6 para. 1 lit. b GDPR").	`<GDPRLaw id="gdpr_art_6_para_1_lit_b" paragraph="Art. 6 para. 1 lit. b GDPR">`
`country`	ISO-Code des Landes.	`<GDPRLaw id="gdpr_art_6_para_1_lit_b" paragraph="Art. 6 para. 1 lit. b GDPR" country="DE">`
`url`	URL zum Gesetzestext (optional).	`<GDPRLaw id="gdpr_art_6_para_1_lit_b" paragraph="Art. 6 para. 1 lit. b GDPR" country="DE" url="https://www.gesetze-im-internet.de/dsgvo_2016/art_6.html">`

GDPRRetentionPurpose

Definiert einen GDPR-Aufbewahrungszweck.

Attribut	Beschreibung	Beispiel
`id`	ID des Aufbewahrungszwecks.	`<GDPRRetentionPurpose id="LegalCompliance">`

Entity

Eine Entity repräsentiert ein Datenobjekt innerhalb Ihres MyTISM-Systems, z. B. einen Kunden, ein Produkt oder einen Auftrag. Im Schema definieren Sie die Struktur einer Entity, indem Sie ihre Attribute festlegen und Beziehungen zu anderen Entitäten herstellen.

Einleitender Tag: `<Entity BEFEHLE>`

Schliessender Tag: `</Entity>`

Attribut	Beschreibung	Beispiel
`name`	Name der Entität.	`<Entity name="Lieferant">`
`plural`	Plural-Bezeichnung der Entität.	`<Entity name="Lieferant" plural="Lieferanten">`
`extends`	Gibt die Basisklasse an, von der die Entität erbt.	`<Entity name="Lieferant" plural="Lieferanten" extends="XBuchungsKonto">`
`abstract`	Gibt an, ob die Entität abstrakt ist. Abstrakte Entitäten können nicht direkt instanziiert werden und dienen als Basisklassen für andere Entitäten. Standardwert: `false`.	`<Entity name="XBuchungsKonto" plural="XBuchungsKonten" extends="CBO" abstract="true">`
`noAbstractWarning`	Unterdrückt die Warnung, dass die Entität abstrakt ist, und erlaubt (eingeschränkt) die Instanziierung der Entität. Dies ist für technische Entitäten nützlich, die programmatisch erstellt, aber nicht über die MyTISM-GUI bearbeitet werden sollen.	`<Entity name="XBuchungsKonto" plural="XBuchungsKonten" extends="CBO" abstract="true" noAbstractWarning="true">`
`ignoreReverseRelations`	Gibt an, ob Rückwärtsbeziehungen für eine Entität ignoriert werden sollen.	`<Entity name="Lieferant" plural="Lieferanten" extends="XBuchungsKonto" ignoreReverseRelations="true">`
`discriminator`	Setzt einen benutzerdefinierten Diskriminator (d.h. die BOT Id) für die Entität.	`<Entity name="Lieferant" plural="Lieferanten" extends="XBuchungsKonto" discriminator="4711">`
`suid`	Legt eine speziellen suid (SerialVersionUID, serial version unique ID, Klassenversionskennung) fest.	`<Entity name="Lieferant" plural="Lieferanten" extends="XBuchungsKonto" suid="4527893412646488711">`
`package`	Gibt ein abweichendes Package für die Entität an.	`package="de.blues.bo"`

Unter-Element "gdpr" von Entity

Der `<gdpr>`-Knoten ist ein optionaler Unterknoten innerhalb einer Entity, der alle DSGVO-relevanten Informationen für diese Entity bündelt.

Struktur des `<gdpr>`-Knotens:

```
<gdpr dataCategory="Contract Data" retentionStartDatePath="Vertragsende">
  <affectedPerson path="Kunde.AbstraktePerson"/>
  <affectedPerson path="Adressat"/>
  <retentionVeto path="BuchungsKonto.AbstraktePerson"/>
</gdpr>
```

Attribut/Element	Beschreibung	Beispiel
`dataCategory` (Attribut)	Gibt die Datenkategorie der Entity an (z. B. "Contract Data", "Invoice Data", "Health Data"). Dieses Attribut ist **erforderlich**.	`<gdpr dataCategory="Contract Data">`
`retentionStartDatePath` (Attribut, optional)	Enthält den Pfad zu einem Datumswert, der den Start der Aufbewahrungsfrist definiert (z. B. Vertragsende). Nur anwendbar, wenn der data retention start der dataCategorie den Wert "custom" enthält.	`<gdpr retentionStartDatePath="Vertragsende">`
`affectedPerson` (Element, optional), Attribut `path`	Enthält den Pfad zu einer von dem Datensatz betroffenen Person. Mehrere `<affectedPerson>`-Elemente sind möglich, um mehrere betroffene Personen anzugeben.	`<gdpr dataCategory="Contract Data"> <affectedPerson path="Kunde.AbstraktePerson"/> <affectedPerson path="Adressat"/> </gdpr>`
`retentionVeto` (Element, optional), Attribut `path`	Gibt den Pfad zu einem Datensatz an, der ein Veto gegen die Löschung dieser Entity einlegen kann (z. B. eine Archivierungsanforderung). Mehrere `<retentionVeto>` -Elemente sind möglich.	`<gdpr dataCategory="Contract Data"> <retentionVeto path="BuchungsKonto.AbstraktePerson"/> </gdpr>`

Unter-Element "ui" von Entity

Das `<ui>`-Element innerhalb einer Entity-Definition enthält Attribute, die das Verhalten und die Darstellung der Entität in der Benutzeroberfläche (UI) beeinflussen.

Attribut	Beschreibung	Beispiel
`description`	Beschreibung der Entität. Definiert die Ausgabe der `describe()`-Methode der Entität im CBOFormat.	`<ui description="Nachname(', 'Vorname)"/>`
`loadImmediate`	Gibt an, ob eine Übersicht (Liste) der Objekte der Entität direkt in einem geöffneten Lesezeichen (FTable) angezeigt werden soll. Aus Performance-Gründen nur sinnvoll bei Entitäten mit einer überschaubaren Anzahl von Objekten. Standardwert: `false`.	`<ui loadImmediate="false"/>`
`linkOnly`	Gibt an, ob Objekte nicht im Kontext dieses Objekts angelegt, sondern lediglich mit diesem Objekt verknüpft werden können. Standardwert: `false`.	`<ui linkOnly="false"/>`
`tips`	Zum Angeben von GUI-Tipps, d.h. durch Leerzeichen getrennte `Variable:Wert`-Paare, die das Verhalten der UI konfigurieren.	`<ui tips="triState:false"/>`
`defaultSorting`	Definiert die Standardsortierung für Tabellen, die Objekte dieses Typs anzeigen. Format: `Spaltenname:Sortierrichtung`. Die Reihenfolge der Einträge bestimmt die Sortierreihenfolge (sortLevel). Die Mehrfachsortierung erfordert eine zusätzliche Lizenz.	`<ui defaultSorting="Name:ASC Beschreibung:DESC Position.ASC Eigenschaft:ASC Eigenschaft.Name:ASC">`

Attribut	Beschreibung	Beispiel
`autotrim`	Nur sinnvoll für Entity-Attribute vom Typ "String". Legt fest, ob Whitespace am Anfang und Ende der Eingabe in der GUI automatisch entfernt wird. Standardwert für String-Attribute ist `true`. Mit `autotrim="false"` kann die automatische Entfernung deaktiviert werden. Hat **keine** Auswirkungen auf Werte, die mittels Code gesetzt werden.	`<ui autotrim="false"/>`
`defaultSelectionFilter`	Definiert einen nicht-interaktiven Filter für die Auswahl von Objekten diesen Typs. Der Filter kann als `describe`-Clause oder als Verweis auf ein virtuelles Attribut angegeben werden und wird in Popups und anderen Auswahlfeldern verwendet. Um gar keine Ergebnisse anzubieten, muss man `FALSE` oder `false` zurückgeben. Um keine Filterung zu erreichen, d.h. es werden alle Datensätze angeboten, kann man entweder `TRUE`, `true`, einen leeren String, einen String mit nur Whitespaces oder aber einfach nur null zurückgeben, wobei null hier die präferierte Variante ist, da dadurch keine sowieso ignorierte clause hinzugefügt wird.	`<ui defaultSelectionFilter="'Inaktiv = NULL OR NOT Inaktiv'"/>` oder `<ui defaultSelectionFilter="GeschaeftsbereichFilter"

Unter-Element "lookup" von Entity

Das `<lookup>`-Element innerhalb einer Entity-Definition ermöglicht die Konfiguration der Standardsuche für die Entität, wenn sie in einem Popup-Feld in der Benutzeroberfläche referenziert wird.

Attribut	Beschreibung	Beispiel
`defaultProperty`	Gibt an, in welchem Attribut der Entität bei der Suche im Popup-Feld standardmäßig gesucht werden soll. Wenn der Benutzer einen Suchbegriff eingibt, werden die Objekte der Entität durchsucht, deren Wert im angegebenen Attribut dem Suchbegriff entspricht.	`<lookup defaultProperty="Name"/>`
`defaultSubstring`	Gibt an, ob die Suche nach einer exakten Übereinstimmung oder einem Teilstring suchen soll. * `true`: Es wird nach einem Teilstring gesucht (Standardwert). * `false`: Es wird nach einer exakten Übereinstimmung gesucht.	`<lookup defaultSubstring="true"/>`
`defaultCaseSensitive`	Gibt an, ob die Suche die Groß-/Kleinschreibung beachten soll. * `true`: Die Groß-/Kleinschreibung wird beachtet. * `false`: Die Groß-/Kleinschreibung wird nicht beachtet (Standardwert).	`<lookup defaultCaseSensitive="true"/>`

Beispiel:

```
<Entity name="Person">
  <lookup defaultProperty="Nachname" defaultSubstring="true"
defaultCaseSensitive="false"/>
  <attr name="Vorname" type="String"/>
  <attr name="Nachname" type="String"/>
</Entity>
```

In diesem Beispiel wird festgelegt, dass bei der Suche nach Personen in einem Popup-Feld standardmäßig im Attribut `Nachname` gesucht wird. Die Suche erfolgt dabei ohne Berücksichtigung der Groß-/Kleinschreibung und akzeptiert auch Teilstrings.

Hinweis: Diese Einstellungen können für einzelne Attribute durch das `<lookup>`-Element innerhalb des Attributs überschrieben werden.

Unter-Element "code" von Entity

Das `<code/>`-Element innerhalb einer Entity-Definition enthält Attribute, die die Codegenerierung für die Entität beeinflussen.

Attribut	Beschreibung	Beispiel
`package`	Gibt ein abweichendes Package für die Entität an.	`<code package="de.blues.bo"/>`
`generateAs`	Gibt den Namen der generierten Basisklasse an. Es muss dann manuell eine Klasse erstellt werden, die von dieser Basisklasse erbt.	`<code generateAs="PersonBase"/>`
`custom`	Gibt an, ob die Basisklasse mit dem Suffix "Base" generiert werden soll. Es muss dann manuell eine Klasse erstellt werden, die von dieser Basisklasse erbt.	`<code custom="true"/>`
`generate`	Gibt an, ob für die Entität Sourcecode generiert werden soll. Standardwert: `true`.	`<code generate="false">`
`dependents`	Gibt an, welche Klassen von dieser Entität abhängen und bei Änderungen neu generiert werden müssen.	`<code dependents="GeschaeftsVorfallSchemaAspects"/>`

Unter-Element "db" von Entity

Das `<db>`-Element innerhalb einer Entity-Definition enthält Attribute, die das Datenbankverhalten der Entität beeinflussen.

Attribut	Beschreibung	Beispiel
`persistent`	Gibt an, ob die Objekte der Entität in der Datenbank persistiert werden sollen. Standardwert: `true`.	`<db persistent="false"/>`
`name`	Gibt einen optionalen, abweichenden Tabellennamen für die Entität an. Der Name darf maximal 63 Zeichen lang sein und keine Unterstriche enthalten.	`<db name="MyShorterTableNameNoUnderscore"/>`

Attribut	Beschreibung	Beispiel
`streamResource`	Gibt an, ob mit Instanzen dieses Typs normalerweise ein BLOB (Binary Large Object) verknüpft sein kann. Wird verwendet, um zu entscheiden, ob beim Synchronisieren von Objekten ins Dateisystem auch das BLOB exportiert werden soll. Standardwert: `false`.	`<db streamResource="true"/>`
`noStreamResourceHistory`	Legt fest, dass keine Sicherheitskopien der BLOBs beim Ersetzen im Dateisystem des Servers angelegt werden sollen. Standardwert: `false`.	`<db noStreamResourceHistory="true" />`
`forbidDirectChanges`	Gibt an, ob direkte Änderungen an Objekten dieses Typs nur vom Server vorgenommen werden dürfen. Solche Objekte werden nicht zwischen MyTISM-Nodes synchronisiert. Standardwert: `false`.	`<db forbidDirectChanges="true"/>`

Unter-Element "report" von Entity

Das `<report>`-Element innerhalb einer Entity-Definition enthält Attribute, die die Generierung von Reports für die Entität beeinflussen.

Attribut	Beschreibung	Beispiel
`title`	Titel für automatisch generierte Reports.	`<report title="Analyse"/>`
`orientation`	Seitenausrichtung für automatisch generierte Reports ("Portrait" oder "Landscape").	`<report orientation="Portrait"/>`
`fontSizeNormal`	Standard-Schriftgröße für automatisch generierte Reports. Standardwert: 9.	`<report fontSizeNormal="10"/>`
`fontSizeBig`	Große Schriftgröße für automatisch generierte Reports. Standardwert: 16.	`<report fontSizeBig="14"/>`

Unter-Element "export" von Entity

Das `<export>`-Element innerhalb einer Entity-Definition ermöglicht die Konfiguration des Exports von Entitätsdaten. Diese Funktionalität befindet sich noch in der Entwicklung und soll zukünftig

den Struktur-Sync, den Initialdaten-Import und manuelle Exports in Fremdsysteme ersetzen.

Zukünftig soll es auch möglich sein, Objekte über die Zwischenablage zwischen verschiedenen MyTISM-Systemen zu kopieren, sofern die entsprechenden Entitäten in beiden Systemen existieren.

Attribut	Beschreibung	Beispiel
`name`	Name des Exports. **(Pflichtangabe)**	`<export name="Initialdaten"/>`
`primaryKey`	Primärer Schlüssel für den Export. Wird für Referenzen aus anderen Exporten verwendet. **(Pflichtangabe)**	`<export primaryKey="ISOCode"/>`
`mode`	Modus für den Export: * `SINGLE`: Für jedes Objekt wird eine separate Datei erstellt. * `LIST`: Alle Objekte werden in eine gemeinsame Datei geschrieben. **(Pflichtangabe)**	`<export mode="SINGLE"/>`

Hinweise:

- Die genaue Funktionsweise und die Verwendung des `<export>`-Elements werden in der zukünftigen Dokumentation detailliert beschrieben.
- Derzeit ist die Funktionalität noch nicht vollständig implementiert.

Beispiel für eine Entity-Definition

```
<Entity name="Person" extends="CBO" plural="Personen" folder="Kontakte">
  <code custom="true"/>
  <ui description="Nachname(', 'Vorname)"/>
</Entity>
```

Diese Definition erstellt eine Entität namens "Person" mit den folgenden Eigenschaften:

- Sie erbt von der Klasse `CBO`.
- Der Plural ist "Personen".
- Sie wird im Ordner "Kontakte" im Navigationsbaum angezeigt.
- Es wird eine benutzerdefinierte Klasse generiert (`custom="true"`).
- Die `describe()`-Methode verwendet als Format "Nachname(', 'Vorname)".

Attribut

Attribute beschreiben die Eigenschaften einer Entität. Sie werden innerhalb einer Entity-Definition mit den Tags `<attr>`, `<vattr>` (für virtuelle Attribute) und `<npattr>` (für nicht-persistente Attribute) definiert.

Tag für Attribute: `<attr BEFEHLE />`

Tag für virtuelle Attribute: `<vattr BEFEHLE />`

Tag für nicht-persistente Attribute: `<npattr BEFEHLE />`

Attribut	Beschreibung	Beispiel
`name`	Name des Attributs.	`name="Adressen"`
`backName`	Definiert den Namen des Attributs auf der "anderen Seite" einer Beziehung.	`backName="MoeglicheReports"`
`singular`	Singularform des Attributnamens (falls abweichend von `name`).	`singular="Adresse"`
`type`	Datentyp des Attributs.	`type="Kontakt"`
`displayFormat`	Standard-Anzeigeformat des Attributs.	`displayFormat="Name"`
`relation`	Beziehung des Attributs zu einer anderen Entität ("n-1", "1-n", "n-m").	`relation="1-n"`
`dependent`	Gibt an, ob abhängige Objekte beim Löschen des übergeordneten Objekts ebenfalls gelöscht werden. Standardwert: `false`.	`dependent="true"`
`itemProperty`	Ermöglicht die manuelle Sortierung von Objekten in einer 1-n- oder n-m-Beziehung über die UI.	`itemProperty="Position"`
`shared`	Bestimmt, ob die Referenz von der Entität dieses Attributs geteilt wird. Standardwert: `false`.	`shared="true"`
`default`	Definiert einen Standardwert für das Attribut.	`default="#,##0.00"`
`ignoreBackRelation`	Gibt an, ob Methoden für die Rückwärtsbeziehung generiert werden sollen.	`ignoreBackRelation="true"`
`readonly`	Gibt an, ob das Attribut in der UI schreibgeschützt ist. Standardwert: `false`.	`readonly="true"`

Attribut	Beschreibung	Beispiel
`lazy`	Gibt für many-relations an, ob die in Beziehung stehenden Objekte erst bei der Verwendung nachgeladen werden sollen. Standardwert: `true`	`lazy="true"`
`omitOnCopy`	Gibt an, ob das Attribut beim Kopieren eines Objekts dieses Typs ignoriert werden soll.	`omitOnCopy="true"`

Unter-Element "ui" von Attribute

Das `<ui>`-Element innerhalb eines Attributs beeinflusst die Darstellung und das Verhalten des Attributs in der Benutzeroberfläche.

Attribut	Beschreibung	Beispiel
`editMode`	Gibt an, wie das Attribut in der UI bearbeitet werden kann (`linkonly`, `viewonly`, `locked`, `writenew`, `all`).	`<ui editMode="linkonly">`
`createInDetailView`	Gibt an, ob im Formular der Entität Eingabefelder für die Many-Relation erstellt werden sollen. Standardwert: `false`.	`<ui createInDetailView="true">`
`mandatory`	Definiert das Attribut als Pflichtfeld. Derzeit nur teilweise unterstützt (z. B. nicht von Solstice). Standardwert: `false`.	`<ui mandatory="true">`
`tips`	Zum Angeben von GUI-Tipps, d.h. durch Leerzeichen getrennte `Variable:Wert`-Paare, die das Verhalten der UI konfigurieren. Mögliche Werte sind z. B.: `StyledText`, `Area`, `combobox:ATTRIBUTNAME`, `formRecursionDepth:INTEGER` (Standardwert: 3), `createShared`.	`<ui tips="triState:false"/>`
`visible`	Gibt an, ob das Attribut im Formular angezeigt werden soll. Standardwert: `true`.	`<ui visible="false">`
`expectedWidth`	Definiert die erwartete Breite des Attributs im Formular (in Zeichen).	`<ui expectedWidth="10">`

Attribut	Beschreibung	Beispiel
`selectionFilter`	Definiert einen nicht-interaktiven Filter für die Auswahl von Objekten in diesem Attribut. Der Filter kann als `describe`-Clause oder als Verweis auf ein virtuelles Attribut angegeben werden. Um gar keine Ergebnisse anzubieten, muss man in dem virtuellen Attribut `FALSE` oder `false` zurückgeben. Um keine Filterung zu erreichen, d.h. es werden alle Datensätze angeboten, kann man entweder `TRUE`, `true`, einen leeren String, einen String mit nur Whitespaces oder aber einfach nur null in dem virtuellen Attribut zurückgeben, wobei null hier die präferierte Variante ist, da dadurch keine sowieso ignorierte clause hinzugefügt wird.	`<ui selectionFilter="('BOTyp.Id='Bot.Id)"/>` oder `<ui selectionFilter="Geschaeftsbereichfilter" editMode="linkOnly"/>`

Unter-Element "lookup" von Attribute

Das `<lookup>`-Element innerhalb eines Attributs ermöglicht die Konfiguration der Suche nach zugehörigen Objekten, wenn das Attribut als Popup-Feld in der Benutzeroberfläche dargestellt wird.

Attribut	Beschreibung	Beispiel
`property`	Gibt an, in welchem Attribut der referenzierten Entität bei der Suche im Popup-Feld gesucht werden soll. Wenn der Benutzer einen Suchbegriff eingibt, werden die Objekte der referenzierten Entität durchsucht, deren Wert im angegebenen Attribut dem Suchbegriff entspricht.	`<lookup property="Name">`

Attribut	Beschreibung	Beispiel
`substring`	Gibt an, ob die Suche nach einer exakten Übereinstimmung oder einem Teilstring suchen soll. * `true`: Es wird nach einem Teilstring gesucht (Standardwert). * `false`: Es wird nach einer exakten Übereinstimmung gesucht.	`<lookup substring="true">`
`caseSensitive`	Gibt an, ob die Suche die Groß-/Kleinschreibung beachten soll. * `true`: Die Groß-/Kleinschreibung wird beachtet. * `false`: Die Groß-/Kleinschreibung wird nicht beachtet (Standardwert).	`<lookup caseSensitive="true">`

Beispiel:

```
<Entity name="Kunde">
  <attr name="Ansprechpartner" type="Person" relation="n-1">
    <lookup property="Name" substring="true" caseSensitive="false"/>
  </attr>
</Entity>
```

In diesem Beispiel wird das Attribut `Ansprechpartner` der Entität `Kunde` als Popup-Feld dargestellt. Wenn der Benutzer in diesem Feld einen Suchbegriff eingibt, werden alle Personen gesucht, deren Name den Suchbegriff enthält (`substring="true"`). Die Suche erfolgt dabei ohne Berücksichtigung der Groß-/Kleinschreibung (`caseSensitive="false"`).

Unter-Element "report" von Attribute

Das `<report>`-Element innerhalb eines Attributs ermöglicht die Anpassung der Darstellung des Attributs in automatisch generierten Reports.

Attribut	Beschreibung	Beispiel
`visible`	Steuert die Sichtbarkeit des Attributs im Report. * `true`: Das Attribut wird im Report angezeigt (Standardwert). * `false`: Das Attribut wird im Report nicht angezeigt.	`<report visible="false">`

Attribut	Beschreibung	Beispiel
`relativeWidth`	Legt die relative Breite des Feldes für das Attribut im Report fest. Der Standardwert ist 1. Ein höherer Wert vergrößert die Breite des Feldes.	`<report relativeWidth="2">`
`position`	Bestimmt die Position des Attributs im Report. Attribute mit niedrigeren Positionswerten werden zuerst angezeigt.	`<report position="100">`
`sort`	Gibt an, ob die Daten im Report nach diesem Attribut sortiert werden sollen. * `asc`: Aufsteigende Sortierung. * `desc`: Absteigende Sortierung.	`<report sort="asc">`
`manySort`	Ermöglicht die Sortierung nach mehreren Attributen. Die Attributnamen werden als kommaseparierte Liste angegeben, wobei jedem Attributname der Suffix `:A` (aufsteigend) oder `:D` (absteigend) angehängt wird.	`<report manySort="Tid:A, Nummer:D">`
`alias`	Erstellt im Report eine Gruppe mit dem angegebenen Alias, um die Attribute der zugehörigen Entität aus einer Many-Relation anzuzeigen.	

Beispiele:

```
<Entity name="Produkt">
  <attr name="Name" type="String">
    <report relativeWidth="2" position="1" sort="asc"/>
  </attr>
  <attr name="EAN" type="String">
    <report relativeWidth="1" position="2"/>
  </attr>
  <attr name="Preis" type="Decimal">
    <report visible="false"/>
  </attr>
</Entity>
```

In diesem Beispiel wird das Attribut `Name` im Report mit doppelter Breite (`relativeWidth="2"`) an erster Stelle (`position="1"`) angezeigt und die Daten werden alphabetisch nach dem Namen sortiert (`sort="asc"`). Das Attribut `EAN` wird an zweiter Stelle (`position="2"`) angezeigt. Das Attribut `Preis` wird im Report nicht angezeigt (`visible="false"`).

```
<Entity name="Auftrag">
  <attr name="Auftragsnummer" type="String"/>
  <attr name="Positionen" type="Auftragsposition" relation="1-n">
    <report alias="Position"/>
  </attr>
</Entity>

<Entity name="Auftragsposition">
  <attr name="Artikelnummer" type="String"/>
  <attr name="Menge" type="Integer"/>
</Entity><report alias="Position">
```

Dieses Beispiel zeigt die Attribute der Entität Auftragsposition in einer Gruppe namens "Position" an.

Unter-Element "virtual" von Attribute

Das `<virtual>`-Element innerhalb eines Attributs definiert zusätzliche Eigenschaften für virtuelle Attribute.

Attribut	Beschreibung	Beispiel
`aggregate`	Gibt an, ob der Wert des virtuellen Attributs durch eine Aggregatfunktion berechnet werden soll.	`<virtual aggregate="BO.Union:Gruppe.Benutzer">`
`cacheMode`	Gibt an, ob der Wert des virtuellen Attributs versioniert werden soll.	`<virtual cacheMode="VERSIONED">`
`preCachingHook`	Name einer Methode, die auf den Wert des virtuellen Attributs angewendet wird, bevor er im Cache gespeichert wird.	`<virtual preCachingHook="compactBigDecimal">` (mit einer entsprechenden Methode `compactBigDecimal(attribute = String, value = BigDecimal) returns BigDecimal`)

Aggregatfunktionen:

Aggregatfunktionen ermöglichen die einfache Definition von virtuellen Attributen, die berechnete Werte zurückgeben (z. B. die Summe aller Elemente einer Relation).

Beispiele für Aggregatfunktionen:

- `String.sortJoinCommaList`: Verbindet alle Strings einer Liste zu einem kommaseparierten String.
- `BigDecimal.sum`: Summiert alle Zahlen einer Liste.
- `BO.union`: Bildet die Vereinigung mehrerer Objektmengen.
- `Object.firstNonNull`: Gibt das erste nicht-`null`-Element einer Liste zurück.

Unter-Element "np" von Attribute

Das `<np>`-Element innerhalb eines Attributs definiert zusätzliche Eigenschaften für nicht-persistente Attribute.

Attribut	Beschreibung	Beispiel
`calculationAuthority`	Gibt an, wo der Wert des nicht-persistenten Attributs berechnet werden soll ("client" oder "server"). Wenn die Berechnung auf dem Server erfolgt, wird der berechnete Wert an den Client übertragen und dort der bestehende Wert ersetzt. Standardwert: `client`.	`<np calculationAuthority="server"/>`

Unter-Element "db" von Attribute

Das `<db>`-Element innerhalb eines Attributs beeinflusst das Datenbankverhalten des Attributs.

Attribut	Beschreibung	Beispiel
`indexed`	Gibt an, ob die Werte des Attributs für die Datenbank-Volltextsuche indiziert werden sollen. Standardwert: `true`.	`<db indexed="false">`
`unique`	Gibt an, ob die Eindeutigkeit der Werte des Attributs in der Datenbank erzwungen werden soll. Standardwert: `false`.	`<db unique="true">`

Restliche Unter-Elemente von Attribute

Unter-Element	Beschreibung	Beispiel
`backRelation`	Ermöglicht die explizite Definition der Rückwärtsbeziehung, z. B. um einen anderen Namen zu verwenden oder die Beziehung genauer zu konfigurieren.	`<backRelation name="BezugnehmendePosten"/>`

Unter-Element	Beschreibung	Beispiel
`comment`	Ermöglicht das Hinzufügen eines Kommentars zum Attribut im Schema.	`<comment>Dient zur Speicherung von XXX</comment>`

Beispiel für eine Attribut-Definition

```
<attr name="Adressen" singular="Adresse" type="Kontakt" relation="1-n"
            dependent="true" itemProperty="Position">
</attr>
```

Diese Definition erstellt ein Attribut namens "Adressen" mit den folgenden Eigenschaften:

- Der Singular ist "Adresse".
- Der Datentyp ist "Kontakt".
- Es handelt sich um eine 1-n-Beziehung.
- Abhängige Objekte werden beim Löschen des übergeordneten Objekts ebenfalls gelöscht.
- Die Reihenfolge der Objekte wird über das Attribut "Position" bestimmt und kann in der UI manuell angepasst werden.

Vordefinierte Datentypen für Attribute

MyTISM bietet eine Reihe von vordefinierten Datentypen für Attribute. Hier sind zwei Beispiele:

Timespan

Der Datentyp `Timespan` speichert eine **feste** Zeitspanne als Anzahl von Sekunden. Aktuell werden intern Sekunden verwendet, dies sollte aber ggfs. aktualisiert werden, da Millisekunden präziser wären und möglicherweise in Zukunft verwendet werden. Wenn ein Attribut mit `type="Timespan"` definiert wird, ist der entsprechende Java-Typ `Long`.

`Timespan` eignet sich für Anwendungsfälle, in denen die genaue Dauer eines Ereignisses oder Vorgangs gespeichert werden soll.

Duration

Im Gegensatz zu `Timespan` speichert `Duration` eine **variable** Zeitspanne als Kombination von Jahren, Monaten, Tagen, Stunden, Minuten und Sekunden (inklusive Millisekunden). Die genaue Zeitspanne wird in Abhängigkeit von einem Referenzdatum berechnet.

Intern wird der Wert als `javax.xml.datatype.Duration` gespeichert.

Beispiel: `P1Y0M0DT0H0M0S` entspricht 365 Tagen ab dem 1.1.2015, aber 366 Tagen ab dem 1.1.2012 (Schaltjahr).

Wichtig: Die Behandlung von `Duration` im Code ist an einigen Stellen noch nicht optimal und sollte verbessert werden.

> Die Verarbeitung von Duration-Werten im Code ist an einigen Stellen noch nicht optimal. Beispielsweise wird in L10nTimespanFormat.nrx und DurationType.nrx die Duration oftmals in eine feste Zeitspanne umgewandelt, wobei das Epoch-Datum (1. Januar 1970) als Referenzdatum verwendet wird, anstatt ein anderes Referenzdatum zu berücksichtigen. Dies kann zu Ungenauigkeiten oder Fehlern bei der Berechnung von Zeitspannen führen.

Schemapflege / Datenbankupdates

Einige Schemaänderungen können automatisch vom Schemagenerator verarbeitet werden. Für komplexere Änderungen, die nicht automatisch durchgeführt werden können, müssen Datenbank-Update-Skripte verwendet werden.

Liste der durch den UpdateHandler zur Verfügung gestellten Hilfsmethoden

Der `UpdateHandler` bietet Hilfsmethoden für die Durchführung von Datenbankupdates. Hier sind zwei Beispiele:

Name	Parameter	Beschreibung	Beispielaufruf
`checkTableExists`	Tabellenname	Prüft, ob die angegebene Tabelle in der Datenbank existiert.	`checkTableExists('beleg')`
`checkColumnExists`	`table`: Tabellenname, `column`: Spaltenname	Prüft, ob die angegebene Spalte in der angegebenen Tabelle existiert.	`checkColumnExists(table: 'beleg', column: 'belegnr')`

Coredata-Generator

Der Coredata-Generator (`de/ipcon/schema/generators/CoreData.nrx`) füllt die Datenbank mit initialen Daten und Objekten:

1. Füllt die BOT-Liste für alle Entitäten.
2. Legt den Admin-Benutzer und die Admins-Gruppe an.
3. Legt Sammelordner für automatisch generierte Objekte an (werden normalerweise wieder gelöscht, da alle Entitäten explizit einen anderen Ordner angeben sollten).
4. Legt Standard-Druckziele an.
5. Erzeugt Standardformulare, -schablonen und -lesezeichen für jede Entität (außer Schablonen für abstrakte Entitäten und Lesezeichen für nicht-persistente Entitäten).
6. Erzeugt Standard-Reports (Einzel und Liste) für jede Entität.
7. Lädt und erzeugt zusätzliche, vorgebaute Strukturelemente.
8. Löscht obsolete automatisch generierte Strukturelemente und leere Ordner.

Zusätzliche, vorgebaute Strukturelemente

Zusätzlich zu den automatisch generierten Strukturelementen können vorgebaute Formulare, Schablonen und Lesezeichen in die Datenbank eingespielt werden. Diese müssen im Verzeichnis `de/ipcon/db/core/resources` abgelegt werden. Das Format entspricht dem der exportierten Dateien aus der Formularsynchronisation.

Beim Bauen von `MyTISM-Kernel.jar` werden die Dateien im Verzeichnis `de/ipcon/db/core/resources` gesammelt und in einer Liste ("ResourceIndex") im JAR gespeichert. Der Coredata-Generator liest diese Liste und erstellt die entsprechenden Objekte in der Datenbank.

Unterstützte Angaben für vorgebaute Strukturelemente

Vorgebaute Strukturelemente (Formulare, Schablonen und Lesezeichen) werden durch XML-Dateien definiert. Folgende Elemente und Attribute werden unterstützt:

Root-Element:

- **Name:** Gibt den Typ des Strukturelements an (`Formular`, `Schablone` oder `Lesezeichen`).

Attribute des Root-Elements:

- **Name:** Name des Strukturelements.
  - Frei wählbar, aber es wird die Konvention "<Entityname> (Vorgebaut)" oder "<Entityname> (Vorgebaut; <Kommentar>)" empfohlen.
- **ElterPfad:** Pfad des Ordners, in dem das Strukturelement abgelegt werden soll (z. B. `/Ordner1/Ordner2`).
  - Ordner werden bei Bedarf automatisch erstellt.

- **Prioritaet:** Priorität des Strukturelements (höhere Werte bedeuten höhere Priorität).
  - Sollte im Allgemeinen nicht verwendet werden, da automatisch ein Standardwert (-50) vergeben wird.
- **Tid:** "Klartext-Identifier" des Strukturelements.
  - Sollte im Allgemeinen nicht verwendet werden, da automatisch ein konsistenter Wert vergeben wird.

Kind-Elemente des Root-Elements:

- **Beschreibung:** Textuelle Beschreibung des Strukturelements.
- **BOTyp:** Gibt an, für welche Entität das Strukturelement verwendet werden soll (Attribut `Name`).
- **Parameter:** Enthält die detaillierte Definition des Strukturelements, insbesondere für Formulare.
- **Formular:** (Nur für Schablonen) Gibt das Formular an, das zum Bearbeiten neuer Objekte verwendet werden soll (Attribut `Name`).

Nicht unterstützte Elemente/Attribute:

- **Gruppen, Polymorphic:** Diese Elemente/Attribute werden derzeit nicht unterstützt. Alle Strukturelemente werden automatisch der Admins-Gruppe zugewiesen.

Beispiel für ein vorgebautes Formular:

```
<Formular Name="$R{_Benutzer} (Vorgebaut)"
ElterPfad="/Admins/MyTISM/Benutzerverwaltung">
  <Beschreibung>Vorgebautes, aufgeraeumteres Benutzer-Formular, mit Gruppierungen der
Alarm-
    und Benachrichtigungsinfos und einfacherer, halbautomatischer
    Benachrichtigungskonfiguration.</Beschreibung>
  <Parameter>
    <TabbedView tabPlacement="TOP">
      <!-- ... Inhalt ... -->
    </TabbedView>
  </Parameter>
  <BOTyp Name="Benutzer"/>
  <Gruppen>
    <Gruppe Name="RG_Solstice_Login"/>
  </Gruppen>
</Formular>
```

Wichtig: Wenn ein vorgebautes Strukturelement geändert oder hinzugefügt wurde, muss die Datei `.checked-initialdata` vor dem Start des Servers gelöscht werden, damit die Änderungen wirksam werden. Dies wird in Zukunft evtl. automatisiert werden.

Der Array Datentyp

Arrays sind ein weiterer Datentyp, der vom Schema unterstützt wird. Mit ihnen können geordnete Listen von skalaren Werten in der Datenbank abgelegt werden. Arrays sind in ihrer Länge, zumindest theoretisch, unbeschränkt und nicht auf eine feste Anzahl Einträge pro Attribut festgelegt. Praktisch muss natürlich abgewogen werden, wie "groß" ein Array werden darf, da praktische Limits sowohl im Speicher als auch auf Datenbankebene existieren.

Jedoch eigenen sich Arrays nicht für Situationen, in denen Datenbankqueries häufig einzelne im Array enthaltene Werte abfragen müssen, denn hierfür gelten teils starke Geschwindigkeitseinbußen. Für diese Fälle sollte eine explizite Posten-Relation bevorzugt werden. Oder, falls die maximale Länge der Arrays konstant und ausreichend klein ist, eine umsetzung der möglichen Werte als einzelne Attribute.

Vordefinierte Arrays von Skalaren

Die folgenden Skalare können in Array Form abgelegt werden. In den meisten Fällen reicht es, eckige Klammern hinter den Typnamen des Skalars hinzuzufügen um einen neuen Typ als dessen Array zu definieren.

Arrays werden im Zuge der Übertragung zwischen Server und Client zu String normalisiert und serialisiert. Dabei wird üblicherweise eine json kompatible Darstellung verwendet. Die Spalte *Protokollserialisierung* gibt das Format der einzelnen Werte in der Json-Liste an. Fehlt bei einem Index der Wert, sofern erlaubt, wird dieser immer als *null* (ohne Anführungszeichen) gerendert.

Table 1. Die folgenden Arrays sind vordefiniert.

Skalar	**Typname des Arrays**	**SQL interner Datentyp**	**interne Serialisierung**
Boolean	Boolean[]	boolean[]	boolean, d.h. *true*, *false*
Datetime	Datetime[]	timestamptz[]	String, im ISO 8601 Format
Decimal	Decimal[]	decimal[]	Dezimalzahl, int./U.S. Notation (Punkt), ohne Anführungszeichen
Duration	Duration[]	interval[]	String, im XML 1.0 Duration Format
Email	Email[]	text[]	String
Integer	Integer[]	int4[]	Ganzzahl, ohne Anführungszeichen
Long	Long[]	bigint[]	Ganzzahl, ohne Anführungszeichen
String	String[]	text[]	String
Timespan	Timespan[]	bigint[]	Ganzzahl, in Sekunden

Bei Datentyp String[] sollte die Anwendungslogik darauf achten, dass die Textmenge pro Index nicht Überhand nimmt. Für große bzw. viel Text sind alternative Speichermöglichkeiten, z.B. als BLOB, möglicherweise die bessere Wahl.

Verwendung als Attributtyp

Arrays lassen sich im Schema wie normale Skalare verwenden.

```
<attr name="Longs" type="Long[]"/>
<attr name="Integers" type="Long[]"/>
<attr name="Kommazahlen" type="Decimal[]"/>
```

Attributnamen von Arrays sollten im Plural sein, da es sich um eine Menge handelt.

Derzeit darf *kein* Singular bei der Attributedefinition angegeben werden, da sonst der Singular den Namen der Tabellenspalte in der DB bestimmt. Das kann bei OQL Queries zu Verwirrung und Datenverlust führen, falls sich das Verhalten hier ändern sollte.

Definition von neuen Arraytypen

Es gibt folgende Möglichkeiten einen neuen Array-Type zu definieren:

1. Man leitet von einem existierendem Array ab. In diesem Fall erbt man alle vorhandenen Parameter, sowohl auf der Array- als auch der Komponenten Ebene. Parameter des abgeleiteten Arrays werden übernommen und können überschrieben werden. Nachteil ist, dass man die einzelnen Komponenten nicht weiter einschränken kann.
2. Man erstellt einen neuen Array, der sich auf eine existierende Komponente stützt. In diesem Fall kann man Einschränkungen auf Komponenten Ebene beliebig bestimmen. Nachteil kann sein, dass man Einschränkungen auf Array Ebene explizit definieren muss.

Beispiele hierzu werden in der Sektion Vererbung genannt.

Limitiere die Komponenten

Einschränkungen auf den Wert einzelner Komponenten werden über die Typdefinition der Komponente bestimmt.

Parameter

Die folgenden Parameter stehen Arraytypen zur Verfügung.

allowNullElements

Erlaubte Werte: *true*/*false*
Default: *false*
Wenn *true*, dann ist *null* ein gültiger Einzelwert innerhalb des Arrays. Wenn *false*, dann darf der Array auf keinem Index ein *null* enthalten.

useComponentsGUIText

Erlaubte Werte: *true*/*false*
Default: variabel
Wenn *true*, dann wird die Textdarstellung in der GUI der Einzelwerte von der definierten Komponente übernommen. Um korrektes Escaping des Inhaltes zu gewährleisten wird wird der Array in diesem Fall wie ein String[] dargestellt, unabhängig vom echten Datentyp. Wenn *false*, dann entspricht die Textdarstellung des Arrays dem json Standart. Sofern nichts angegeben ist, wird für Ganzzahlen *false* genommen, für Strings oder internationalisierbare Komponenten *true*.

minElements

Erlaubte Werte: Integer >= 0
Default: ohne Wert
Legt eine untere Schranke (inklusive) für die Anzahl an Einzelwerten des Arrays fest. Ohne Angabe: Unbeschränkt, d.h. effektiv 0.

maxElements

Erlaubte Werte: Integer > 0

Default: ohne Wert
Legt eine obere Schranke (inklusive) für die Anzahl an Einzelwerten des Arrays fest. Muss größer oder gleich *minElements* sein, sofern beide Werte angegeben sind. Ohne Angabe: Unbeschränkt.

Vererbung

Arrays müssen entweder einen anderen Array erweitern oder einen neuen Array definieren. Ein neuer Array kann definiert werden, indem dem Namen des Typs einer einzelnen Komponente ein **[]** nachgestellt wird.

```
<Type name="NumericStringArrayKomponententyp" extends="String"> ①
  <parms mustMatch="[+-]?[0-9]+"/> ②
</Type>
<Type name="NumericStringArray" extends="NumericStringArrayKomponententyp[]"> ③
  <parms allowNullElements="false"/> ④
</Type>
<Type name="NullableNumericStringArray" extends="NumericStringArray"> ⑤
  <parms allowNullElements="true"/> ⑥
</Type>
```

① Definiert einen neuen String-Typ, der sich von String ableitet.

② Einschränkungen, die für jeden einzelnen Wert dieses Typs gelten.

③ Typname der Komponente + [] definiert einen Array dieses Typs.

④ Einschränkungen auf Länge und Inhalte des Arrays, hier: verbiete *null* als Wert im Array.

⑤ Erweitert einen existieren Array typ. Der Komponententyp wird übernommen und ist *NumericStringArrayKomponententyp*

⑥ Erlaube *null* als Array Werte.

Tabellenansicht in der GUI

Verwendet der aktuelle BOLoader ein InstrumentingSchema, kann der Inhalt eines Arrays auch als Relation dargestellt werden. Dies ist in erster Linie für Darstellung in der GUI und (readonly) Zugriff in Reports gedacht. Intern sollten Arrays immer direkt verwendet werden.

Das "AsRelation" Postfix von Array Attributen

Das InstrumentingSchema erzeugt die nötige Infrastruktur automatisch, wenn das erste Mal auf ein Attribut mit Postix **AsRelation** zugegriffen wird, dieses Attribut derzeit noch nicht existiert und der Name ohne das Postfix ein Array Attribut ist.

Daraufhin wird intern ein Objekt vom Typ *ArrayZeilenWrapper* gebaut, welches jede Komponente des Arrays über *ArrayZeilenDelegate* erzeugt und den Zugriff als 1-n Manyrelation erlaubt. Der Wert der Komponente ist typsicher über das Attribut *Value* verfügbar. Um Namenskonflikte bei virtuellen Properties zu vermeiden, wird eine virtuelle Sub-Entität von *ArrayZeilenDelegate* als Namensraum erzeugt.

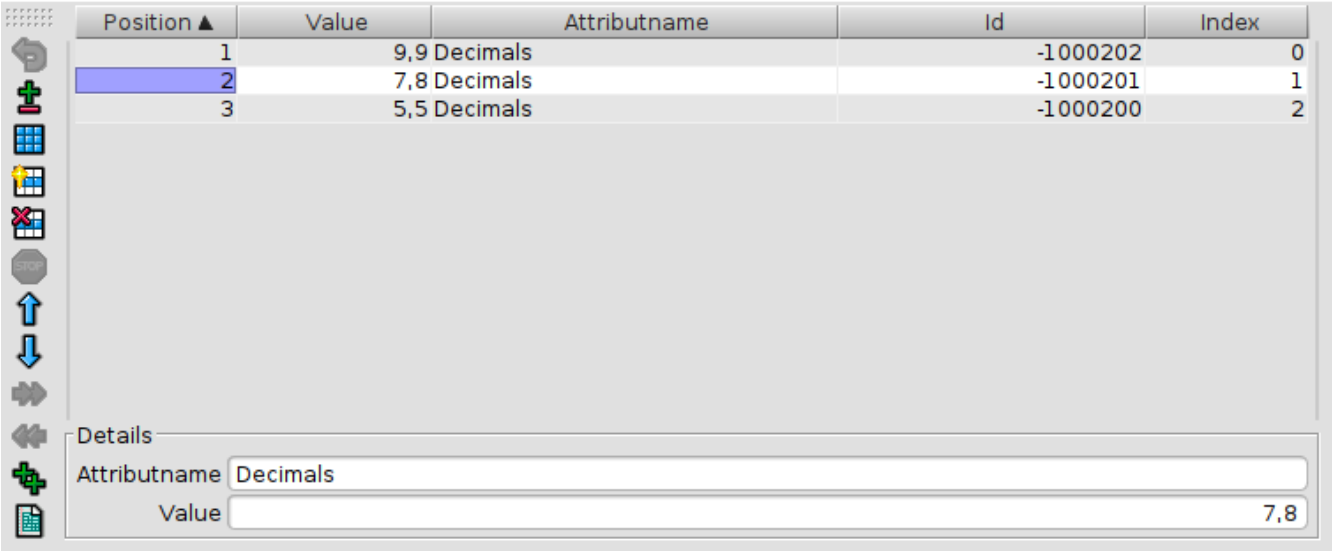
Schema

```
<attr name="Werte" type="Decimal[]"/>
```

Formulardefinition

```
<Table property="WerteAsRelation" columns="Position, ASC | Value | Attributname | Id |
Index">
  <DetailView>
    <Border etched="true" title="$R{Details}">
      <View>
        <Text property="Attributname"/>
        <Text property="Value"/>
      </View>
    </Border>
  </DetailView>
</Table>
```

Mit den Beispielwerten `[9.9,7.8,5.5]` (en_us) bzw. `["9,9","7,8","5,5"]` (de_de) sieht das Formular dann folgendermaßen aus:



Position wird automatisch als *itemProperty* der Relation gesetzt, wodurch es möglich wird zwei Komponenten zu vertauschen, neue einzufügen oder existierende zu löschen. Sofern das Array Attribut nicht Readonly ist, kann der Benutzer über das Value Attribut den Wert auch verändern.

Die virtuelle Entität als virtueller Namensraum

Was ist der virtuelle Namensraum?

Der von den ArrayAsRelation aufgespannte virtuelle Namensraum ist nichts anderes als eine virtuelle Entität. Diese virtuelle Entität existiert nur innerhalb des aktuellen InstrumentingSchemas. Die von Arrays verwendete v-Entität leitet sich immer von ArrayZeilenDelegate ab, sie ist folglich nicht persistent.

Vorteile

Die virtuellen Entitäen erlauben eine einfachere Formulardefinition. So ist der Einzelwert des Arrays immer mit dem (virtuellen) Attribut *Value* definiert. Dieses Attribut enthält sogar die Typinformationen des Arrays, dadurch wird der Wert korrekt gerendert und die GUI kann das Format bei der Eingabe prüfen. Kurz: Dadurch verhält sich die GUI bei den einzelnen Komponenten genau wie bei Skalaren dieses Typs.

Die virtuelle Entität im generischen ArrayAsRelation Fall

Der Name der virtuellen Entität besteht aus drei Elementen, welche durch Unterstriche _ voneinander getrennt sind.

1. *ArrayComponent* ist ein konstanter Prefixtext. Er wurde als 'Future proofing' hinzugefügt und grenzt den erzeugten Namen von anderen bestmöglich ab.
2. *Entitätsname* der Name der Entität, auf dem das Array Attribut definiert ist.
3. *Attributname* der Name des Array Attributs.

Die Kombination Entity + Attribut muss innerhalb des Schemas immer eindeutig sein und bietet deswegen eine gute Basis für eindeutige, generierte Namen. Ist das Array Attribut *Werte* auf einer Entität *Messwert* deklariert, so heißt die virtuelle Entität **ArrayComponent_Messwert_Werte**. Das gilt auch, falls das anzuzeigende BO eine Subklasse von *Messwert* sein sollte.

Die virtuelle Entität bei selbstdefinierten ArrayZeilenWrapper

Werden mehrere Arrays verglichen, so folgt dem o.g. Namensschema noch ein numerischer Hashcode. Dieser berechnet sich aus den Entitäten und Array Attributen bei Initialisierung des Wrappers. Dieser Postfix sollte als instabil angesehen und ist nicht für die Verwendung mit virtualProperties im Formular vorgesehen, sofern diese nicht dynamisch erzeugt werden.

Um in diesem Fall virtualProperties mitgeben zu können, muss dem ArrayZeilenWrapper bei der Initialisierung ein expliziter Name für die Entität mitgegeben werden. Siehe der Parameter mit Wert `AUniqueEntityName` in diesem Beispiel.

Zeilen um virtuelle Properties anreichern

Das Vorgehen um weitere Tabellenspalten zu generieren ist (fast) wie im normalen Fall. *Namen* ist hier ein einfaches Attribut vom Typ `String[]`, welches auf einer Entität *Schauspieler* definiert ist und eine Liste von Charakternamen enthält.

Die virtuelle property *Greeter* greift auf den einzelnen Namen sowie die Position in der Liste zu und erzeugt einen einfachen Text. Da der Text etwas länger ist, wurde die expectedWidth erhöht - alternativ lässt sich das natürlich auch über die Spaltendefinition machen.

Der erzeugte Namensraum für die Entität (singular) Schauspieler und das Attribut Namen ist *ArrayComponent_Schauspieler_Namen*.

```
<Table property="NamenAsRelation" columns="Position, ASC | Value | Greeter">
  <virtualProperty name="Greeter" entity="ArrayComponent_Schauspieler_Namen">
    <ui expectedWidth="30"/>
    <get><![CDATA[ return "Hallo $bo.value, du bist ${bo.position > 3 ? 'leider nicht
' : ''}in den Top 3" ]]></get>
  </virtualProperty>
  <DetailView>
    <Border etched="true" title="$R{Details}">
      <View>
        <Text property="Value"/>
        <Text property="Greeter"/>
      </View>
    </Border>
  </DetailView>
</Table>
```

Der Namensraum wird lazy erzeugt, wenn im Formular das erste mal auf das *AsRelation* Attribut zugegriffen wird. Deswegen muss die virtualProperty derzeit innerhalb des Table-Elements definiert werden.

Ausserhalb kommt es zu folgender Fehlermeldung:
Das in einer Spaltendefinition angegebene Attribut "Greeter" existiert nicht; vielleicht ein Tippfehler?

Selbstdefinierte Tabellenansicht in der GUI

Es ist möglich beliebig viele Arrays miteinander zu vergleichen, diese müssen nichteinmal von der gleichen Instanz kommen.

Gegeben sei folgende, einfache Entität.

```
<Entity name="Beispiel" extends="CoreBO"....>
   <attr name="Other" type="Beispiel" relation="n-1"/>
   <attr name="Longs" type="Long[]"/>
   <attr name="Strings" type="String[]"/>
</Entity>
```

Um die einzelnen Werte von *Longs* und *Strings* des aktuellen BOs mit dem von *Other* zu vergleichen, ist folgende Infrastruktur nötig:

```
<virtualProperty name="CombinedArrayWrapper" entity="Beispiel"
type="ArrayZeilenWrapper" relation="n-1" cached="SIMPLE">
  <get><![CDATA[
    import de.ipcon.db.core.ArrayZeilenWrapper
    return ArrayZeilenWrapper.of([bo, bo, bo.other, bo.other] as BO[],
         ['Longs', 'Strings', 'Longs', 'Strings'] as String[],
         ['MyLong', 'MyString', 'OtherLong', 'OtherString'] as String[],
         'AUnqiueRelationName',
         'AUniqueEntityName')
  ]]></get>
</virtualProperty>
<!-- switch the bo to the wrapper -->
<Element property="CombinedArrayWrapper">
  <visibleIf language="groovy"><![CDATA[ true //FIXME currently needed ]]></visibleIf>
  <!-- show the arrays as relation -->
  <Table property="AUnqiueRelationName"
         columns="Position, ASC | MyLong | OtherLong | MyString | OtherString">
    <!-- optional, define additional properties -->
    <virtualProperty name="Identisch" entity="AUniqueEntityName" type="Boolean">
      <get><![CDATA[ return bo.MyLong == bo.OtherLong && bo.MyString == bo.OtherString
]]></get>
    </virtualProperty>
    <DetailView>
      <Border etched="true" title="$R{Details}">
        <View>
          <Text property="MyLong"/>
          <Text property="OtherLong"/>
          <Text property="MyString"/>
          <Text property="OtherString"/>
          <Checkbox property="Identisch"/>
        </View>
      </Border>
    </DetailView>
  </Table>
</Element>
```

- **cached="SIMPLE"** ist wichtig, da der ArrayZeilenWrapper bei einer Änderung am bo nicht neu erzeugt werden darf.
  - Ein ArrayZeilenWrapper registriert sich als Listener an den Quell-Arrays und reagiert auf Änderungen an den Daten.
  - Mehrere Wrapper die von der Definition identisch sind, könnten sich ohne die *cached* Anweisung gegenseitig in die Quere kommen.
- **ArrayZeilenWrapper#of** hat einige Parameter, welche, über den gleichen Index als Tuple zusammengefasst, die Datengrundlage definieren.
  - **BO[] bos** : Die BO Instanzen mit den konkreten Daten.
  - **String[] attribute**: Der Name des Array Attributs, aufgelöst auf dem entsprechenden BO Parameter, welches als Datengrundlage gilt.

    - **String[] aliase**: Entweder komplett null für autom. generierte Attribute ODER eine Liste mit eindeutigen Bezeichnern, die eine Komponente des BO x Attribut - Tupels repräsentieren.
        - Im folgenden bedeutet 'alias': Der Name des autom. generierten virtualProperties, welches auf einer virtualEntity von ArrayZeilenDelegate definiert ist.
    - **AUnqiueRelationName** darf noch nicht als Attribut auf der Entität ArrayZeilenWrapper existieren.
        - Es ermöglicht den Wechsel in den erzeugten Namensraum.
        - Dies ist das Gegenstück zum autom. generierten "...AsRelation" Attributnamen des einfachen Falls.
        - Übliches Fehlerbild, wenn der Name nicht eindeutig ist: Die Columns können im Formular nicht aufgelöst werden, zudem haben die Properties u.U. die falsche Formatierung.
    - **AUniqueEntityName** bestimmt den Namen der virtuellen Entität, die sich von *ArrayZeilenDelegate* ableiten muss.
        - Wird automatisch angelegt, sofern sie noch nicht existiert.
        - Kann weggelassen werden, solange der Namensraum nicht erratbar sein muss - i.e. keine virtualProperties hinzugefügt werden müssen.
- **visibleIf** und Element-Wrapper sind derzeit nötig.
    - Todo: Zumindest für das 'visibleIf' die Ursache prüfen und vereinfachen.

Einschränkungen:

- Die Kombination **bo-Instanz** zu Attribut muss eindeutig sein, d.h. wird es zu einem Fehler kommen, falls `bo.other == bo` ist.
- Die verwendeten Aliase dürfen noch nicht als Attribute auf ArrayZeilenDelegate oder einer super-Entität existieren.
    - Ein Alias 'Id' ist z.B. nicht erlaubt, weil das Attribut 'Id' auf der Entität BO definiert ist.
    - Als Referenz-Entität gilt *ArrayZeilenDelegate*, d.h. es sind nur Aliase erlaubt, die auf ArrayZeilenDelegate noch nicht existieren.
    - Gleiche Aliase in unterschiedlichen virtualEntities von ArrayZeilenDelegate sind jedoch explizit erlaubt. Der Alias 'Value' ist solches ein Beispiel.
- Die zusammengefassten Arrays **müssen** alle die gleiche Länge haben. Ansonsten kommt es zu einer Exception und/oder die Tabelle wird leer dargestellt.
- Falls Arrays mit unterschiedlichen Dimensionen angezeigt werden sollen, kann dies über ein/mehrere addVirtualProperty gemacht werden, die jeweils einen Array zurückgegeben. Das virtualProperty muss sich darum kümmern, den Rückgabewert mit passenden Dummy-Werten auf die richtige Länge zu padden. Ist etwas umständlich, aber machbar.

Ansicht in Automatikformularen als String

In Automatikformularen werden Arrays in einfachen Textfeldern mit ihrer an json angelehnten Textdarstellung angezeigt. Diese Darstellung kann die Locale beachten, d.h. der angezeigte String

kann sich vom *newValue* der BPs unterscheiden.

- FIXME ein paar konkrete Beispiele.

ArrayZeilenDelegate / Wrapper

- Erklärung Attribute auf ArrayZeilenDelegate
  - Index nur für interne verwendung, **maximal** readonly zugriff von außerhalb erlaubt. Falsche Verwendung kann die Arraydaten zerstören.

Verwendung in OQL Queries

siehe OQL Dokumentation

Umgang mit Arrays im Code

Der Umgang mit Arrays im Code wurde bestmöglichst an das Verhalten der normalen Attribute angelehnt. Im Gegensatz zu den normalen Skalaren sind Arrays jedoch von rund am "mutable", was in der Implementierung von beschreibbaren v-attrs beachtet werden muss.

Persistente Array Attribute

Verfügbare Methoden

Beispiel: Ein Attribut "Werte" ist im Schema als "Integer[]" definiert. Dann werden die folgenden Methoden automatisch generiert:

- `getWerte():Integer[]` gibt eine Kopie des gespeicherten Arrays zurück. Das zurückgegebene Array darf modifiziert werden, es hat keinen Einfluss auf die Werte des BOs. Kann null oder leer sein.
- `getWerte(fallback:Integer[]):Integer[]` gibt eine Kopie des gespeicherten Arrays zurück bzw. `fallback`, falls das Attribut *null* ist. Ist das Attribut das leere Array, dann wird dieses leere Array zurückgegeben.
- `getWerteNN():Integer[]` wie oben, mit einem leeren Array als fallback.
- `getWerte(index:int):Integer` gibt die Komponente des Arrays mit diesem 0 basierten Index zurück. Hierdurch können auf einzelne Werte zugegriffen werden, ohne das eine Kopie des Arrays erstellt wird. Es findet keine Indexprüfung statt und der Array wird direkt zugegriffen, d.h. NullPointer- oder IndexOutOfBounds-Exceptions sind möglich.
- `getWerteLength():int` gibt die aktuelle Länge des Arrays zurück. Ist das Attribut *null*, wird 0 zurückgegeben.
- `setWerte(neuerWert = Integer[]):void` setzt das Attribut auf einen neuen Wert und zeichnet die Änderungen auf, sofern das BO included wurde. Die übergebene Instanz wird intern als Kopie gespeichert, wodurch spätere/implizite Änderungen an der Parmameter Instanz ignoriert werden.
- `setWerte(index:int, v:Integer)` aktualisiert eine einzelne Komponente des Arrays am gegebenen Index auf einen neuen Wert. Es findet keine Indexprüfung statt, d.h. NullPointer- oder IndexOutOfBounds-Exceptions sind möglich. Erstellt im Hintergrund eine Kopie des Arrays.

Hinweise

- Code sollte so strukturiert werden, dass ein Array zuerst vollständig gebaut und befüllt wird, bevor dessen es einem Attribut zugewiesen wird. Da Arrays bei den Attributzugriffen kopiert werden müssen, erhöht das die Performance und verhindert je nach Kontext, z.B. während verifyOnServer, dass Zwischenergebnisse aufgezeichnet werden.
- Soll nur über ein Array iteriert werden, dann kann ist ein indexbasierter loop effizienter als ein 'loop over':

```
loop i=0 to bo.getWerteLength()
   val = bo.getWerte(i)
   ...
   end
```

- Die generierten Methoden verwenden Arrays statt immutable Listen, da zum Zeitpunkt der implementation keine Generics in Netrexx verfügbar waren. D.h. konnten dadurch mehrere explizite Casts vermieden werden.

Hinweise bei v-attrs:

- Vorsicht, wenn der Rückgabewert des getters intern gecached wird. Arrays sind von natur aus mutable und sollten nicht direkt rausgegeben werden, falls Caching im Spiel ist.

Persistenzschicht von MyTISM

MyTISM verwendet ein mehrschichtiges Persistenzkonzept, um Daten sicher und effizient zu speichern. Ein zentrales Element ist dabei die Unterscheidung zwischen "Soft Delete" und "Hard Delete" (Purge), die im Folgenden detailliert erläutert wird.

Löschen von Daten in MyTISM

Soft Delete

Standardmäßig werden Objekte in MyTISM nicht physisch aus der Datenbank gelöscht, sondern lediglich als "gelöscht" markiert. Dies geschieht durch das Setzen eines Flags namens "Ldel" auf den Wert `TRUE`. Dieser Mechanismus wird als "Soft Delete" bezeichnet.

Diese Objekte werden in den meisten Abfragen und Ansichten dann nicht mehr angezeigt. Admins könnten diese Objekte einblenden. Sie werden dann normalerweise durchgestrichen dargestellt.

Vorteile:

- **Datenwiederherstellung:** Versehentlich gelöschte Objekte können von Admins leicht wiederhergestellt werden.
- **Datenhistorie:** Gelöschte Objekte sind weiterhin mit ihrer kompletten Änderungshistorie inkl. des Löschvorgangs in der Datenbank verfügbar, was für Audits oder Analysen nützlich sein kann.

Nachteile:

- **Speicherplatz:** "Soft deleted" Objekte belegen weiterhin Speicherplatz in der Datenbank.
- **Performance:** Bei einer großen Anzahl von "soft deleted" Objekten könnte die Performance der Datenbankabfragen beeinträchtigt werden, da die Tabellen größer sind, als sie sein müssten.
- **Fehlerquelle:** Bei selbst erstellen programmatischen Abfragen in der Datenbank oder beim Traversieren von Objekten in Relationen muss man daran denken, gelöschte Objekte explizit auszunehmen oder zu überspringen, wenn das gewünscht ist.
- **Datenschutz:** Reicht für personenbezogene Daten nicht zur Einhaltung von Datenschutzbestimmungen wie der DSGVO aus.

Wiederherstellung von "Soft Deleted" Objekten

"Soft deleted" Objekte können über die MyTISM-GUI (sofern die Undelete-Action für den Benutzer verfügbar ist und die gelöschten Objekte sichtbar sind) oder programmgesteuert über die Methoden `BO#undelete()` bzw. `BO#markUndelete()` wiederhergestellt werden.

Hard Delete (Purge)

Im Gegensatz zum Soft Delete ermöglicht ein Hard Delete das endgültige Löschen von Objekten aus der Datenbank. In MyTISM wird dieser Vorgang als "Purge" bezeichnet und ist nur dem Backend und speziellen Backend-Diensten vorbehalten.

Anwendungsfälle:

- **DSGVO-Konformität:** Zum Löschen von Daten, die gemäß der Datenschutzgrundverordnung (DSGVO) gelöscht werden müssen.

- **Datenbereinigung:** Zum Entfernen von alten, irrelevanten Daten, um Speicherplatz freizugeben.

> Purging ist unwiderruflich! Gelöschte Daten können nicht wiederhergestellt werden!

Unterschiede zwischen Soft und Hard Delete im Überblick

Feature	Soft Delete	Hard Delete (Purge)
Löschvorgang	Objekt wird als "gelöscht" markiert (Ldel = TRUE)	Objekt wird physisch aus der Datenbank entfernt
Wiederherstellung	Möglich	Nicht möglich
Zugriff	Standardmethode für Benutzer	Nur für Backend und spezielle Dienste
Anwendungsfälle	Standardlöschvorgänge	DSGVO-Konformität, Datenbereinigung

Zusammenfassung

MyTISM bietet mit Soft Delete und Hard Delete (Purge) zwei Mechanismen zum Löschen von Daten. "Soft Delete" ist die Standardmethode und ermöglicht die Wiederherstellung von Daten. "Hard Delete" dient dem endgültigen Löschen von Daten und ist nur für spezielle Anwendungsfälle vorgesehen.

Sprachunterstützung und Internationalisierung

Einführung

MyTISM bietet eine durchgehende Unterstützung für verschiedene Locales, sowohl für die Übersetzung von Texten und Namen als auch für Ein- und Ausgabe von Zahlen, Daten, etc.

Wo wird Mehrsprachigkeit unterstützt und wie benutze ich sie?

Neben der direkten Benutzung in Programmcode mittels der Methoden L10n.msg() bzw. L10n.applyL10n() gibt es weitere Stellen an denen die Mehrsprachigkeit unterstützt wird.

Im GUI-Client Solstice, an vom Benutzer bearbeitbaren Stellen:

- In den Benutzer-Login-Scripten. Beispiel:

```
----
<Configuration>
  [...]
  <Locale>de</Locale>
</Configuration>
----
```

- In den XML-Definitionen für Plugins in den Benutzer-Voreinstellungen.
- In den XML-Definitionen für Defaults in Benutzer-Voreinstellungen.
- In Report-Definitionen und in den Parametern von Formularen, Schablonen und Lesezeichen.

Im GUI-Client Solstice, interne Funktionalität:

- Ausgabe von Name und ElterPfad von Benannts im Navigationsbaum bzw. in PolymorphicTemplateSelectionTreeModel.

In diesen "Texten" können Platzhalter `$R{key}` eingefügt werden. Diese werden dann automatisch vor der "Benutzung" durch zum aktuellen Locale passende Texte ersetzt.

Wie wird die konkrete Zeichenkette für einen Schlüssel gefunden?

L10nPackProviderI (z.Zt. de/ipcon/tools/L10n und de/ipcon/db/AbstractClient) halten benannte L10nPacks bereit, welche die diversen Textbausteine in unterschiedlichen Sprachen enthalten, gruppiert mittels entsprechender Schlüssel für jeden Textbaustein.

Beim Auflösen von `$R{key}`-Platzhaltern z.B. im Formularcode (und ebenso beim direkten Aufruf der L10n.msg()-Methoden) wird eine - je nach Aufrufart bzw. -ort unterschiedliche, und bei den im vorherigen Abschnitt aufgeführten Stellen, automatisch zusammengestellte - Liste der an dieser Stelle beteiligten bzw. relevanten Objekte übergeben (Z.B. für Formulare das Formular-Objekt

selbst).

Mittels dieser übergebenen Objekte wird nun eine Liste von relevanten L10nPacks erstellt, in denen mittels des Schlüssels "key" nach den angeforderten Textbausteinen gesucht wird. Wird ein zum Schlüssel passender Textbaustein gefunden, wird seine zur gewünschten Sprache passende Version zurückgeliefert.

Schlüsselnamen dürfen nur Buchstaben, Zahlen, '_', '-', '.', '~' und '/' enthalten.

Welche L10nPacks gibt es und wie sind diese organisiert? Wie wird bestimmt, welche L10nPacks nach Texten durchsucht werden?

Die Benamsung bzw. Hierarchie der L10nPacks folgt in der Regel der Klassen- bzw. Paketstruktur der Java-Klassen. In den meisten Fällen bestimmen die Java-Klassen der beteiligten Objekte und die Java-Pakete denen diese angehören die zu durchsuchenden L10nPacks.

L10nPack-Namen dürfen nur Buchstaben, Zahlen, '_', '-' und '.' enthalten, wobei '.' das Trennzeichen zum Aufsplitten der Namen ist.

Beispiel: Ein Objekt der Klasse "de.ipcon.form.FText" will den Textbaustein mit Schlüssel "eineNachricht" in der aktuellen Sprache ausgeben. In diesem Fall werden - sofern vorhanden - die L10nPacks "de.ipcon.form.FText", "de.ipcon.form.FPanel" (FText leitet sich von FPanel ab), "de.ipcon.form", "de.ipcon" und "de" in dieser Reihenfolge nach einer zum Schlüssel passenden Version des Textes durchsucht.

Web

Abweichend hiervon ist in Grails und Cauldron aktuell historisch bedingt noch eine andere Namensgebung üblich und es gibt kein eigentliches Mapping mit Automatismen für Klassen oder Entitäten.

In Grails gibt es oft nur ein einzelnes Bundle `Grails`, das aber auch aufgeteilt werden kann. Die Namensgebung lautet dann bspw. `Grails.checkout` oder `Grails.user.settings`.

In Cauldron besteht hier prinzipiell freie Namenswahl, neue Projekte sollten sich aber ebenfalls an das Paketnamensschema halten.

Welches sind die "beteiligten bzw. relevanten Objekte"?

Dies ist unterschiedlich und hängt davon ab, wo und wie die L10n-Funktionalität genutzt wird, z.B.:

- Beim direkten Aufruf von L10n.msg() wird normalerweise automatisch die aufrufende Klasse ermittelt und als das (einzige) beteiligte Objekt übergeben.
- Bei Benutzung von `$R{key}`-Platzhaltern im Parameter von Formularen, etc. wird die Klasse des

im Formular, der Schablone oder im Lesezeichen dargestellten BOs sowie die Klasse des Strukturelements (Formular.class, etc.) selbst übergeben.

Im Normalfall wird die Liste der L10nPacks für eine Klasse einfach nach der im vorherigen Abschnitt beschriebenen Methode (Klassen + Pakete) erstellt. Ist für eine Klasse aber ein sog. L10nPathCompilerI beim L10n registriert, so bestimmt dieser, welche L10nPacks für die entsprechende Klasse in die Liste aufgenommen werden (siehe z.B. "FormularPathCompiler" in Formular.nrx).

Genauere Informationen finden sich bei den entsprechenden Aufrufen von L10n.applyL10n() bzw. L10n.msg() (bei letzterem nur selten, da dort fast nie ein expliziter "path" übergeben wird und fast immer die oben erwähnte Automatik benutzt wird) im Quellcode von MyTISM. L10n.compilePath() enthält weitere Informationen darüber, wie die Liste der zu durchsuchenden L10nPacks zusammengestellt wird.

Wo kommen die (Daten der) L10nPacks her?

Die Texte/Daten für die L10nPacks werden z.Zt. aus zwei Quellen gezogen. Die L10n-Klasse lädt ihre Daten aus Dateien .../resources/*.properties die in den Quellcode-Verzeichnissen liegen und beim Bauen ebenfalls in die JARs eingebunden werden. Daneben lädt der Server noch die in der Datenbank befindlichen L10nBundles in seinen L10nCache.

Die Texte der *.properties-Dateien werden alle "von Hand" angelegt und bearbeitet. Die L10nBundles, etc. werden teilweise automatisch generiert, können aber auch von Hand angelegt und bearbeitet werden.

Für jede im Schema definierte Entität, Beispiel "de.ipcon.db.core.Benannt", werden einige L10n-Daten automatisch angelegt und "gewartet". Diese sind im Beispiel zuerst das L10nBundle "de.ipcon.db.core" mit den L10nResourcen "_Benannt" sowie "_Benannt-s" (Name/Singular und Plural der Entität), L10nResources für die Ordner, also hier "Interna" (FIXME ggf. weitere) sowie L10nResources für jedes Attribut der Entität also hier "Name", "Beschreibung" usw.

Desweiteren wird für jede Entität noch ein eigenes Bundle, im Beispiel "de.ipcon.db.core.Benannt", angelegt und dort werden ebenfalls noch einmal wie oben für alle Attribute L10nResources angelegt.

Diese L10nBundles, bzw. deren L10nEntries und L10nResources, werden bei jedem Serverstart überprüft, ob Entitäten oder Attribute hinzugekommen sind - diese werden dann automatisch angelegt - bzw. weggefallen sind - dort werden dann für weggefallene Attribute automatisch entsprechende, ehemals gebrauchte L10nEntries und L10nResources entfernt.

In neueren Projekten werden Daten aus `nrx/[ Projektverzeichnis ]/resources/l10n/[ Projekt-Package ].bo_[ISO-Kürzel]` automatisch importiert. In diesen, von Hand angelegten Dateien, dürfen (bzw. zumindest sollten) sich nur Schlüssel-Text-Paare für im Schema definierte Entitäten und deren Attribute befinden. Ansonsten werden die "überzähligen" Texte bei jedem Serverstart erst angelegt, und dann, da es keine entsprechenden Entitäten bzw. Attribute (mehr) gibt, direkt wieder gelöscht.

"Freie" Texte (z.B. solche für Titel/Texte in Formularen) sollten - selbst wenn sie im Prinzip nur für

eine Entität benutzt werden - in einem eigenen Paket `nrx/[ Projektverzeichnis ]/resources/l10n/[ Projekt-Package ]_[ISO-Kürzel]` untergebracht werden.

Genauere Informationen finden sich in L10nBundle.initEnvironment() und den dort benutzten Methoden.

L10n und das Anführungszeichen bzw. Apostroph

Die Verwendung von Anführungszeichen oder Apostrophen in Übersetzungen (in `properties`- oder Bundles-Dateien im Verzeichnis `resources/l10n/`) kann zu Problemen führen, insbesondere wenn diese in XML-Texten verwendet werden und unbeabsichtigt Abschnitte beenden, die nicht beendet werden sollten. Dies tritt häufig auf, wenn es um die Definition von Tabellenspalten geht. Besondere Vorsicht ist geboten, wenn es um Übersetzungen von Entitäts- und Attributnamen geht, da diese oft an solchen Stellen eingesetzt werden.

Um solche Probleme zu vermeiden, empfiehlt es sich, anstelle der „einfachen“ Zeichen `'` und `"`, die ursprünglich noch aus der Zeit der Schreibmaschinen stammen und lediglich eine vereinfachte Darstellung dieser Zeichen bieten, die schöneren und eigentlich korrekten typographischen Zeichen (erreichbar über AltGR+`'`) sowie (AltGr+`V`) und (AltGr+`B`) zu verwenden. Diese Zeichen werden in der Regel nicht als Steuerzeichen zur Abtrennung verwendet und tragen somit zur Vermeidung von unerwünschten Effekten bei.

Wichtige Klassen

de.ipcon.tools.L10n

Die zentrale Klasse. Enthält u.A. die msg()-Methoden die zu einem gegebenen Key die zum gewünschten/aktuellen Locale passende Version der entsprechenden Zeichenkette liefern. Enthält auch diverse Methoden um Format-Objekte zum formatieren von Zahlen, Daten, etc. zu erhalten.

de.ipcon.db.core.L10nBundle

Sammlung von L10nResources.

de.ipcon.db.core.L10nResource

Entspricht grob einer Zeichenkette welche in unterschiedlichen Sprachen ausgegeben werden können soll. Hat einen oder mehrere L10nEntries.

de.ipcon.db.core.L10nEntry

Konkrete Version der Zeichenkette für ein bestimmtes Locale (grob: eine Sprache).

Eingabe von L10n-Daten

FIXME Stichworte

neues Bundle anlegen (de.venice) bzw. in einem bestehenden Bundle (de.venice.bo) was hinzufuegen ⇒ schauen, dass das Bundle auch Preloaded wird und auch die PfadPos setzen (0). Wenn trotzdem ein neuer Eintrag nicht direkt gefunden wird, dann kann es noetig sein den Server durchzustarten. Das kann der Fall sein, wenn das Bundle erstmalig auf Preload und/oder PfadPos gesetzt wird.

Einfaches Hochkomma muss "escaped" werden ⇒ doppelt schreiben

$R-Tags in Formularen, etc.: Suche nach title=", label=", text=".

Die Formularengine des Solstice Clients

de.ipcon.form

Nachfolgende Dokumentation behandelt den Aufbau eines sogenannten Formulars im de.ipcon.form Package des MyTISM Frameworks. Sie soll den Entwickler in die Lage versetzen, Wünsche des Anwenders an die grafische Oberfläche umzusetzen. Im Gegensatz zu Web-Oberflächen sind diese für den Anwender viel effektiver und schneller bedienbar als die etwas generischeren und graphisch meist viel ansprechenderen, aber dennoch umständlich zu bedienenden Web-Oberflächen. Allerdings ist das Abstraktionsniveau im grafischen Client etwas geringer, um schneller zum Ergebnis zu kommen - manchmal ist es besser, den ein oder anderen Wunsch eines Anwenders zugunsten einer saubereren oder aber auch für eingeschränkte Benutzer (sei es technisch (niedrige Farbe, langsamer Rechner) oder auch körperlich (Farbenblindheit, Kurzsichtigkeit)) bedienbaren Lösung abzulehnen.

Hintergrund

Das Formularframework bzw. die Engine, die die Formulare aufbaut, hat zwei Ziele: Flexible Formulare und eine flexible Verwaltung dieser Formulare. Das HTML-Format bzw. dessen Vorgänger SGML bzw. XML haben mit ihrem Markup-Konzept einen entscheidenden Denkanstoß zur Entwicklung der jetzigen Implementation geliefert. Angereichert mit einer Meta-Ebene, die aus dem Datenbank-Backend MyTISM kommt und somit einen schnellen und effizienten Zugriff auf die Formulare gestattet sowie die Synchronisation der Formulare realisiert. Im folgenden werden die Objekte einzeln ausführlich vorgestellt, ihre Eigenschaften und ihre Verwendung dokumentiert. Alle Objekte außer den Lesezeichen haben die Eigenschaft, in bestimmten Kontexten als Auswahl zur Verfügung zu stehen, sei es, um ein neues Objekt zu erzeugen oder ein bestehendes anzuzeigen. Das Verfahren, diese Auswahl zu generieren, wird ebenfalls beschrieben.

Das Formular-Objekt

Das Formular-Objekt hat die Aufgabe, eine Eingabemaske für ein Objekt einer bestimmten BO-Klasse zu beschreiben. Das de.ipcon.form Package enthält die notwendigen Methoden, um eine solche Eingabemaske bestehend aus Oberflächenelementen wie Textfelder, Popuplisten und ähnlichem zu erzeugen und verwalten. Es stellt eigentlich das wichtigste und gleichzeitig das komplizierteste Objekt des Formularframeworks dar.

Eigenschaften

Das Formular-Objekt hat im wesentlichen folgende Eigenschaften:

1. Name: Eine klar abgrenzende Bezeichnung, die auch im Kontextmenu des jeweiligen Objektes erscheint.
2. Beschreibung: Eine etwas ausführlichere Beschreibung, durchaus als Platz für Bemerkungen wie den Verweis auf spezielle Versionen oder Spezifika. Sie wird dem Benutzer nicht in der GUI präsentiert und ist ausschließlich den Entwicklern vorbehalten.
3. Elter: Ein Verweis auf die ID des Strukturelements (meist ein Ordner oder eine Gruppe), unter dessen Repräsentation im Menubaum dieses Element absortiert wird. Wird gesetzt beim Drag’n’Drop im NavigationTree in der GUI.

4. IstAutomatik: Ein Wahrheitswert, der anzeigt, ob das Formular direkt aus dem Formulargenerator stammt. Falls Sie ein Formular abändern, achten Sie bitte darauf, daß dieser Wert false ist, sonst wird beim nächsten Schema-Update das Formular neu erstellt; der NavigationTree setzt es beim 'Move' dieses Flag automatisch auf false, um eine Fehlbedienung zu vermeiden.
5. Parameter: Hier steckt der Source des eigentlichen Formulars. Im folgenden wird der Inhalt dieser Eigenschaft ausführlich beschrieben.
6. BOTyp: Ein Verweis auf den Typ des BO (selbst natürlich ebenfalls ein BO), welches mit diesem Formular angezeigt werden kann. [fixme: Polymorphie?]
7. Gruppen: Ein Mehrfach-Verweis auf die Gruppen, die dieses Formular benutzen sollen.
8. Schablonen: Ein Mehrfach-Verweis auf die Schablonen-Objekte, die direkten Gebrauch von diesem Formular machen.
9. Priorität: ein 32bit signed Integer, der die Priorität des Formulars im Falle einer mehrfachen Auswahl von Formularen für ein Objekt festlegt und damit die Präferenz in diesem Fall festlegt.

Auswahl

Die Auswahl eines Formulars wird aufgrund folgender Regeln getroffen:

1. Zunächst werden alle passenden (gleicher BOTyp [fixme: Polymorphie]) Formulare erfragt, deren Priorität gesetzt ist und einer der eigenen Gruppen zugeordnet ist. Beim Benutzer Admin werden als Ausnahme auch diejenigen Formulare mit einbezogen, die keine Priorität haben (diese Mechanismus wird in einer der nächsten Versionen ausgebaut und ist damit als obsolet deklariert!).
2. Diese passenden Formulare werden der Priorität nach absteigend geordnet und dem Benutzer ggfs. per Kontextmenu zur Verfügung gestellt. Ein Doppelklick oder adäquate Aktion öffnet das nach dieser Sortierung am höchsten priorisierte Element. Die momentane Implementation ersetzt Formulare gleicher Priorität ohne eine deterministische (oder vielmehr eine dokumentierte Deterministik) oder vorhersehbare Präferenz. Daher bitte ich unbedingt auf eine klare Priorisierung zu achten. Der Zahlenraum der Priorität bietet genügend Spielraum: ca. -2 bis 2 Milliarden.

Definition

Fehler und Ursachen

Einige Fehler denen ich bei der MyTISM-Entwicklung schon begegnet sind und deren Ursachen bzw. Lösungsmöglichkeiten:

Compiler-Meldung "Object cannot be null"

BOs brauchen einen Konstruktor (ohne Argumente); es wird nicht automatisch einer gebaut oder der der Superklasse benutzt. BO-Klassen duerfen nicht "abstract" sein

bi-Tabelle kann nicht erstellt werden (nachdem die Datenbank gedropped und recreated wurde)

".checked*"-Dateien im Projektverzeichnis löschen.

Compiler-Meldung "Object bla is null but shouldn’t" (sic)

Beispiel: "Zustellversuch" hängt an "Sendeauftrag". Neuer Zustellversuch wurde angelegt, in Transaction included und an Sendeauftrag angehängt. Dumm nur: Sendeauftrag war nicht in Transaction included! FIXME: Hmm … das war aber wohl doch nicht das Problem :-(

Synchronisation der Strukturelemente

Das Formular "DateiSystemSync"

Das Benutzerhandbuch enthält bereits einen Abschnitt zu diesem Thema; ggf. sollten diese zusammengeführt werden.

FIXME! (werde später noch ein bisschen was dazu schreiben - sw) - anhand was wird rein- bzw. rausgesynct - Konventionen Dateiname - wofuer Tid - ...

Volltextsuche

Die Volltextsuche erlaubt die einfache und schnelle Suche nach gegebenen Suchbegriffen über alle in der MyTISM-Datenbank gespeicherten Objekte. Informationen zur allgemeinen Konfiguration und Bedienung finden sich in der MyTISM-Benutzerdokumentation. In diesem Kapitel befinden sich noch einige nur für Entwickler interessante Informationen, insb. zur Konfiguration der Suche im Schema und der Benutzung von Volltextsuche-Queries in Programm- oder Skriptcode.

Konfiguration im Schema

Berücksichtigte Daten

Standardmässig werden, mit wenigen Aussnahmen, die BOs aller Entitäten für die Volltextsuche aufbereitet. Von diesen BOs werden standardmässig die Inhalte alle Attribute, wiederum mit einigen Aussnahmen, in den Suchindex aufgenommen.

Berücksichtigte Entitäten

vgl. de/ipcon/db/fulltext/compass/SchemaMappingBuilder.isEntityIgnored()

Explizit *immer* ausgeschlossen werden die BOs nicht persistenter Entitäten, sowie die BOs der Entiäten `BT`, `BP` und `BX`.

Durch explizite Angabe von `indexed="no"` im Schema ist es möglich, weitere Entitäten von der Indexierung auszunehmen.

Beispiel:

```
<Entity name="InterneEntitaet" extends="BO" plural="InterneEntitaeten">
  <fulltext indexed="no"/>
  <attr name="KryptischerString"/>
</Entity>
```

FIXME Infos zum "Cascading", (Nicht-)Indexierung von Unterklassen

Berücksichtigte Attribute

vgl. de/ipcon/db/fulltext/compass/SchemaMappingBuilder.isAttributeIgnored()

Standardmässig ausgeschlossen werden die Daten von

- Relationen-Attributen (sowohl Single als auch Many)
- Attributen mit (Java-)Typ `Boolean`, `Date` oder `Number`
- virtuelle Attribute

Durch explizite Angabe im Schema ist es jedoch möglich, entsprechende Attribute von bestimmten Entitäten doch in den Index aufzunehmen. Andererseits können auch normalerweise indexierte Attribute explizit von der Indexierung ausgenommen werden.

Beispiel:

```
<Entity name="InterneEntitaet" extends="BO" plural="InterneEntitaeten">
  <attr name="Name">
  <attr name="InteressantesDatum" type="DateTime">
     <fulltext indexed="yes"/>
  </attr>
  <attr name="KryptischerString">
     <fulltext indexed="no"/>
  </attr>
</Entity>
```

Wird für Relationen-Attribute (sowohl Single- als auch Many-Relationen) `indexed="yes"` angegeben, ist das Resultat, dass Objekte der "Elter"-Klasse (die, die das Relationen-Attribut enthält) auch als Suchtreffer gefunden werden, wenn ein Suchbegriff "nur" auf eines der "Kind"-Objekte (die in der Relation enthaltenen Objekte) zutrifft.

Beispiel:

```
<Entity name="Elter" extends="BO" plural="Eltern">
  <attr name="Name">
  <attr name="Kinder" type="Kind" relation="1-n">
     <fulltext indexed="yes"/>
  </attr>
</Entity>

<Entity name="Kind" extends="BO" plural="Kinder">
  <attr name="Name">
</Entity>
```

Es existiert ein Elter "Elter1" mit Kindern "Kind1" und "Kind2". Wird jetzt z.B. im "Eltern"-Lesezeichen nach "Kind1" gesucht, so wird das Objekt "Elter1" als Ergebnis geliefert, obwohl der Suchbegriff "Kind1" eigentlich nur in einem der "Kinder"-Objekte vorkommt.

Weitere Einstellungen im Schema

analyzed

vgl. de/ipcon/db/fulltext/compass/SchemaMappingBuilder.buildScalarConfig() sowie das entsprechende Kapitel in der Compass-Dokumentation

Für einzelne Attribute kann im Schema definiert werden, ob die entsprechenden Inhalte bei der Indexierung "analysiert" werden sollen oder nicht.

Ein Text wie "Dies ist ein Attributwert" wird normalerweise nicht in dieser Form im Index abgelegt, sondern in seine einzelnen Bestandteile (normalerweise "Wörter", d.h. durch Whitespace abgetrennte Tokens) aufgeteilt. Auch werden einige sehr häufig vorkommende Wörter (sog. "Stopwords") entfernt.

Durch diese Behandlung ist es möglich, dass bei der Suche nach z.B. "Dies" das Objekt mit dem obigen Attributwert gefunden wird.

Wäre der Wert nicht "analysiert" worden, so wäre nur die gesamte Zeichenkette genau in dieser Form im Index abgelegt und das Objekt würde nur bei Eingabe genau von "Dies ist ein Attributwert" (oder ggf. noch bei Benutzung von Platzhaltern oder Ähnlichkeitssuche) gefunden, nicht aber nur bei Eingabe von "Dies".

Standardmässig werden alle Attribute mit (Java-)Typ `String` "analysiert"; alle anderen Attribute (insb. z.B. Zahlen) nicht. Im Normalfall ist diese Einstellung wohl sinnig; in Einzelfällen (z.B. vielleicht wenn es sich um eine Bezeichnung handelt, die nur genau in der eingegebenen Form gefunden werden soll) kann das Verhalten aber durch eine explizite Angabe beim Attribut geändert werden.

Beispiel:

```
<Entity name="InterneEntitaet" extends="BO" plural="InterneEntitaeten">
  <attr name="Name">
  <attr name="Typenbezeichnung">
     <fulltext analyzed="no"/>
  </attr>
</Entity>
```

boost

Sowohl für Entitäten als auch für einzelne Attribute kann ein "Boost"-Wert im Schema angegeben werden. Dieser dient dazu, einen Treffer für die entsprechende Entität oder das entsprechende Attribut höher oder niedriger zu bewerten und damit im Ranking der Suchergebnisse weiter nach vorne oder hinten zu plazieren. Da jedoch z.Zt. in MyTISM kein Ranking von Suchergebnissen benutzt wird, ist die Angabe dieses Wertes z.Zt. weder erforderlich noch sinnvoll. :toc: left :toc-title: Inhaltsverzeichnis :toclevels: 2 :icons: font

Formularelemente

Action

Name	Erlaubte Werte	Beschreibung
`acceleratorKey`	String: z. B. `"ENTER"`, `"control shift F5"`, ...	
`accKey`	siehe `acceleratorKey`	
`animation`	Boolean: **true**, false	Während die Action ausgeführt wird, eine Ladeanimation über das Formular legen.
`cmd`	String	Funktionsname, der z. B. von Buttons gerufen werden kann.
`contextMenu`	Boolean: true, **false**	
`formElementSync`	Boolean: **true**, false	
`icon`	String: z.Bsp. icon="20x20/New.gif", icon="image/remove_red_eye.svg" oder icon="image/remove_red_eye.svg@5085dc" (mit Farbangabe in hex; nur für SVGs verfügbar)	Pfad zum gewünschten icon.
`initialState`	Boolean: true, false	Wird zu Boolean Action mit dem angegebenen Anfangszustand. Dieser schaltet bei jeder Ausführung der Action um. Wird aktuell nur vom ToggleButton unterstützt.
`local`	Boolean: true, false	
`menu`	String	
`merge`	Boolean: true, **false**	Actions können zusammengeführt oder überschrieben werden.
`mnemonicKey`		
`name`	String	Name der Action. Optional - wenn nicht angegeben, gleich `cmd`. Per default Button-Titel.
`offEDT`	Boolean: true, **false**	Action in neuem Thread ausführen.
`priority`	int: **0**	
`progressShowDelay`	int: **1000**	
`restoreFocus`	Boolean: **true**, false	

Name	Erlaubte Werte	Beschreibung
`shortDescription`		Kurze Beschreibung. Wird als Tooltip angezeigt.
`showLabel`	Boolean: true, **false**	Den Namen der Action unterhalb eines ggf. vorhandenen Icons anzeigen.
`smallIcon`		
`toolBar`	Leerer String. Bsp.: toolBar=""	Wird hinzugefügt, falls die Action in der Standard-ToolBar neben einer Table bzw. im Falle von topMdiOnly in der "obersten" Toolbar erscheinen soll.
`topMdiOnly`	Boolean: true, false	Bei `true` wird die Action auf die oberste Toolbar gedrückt, d.h. die des Clients (bzw. im SDI/native window manager mode die des Objektfensters).

availableOn

Liefert ein hier angegebenes Skript `true` zurück, wird die Action angezeigt, bei `false` nicht.

> Die Bedingung für `availableOn` wird nur einmal und dann sehr früh evaluiert (bei der Entscheidung ob die Komponente überhaupt gebaut werden soll)

enabledOn

Die Action ist die ganze Zeit sichtbar und jenachdem ob ein hier angegebenes Skript `true` oder `false` zurückliefert, kann die Action angeklickt werden bzw. is ausgegraut.

> `enabledOn` wird laufend, d.h. bei jedem Statuswechsel, evaluiert (nicht wie bei `availableOn`).

initialState

Erfüllt die gleiche Aufgabe wie das Attribut `initialState`. Nur das hier ein Skript zur Bestimmung des Anfangszustandes angegeben wird.

longDescription

Wenn man mal mehr (formatierten) Text zur Erklärung der Action anzeigen möchte (auch HTML ist möglich):

```
<Action cmd="resetData" name="Daten resetten" shortDescription="Daten zurueckcksetzen"
toolBar="" accKey="control R">
  <onAction language="groovy">rootBO.doMagic()</onAction>
  <longDescription><![CDATA[<html>
    Folgende Voraussetzungen müssen erfüllt sein, damit die Daten der selektierten
Zeilen zurückgesetzt werden können:
    <ul>
      <li>Der Benutzer ist Mitglied der "Verwaltung" oder ein ADMIN</li>
      <li>Es ist mindestens eine Zeile ausgewählt</li>
    </ul>
  </html>]]></longDescription>
</Action>
```

onAction

Hier wird das Script definiert, welches für die jeweilige `Action` ablaufen soll.

BooleanInputComponent

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`class`		
`displayProperty` DEPRECATED		siehe `property`
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`text`		

Border

Attribute:

Name	Erlaubte Werte	Beschreibung
`bevel-highlight`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	
`bevel-highlightInner`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	
`bevel-highlightOuter`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	
`bevel-shadow`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	
`bevel-shadowInner`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	

Name	Erlaubte Werte	Beschreibung
`bevel-shadowOuter`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	
`bevel-type`	`LOWERED`, `RAISED`	Abschrägung.
`beveled`	`highlight`, `highlightInner`, `highlightOuter`, `shadow`, `shadowInner`, `shadowOuter`	
`debug`		
`editable`		
`etched-highlight`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	Hervorhebung. Muss immer zusammen mit `etched-shadow` verwendet werden.
`etched-shadow`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	Schattierung. Muss immer zusammen mit `etched-highlight` verwendet werden.
`etched-type`	String: **`LOWERED`**, `RAISED`	Begrenzung als Vertiefung oder Erhöhung darstellen.
`etched`	Boolean: true, **false**	
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	
`implied`		
`line-color`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	
`line`	Boolean: true, false	
`maximumSize`		
`maxSize`		
`minimumSize`		Alias. Siehe `minSize`

Name	Erlaubte Werte	Beschreibung
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	Gibt die minimale Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`missingPropertiesPolicy`		
`name`	String	Um das Form-Element innerhalb des Formulars referenzieren zu können
`preferredSize`		Alias. Siehe `prefSize`
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`title-justification`	String: **`LEFT`**, `CENTER`, `RIGHT`	
`title-position`	String: `NORTH`, `SOUTH`, **`WEST`**, `EAST`	
`title`	String	Überschrift für die durch die Border abgegrenzten Inhalte.
`toolBar-background`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	
`toolBar-floatable`		
`toolBar-layout`		
`toolBar-orientation`	String: `HORIZONTAL`, **`VERTICAL`**	
`toolBar-position`	String: `NORTH`, `SOUTH`, **`WEST`**, `EAST`	Wenn `toolBar-position` verwendet wird, wird `toolBar-orientation` automatisch gesetzt.
`toolBar-rollover`		

Name	Erlaubte Werte	Beschreibung
`toolBar`		ToolBar wird automatisch aktiv, wenn Elemente innerhalb der Border (z.Bsp. FTable) Actions für die ToolBar bereitstellen.
`topMdi`		

Button

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`action`	String	`cmd`-Attribut der Action, die bei Klick auf den Button ausgeführt werden soll.
`background`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	
`defaultButton`	Boolean: true, **false**	Falls **true**, wird der Button aktiviert, wenn der Container den Fokus hat und z.B. die ENTER Taste gedrückt wurde.
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	
`foreground`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	
`hAlign`		
`icon`	String: z.Bsp. icon="20x20/New.gif", icon="image/remove_red_eye.svg" oder icon="image/remove_red_eye.svg@5085dc" (mit Farbangabe in hex; nur für SVGs verfügbar)	Pfad zum gewünschten icon.
`multiClickThreshold`	Integer (750ms)	Spezifiziert innerhalb welchen Zeitraums mehrfaches Klicken des Buttons als ein einfaches Klicken interpretiert wird
`text`	String	Beschriftung des Buttons.

Name	Erlaubte Werte	Beschreibung
`vAlign`	String: TOP, CENTER, BOTTOM, z. B. vAlign="TOP"	Bestimmt die vertikale Ausrichtung des Textes innerhalb des Elements. Die Höhe des Elements muss größer sein als eine normale Zeilenhöhe, was z. B. durch Setzen des Attributs prefSize erreicht werden kann.

Canvas

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`toolTipText`	String.	Text, der angezeigt wird, wenn man den Mauszeiger über das Element hält.

Chart

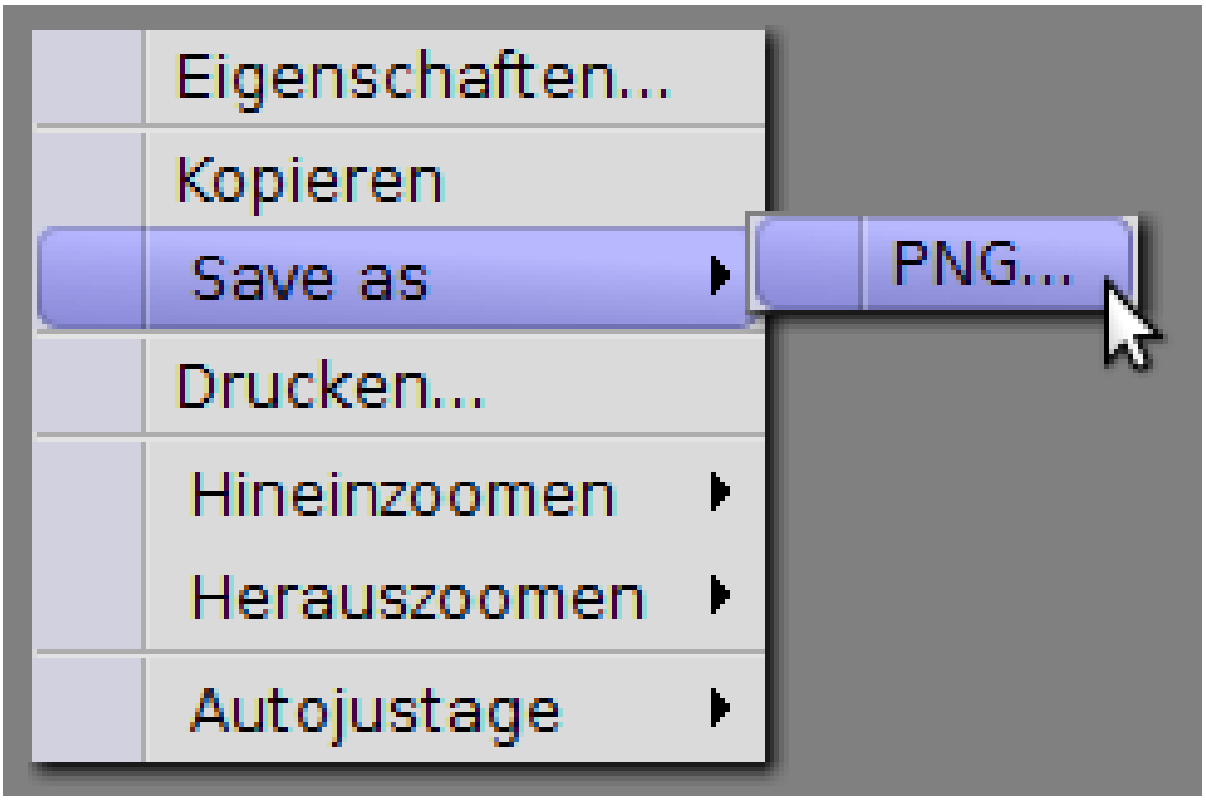
Komponente zur Darstellung von Diagrammen in Formularen. Die zugrundeliegende Bibliothek ist "JFreeChart".

Derzeit werden von unserer direkten Implementierung folgende Diagramme unterstützt:

- timeSeriesChart
- stackedXYAreaChart
- scatterPlot
- xYAreaChart
- xYLineChart
- xYStepAreaChart
- xYStepChart

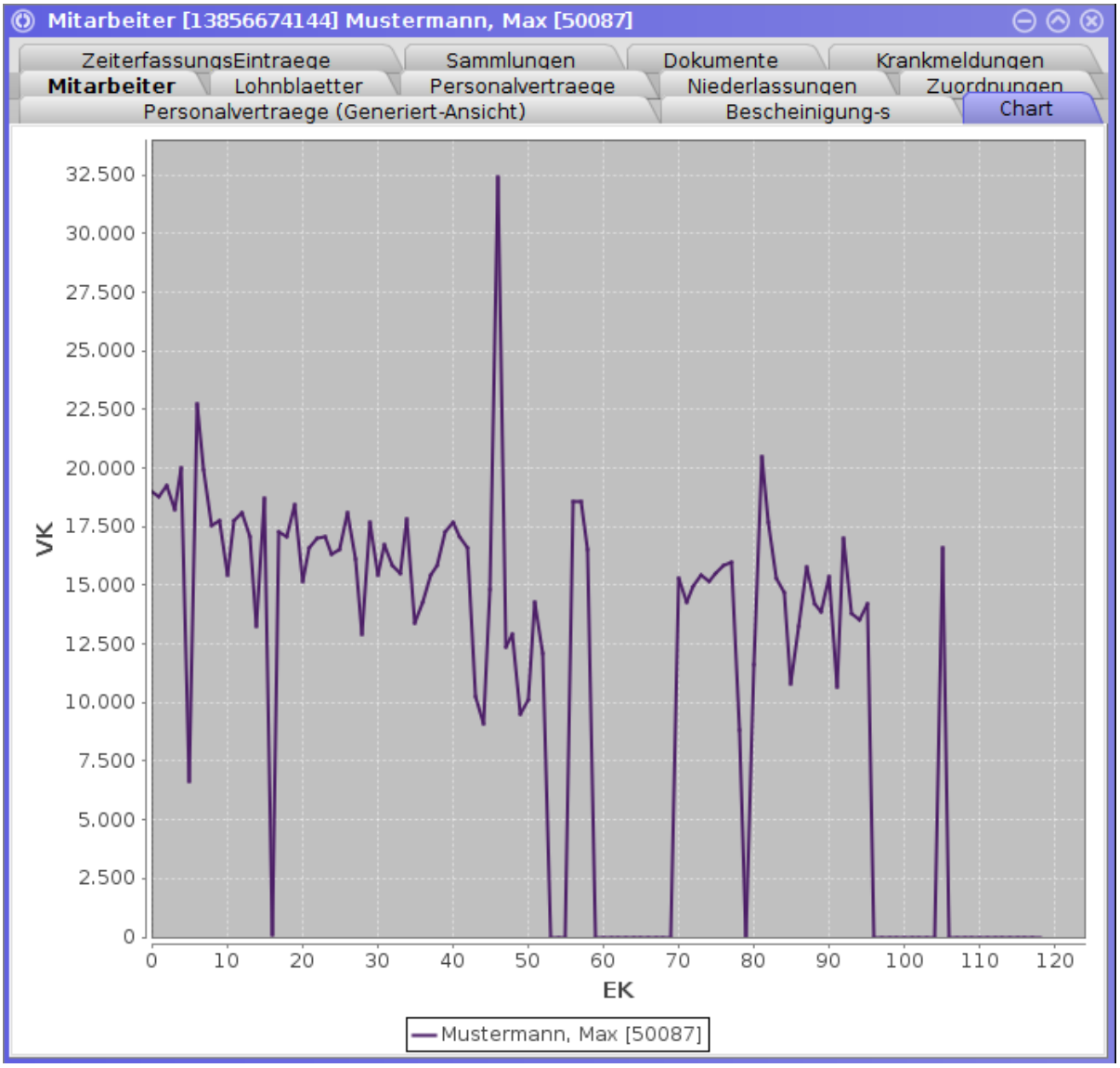
Darüber hinaus kennt JFreeChart noch einige andere Diagramme (z.B. Torten-Diagramme).

Über das Kontextmenü stehen weitere Funktionen wie Export als Bild, Drucken, Einstellungsmöglichkeiten, etc zur Verfügung.



Beispiel unter Verwendung unserer direkten Implementierung:

Das folgende Beispiel dient der Erklärung des Aufbaus. Es werden die ZeiterfassungsEinträge eines Mitarbeiters dargestellt, mit der jeweiligen Anwesenheitsdauer in y-Richtung.
Der JFreeChartBuilder wird als `builder` an das buildScript übergeben. Über diesen können dann die verschiedenen Diagramme gezeichnet werden. Innerhalb der Closure steht das `anchor`-Objekt zur Verfügung, welches das im Formular geöffnete BO referenziert.


Mitarbeiter [13856674144] Mustermann, Max [50087]
ZeiterfassungsEintraege
Sammlungen
Dokumente
Krankmeldungen
Mitarbeiter
Lohnblaetter
Personalvertraege
Niederlassungen
Zuordnungen
Personalvertraege (Generiert-Ansicht)
Bescheinigung-s
Chart
VK
32.500
30.000
27.500
25.000
22.500
20.000
17.500
15.000
12.500
10.000
7.500
5.000
2.500
0
0
10
20
30
40
50
60
70
80
90
100
110
120
EK
Mustermann, Max [50087]

```
<Chart>
  <buildScript><![CDATA[
  builder.xYLineChart(title:'oneTitle', xAxisLabel:"EK", yAxisLabel:'VK') {
  def zes = anchor.ZeiterfassungsEintraege
  antiAlias=true
  borderVisible=false
  borderPaint='#c0c0c0'
  plot {
    tableXYDataset() {
      rows = {
        (0..zes.size()-1).each{ it }
      }
      x = {
        it
      }
      series(name: anchor.kontakt.describe()) {
        values = {
          zes.values().getAt(it).Anwesenheitsdauer
        }
        stroke = 2
        paint = '#4c1e67'
      }
    }
  }
}
  ]]></buildScript>
</Chart>
```

Beispiel der generischen Verwendung (Prüfserie hat Einzelprüfungen mit deren Messwerten):

```
<Chart>
  <buildScript><![CDATA[
    import org.jfree.chart.ChartFactory
    import org.jfree.chart.ChartPanel
    import org.jfree.chart.JFreeChart
    import org.jfree.chart.plot.PlotOrientation
    import org.jfree.data.category.DefaultCategoryDataset

    def pserie = builder.anchor

    // dataset definieren
    def dataset = new DefaultCategoryDataset()
    pserie.getEinzelpruefungen().values().sort{ it.getPosition() }.each{
      dataset.addValue(it.getFeinheitNm(), 'FeinheitNm', it.getPosition())
      dataset.addValue(it.getElastizitaet(), 'Elastizitaet', it.getPosition())
      dataset.addValue(it.getHeissluftschrumpfS130(), 'HLS130', it.getPosition())
    }

    // Chart generieren
    JFreeChart chart = ChartFactory.createLineChart(
            'Messwerte der Einzelpruefungen',
            'Position','Skala',
            dataset,
            PlotOrientation.VERTICAL,
            true, true, false)
return chart
  ]]></buildScript>
</Chart>
```

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`closed`	Boolean: true, **false**	
`print`	Boolean: **true**, false	
`property`		
`props`	Boolean: **true**, false	
`save`	Boolean: **true**, false	
`toolTips`	Boolean: **true**, false	
`zoom`	Boolean: **true**, false	

Subelemente

onClick

Script, an das vorab aufbereitete Click-Events auf Elemente innerhalb der Chart weitergeleitet werden.

Existiert eine `onClick` subnode, werden klickbare Elemente innerhalb der Chart zudem gehighlighted, damit der Benutzer einen visuellen Hinweis hat, dass eine Interaktion möglich ist.

Verfügbare Variablen:

- `row`, `column` : Keys bzw. Labels, welches die Identifizierung einer Bar in einer BarChart ermöglichen
- `section` : Key bzw. Label, welches die Identifzierung eines Abschnitts in einer Ring- oder PieChart ermöglicht.

Um etwas mit den Daten zu tun, die zu dem angeklickten Element gehören, empfiehlt es sich, eine Map mit den Keys und den Daten im Kontext-Binding des Groovy-Scripts zu hinterlegen (z.Bsp. im `onConstruction` der umgebenden View).

Alternativ können die keys / labels aber auch als Filter interpretiert werden, und als Query in einem zu öffnenden Lesezeichen gesetzt werden (als Parameter mit Namen `query` beim Öffnen des LZ übergeben), siehe Beispiel unten.

```
<Chart>
  <buildScript>[ ... ]</buildScript>
  <onClick><![CDATA[
    import de.ipcon.tools.date.DateTimeTools

    def bkm = ctx.getBOLoader().getBOByAttr(Lesezeichen, 'Tid', 'MCS_Rechnungen')
    // re-construct the month range from the column label
    def month = DateTimeTools.getFirstDayOfMonth(L10n.parseDate(column, 'MM/yy'),
true)
    def presetQuery = "[Belegdatum >= '${L10n.formatISODate(month)}'"

    ctx.openView(bkm, [query: presetQuery.toString()])
  ]]></onClick>
</Chart>
```

CheckBox

Die Checkbox kann benutzt werden, um Boolean-Werte zu beeinflussen.

Attribute:

Name	Erlaubte Werte	Beschreibung
`class`		
`debug`		
`displayProperty` DEPRECATED		siehe `property`
`editable`	Boolean: **true**, false	false verhindert das Editieren des Feldes. Das Feld WIRD NICHT ausgegraut.
`implied`		
`maximumSize`		
`maxSize`		
`minimumSize`		Alias. Siehe `minSize`
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	Gibt die minimale Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`missingPropertiesPolicy`		
`name`	String	Um das Form-Element innerhalb des Formulars referenzieren zu können
`preferredSize`		Alias. Siehe `prefSize`
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.

Name	Erlaubte Werte	Beschreibung
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`text`	String	Text, der zusätzlich hinter der Checkbox angezeigt wird.
`triState`	Boolean: **true**, false	Wenn hier true gesetzt ist, kann das Element drei Zustände annehmen: true, false und null. Ansonsten nur true und false. Wenn false, kann Null nicht mehr ausgewählt werden. Kann durch eine Angabe im Schema bereits definiert werden.

ComboBox

Die API-Dokumentation enthält ebenfalls weitere Informationen zu dieser Komponente.

Wie in dem Bereich über Widgets bereits gezeigt, kann eine ComboBox direkt auf ein property zeigen, woraufhin alle möglichen BOs angezeigt werden.

Beispiel:

Will man programmatisch eigene Auswahlmöglichkeiten anbieten kann man das über das choiceSkript machen:

```
...
<ComboBox property="FilterZeitraum" e-label="$R{Zeitraum}" chooseOnly="true"
nullable="false">
  <choiceScript>
    // Title: Value
    ['letzten 2 Tage'    : '2Tage',
     'letzten 7 Tage'    : '7Tage',
     'letzten 30 Tage'   : '30Tage',
     'letzten 60 Tage'   : '60Tage',
     'letzten 100 Tage'  : '100Tage',
     'letztes halbe Jahr': 'HalbesJahr',
     'letztes Jahr'      : 'LetztesJahr',
     'Alle'              : 'Alle']
  </choiceScript>
</ComboBox>
...
```

Attribute

`Name`	**Erlaubte Werte**	**Beschreibung**
`autoSelect`	String: first/last/firstNonNull	Gibt an welches Element als default ausgewählt werden soll.
`chooseOnly`	Boolean: true, **false**	Wenn true, dürfen keine eigenen Werte in das Feld eingetragen werden.
`displayFormat` DEPRECATED		siehe `format`
`displayProperty` DEPRECATED		siehe `property`
`format`	String (CBOFormat).	Legt fest, wie das BO für die Anzeige im Dropdown-Menü und im Feld formatiert ist. Bsp: "Id': 'Name"

Name	Erlaubte Werte	Beschreibung
`nullable`	Boolean: **true**, false	Wenn false wird null nicht als mögliche Auswahl angeboten.
`nullChoiceTitle`	String	Wird angezeigt, wenn nichts ausgewählt wurde. Gilt sowohl für das Feld als auch für den Eintrag im Dropdown-Menü, der den leeren Wert repräsentiert.
`property`		
`relationName` DEPRECATED		
`selectEntity`	String	Name der Entität, deren Objekte hier zur Auswahl angeboten werden sollen.
`selectOutOf`	String	Name der Relation, aus der Objekte zur Auswahl angeboten werden sollen.
`sortBy`	String	Name des Attributes, nach dem sortiert werden soll.
`whereClause`	String (OQL)	Optionaler Filter für im Attribut `selectEntity` gesetzte Entität im OQL-Format.
`showId`	Boolean: true, **false**	Wenn true, wird die Id jedes (BO-)Elements in eckigen Klammern hinter dem Element angezeigt.
`suppressDuplicatesInNonRelationMode`	Boolean: true, **false**	Wenn true, werden für Comboboxen, die keine Relation anzeigen, Duplikate in der Auswahlliste verhindert, indem die Id jedes (BO-)Elements in eckigen Klammern hinter dem Element angezeigt wird, sofern Duplikate auftreten.

DateChooser

Attribute:

Name	Erlaubte Werte	Beschreibung
`autoHideButton`		
`columns`		
`debug`		
`displayFormat` DEPRECATED		siehe `format`
`displayProperty` DEPRECATED		siehe `property`
`editable`	Boolean: **true**, false	false verhindert das Editieren des Feldes. Das Feld WIRD NICHT ausgegraut.
`enabled`	Boolean: **true**, false	
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	
`format`	String: `LONG_`, `MEDIUM_`, `SHORT_`, `dd/MM/YYYY`	Bestimmt die Formatierung des angezeigten Datums. Beispiele: `LONG_`: 7. September 2016 `MEDIUM_`: 07.09.2016 `SHORT_`: 07.09.16
`implied`	Boolean: true, false	
`maximumSize`		
`maxSize`		
`minimumSize`		Alias. Siehe `minSize`
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	Gibt die minimale Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`missingPropertiesPolicy`		
`name`	String	Um das Form-Element innerhalb des Formulars referenzieren zu können.

Name	Erlaubte Werte	Beschreibung
`popupHeight`	Integer mit Einheit px, c, em oder dlu	Die Angabe in Character "c" ist die zu favorisierende Bestimmt die Höhe des Fensters, in dem der Kalender zur Datumsauswahl angezeigt wird.
`popupWidth`	Integer mit Einheit px, c, em oder dlu	Die Angabe in Character "c" ist die zu favorisierende Bestimmt die Breite des Fensters, in dem der Kalender zur Datumsauswahl angezeigt wird.
`preferredSize`		Alias. Siehe `prefSize`
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`selectAllWhenFocused`	Boolean: true, *false*	Wenn das Form-Element den Fokus bekommt wird der gesamte Inhalt selektiert

Editor

Der Editor kann für String-Attribute angezeigt werden. Durch die Angabe des Modus kann die Darstellungsart beeinflusst werden. Es werden dazu intern JEdit-Klassen benutzt.

Beispiel:

```
...
<Editor property="Bemerkung" mode="patch"/>
...
```

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`columns`	int: **60**	Gibt an wieviele Spalten angezeigt werden sollen.
`electricScroll`	int: **3**	Wird der Cursor an eine bestimmte Stelle bewegt, wird sichergestellt, dass x Zeilen über & unter der Position angezeigt werden.
`focusable`	Boolean: **true**, false	Ist focusable false, kann das Textfeld nicht fokussiert werden...
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`maxUndos`	int: **500**	Gibt die Anzahl der gespeicherten Schritte und damit der möglichen Undos an
`mode`	String: **xml**, groovy, log, patch	Definiert die Formatierung des Editor-Fensters.
`rows`	int: **10**	Gibt an wieviele Zeilen angezeigt werden sollen.
`text`	String	Füllt das Attribut automatisch mit dem eingegebenen Text

Element

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`autoCreate`	Boolean: true, false	
`autoHide`	Boolean: true, false	
`displayProperty` DEPRECATED		siehe `property`
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	
`hideForNullBO`	Boolean: **true**, false	Bei **true** wird das eingebettete GUI-Element nicht angezeigt, wenn das BO des angegebenen Attributs null ist, ansonsten ist es ausgegraut. Das ist nützlich im Fall von Attributketten, z.Bsp. bei `property="Person.Name"` wird über das Flag gesteuert, ob das Eingabefeld des Namens angezeigt wird, selbst wenn `Person` nicht gesetzt ist. Das Setzen auf **false** ist ratsam, wenn durch das Wegfallen des Eingabefelds die Anordnung anderer Elemente ungewollt beeinflusst wird.
`hSpaceDist`	Double **-1**	Was soll mit dem restlichen Platz von rightFill geschehen, sofern != 0 (default: FView.defaultCellHAlign); Beispiel: siehe ipcon/db/core/Report.nrx Beschriftung, die vor dem Inhalt des Element Tags angezeigt wird.
`label`	String	Beschriftung, die vor dem Inhalt des Element Tags angezeigt wird.

Name	Erlaubte Werte	Beschreibung
`labelBackground`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	Bestimmt die Hintergrundfarbe des Labels.
`labelForeground`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	Bestimmt die Schriftfarbe des Labes.
`mode`	String: `FREE_FIELD`, `FREE_LABEL`, `LABEL_ON_TOP`, `DEFAULT`	`FREE_FIELD`: Das im Element beinhaltete Feld füllt den gesamten Raum zwischen Ende des Labels und rechtem Rand des Elternelements. `FREE_LABEL`: Das längste Label innerhalb innerhalb eines gemeinsamen Elternelements bestimmt die Breite der "Labelspalte". Alle Labels und die beinhalteten Felder werden am rechten Rand des längsten Labels ausgerichtet. `LABEL_ON_TOP`: Das Label wird über dem beinhalteten Feld angezeigt. `DEFAULT`: siehe `FREE_LABEL`.
`no-label`	Boolean: true, **false**	

Name	Erlaubte Werte	Beschreibung
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt. Nähere Informationen hierzu im GUI-Kochbuch
`rightFill`		
`rows`		
`transactionControl`		
`x`		
`y`		

Email

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`align`	String: `LEFT`, `CENTER`, `RIGHT`, `LEADING`, `TRAILING`	
`class`		
`columns`		
`debug`		
`disabled`	Boolean: true, **false**	false verhindert das Editieren des Feldes. Das Feld WIRD ausgegraut.
`displayFormat` DEPRECATED		siehe `format`
`displayProperty` DEPRECATED		siehe `property`
`editable`	Boolean: true, **false**	false verhindert das Editieren des Feldes. Das Feld WIRD NICHT ausgegraut.
`font`		
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	
`foreground`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	

Name	Erlaubte Werte	Beschreibung
`format`		
`implied`		
`lineWrap`	Boolean: **true**, false	Bricht Text an der rechten Kante um, statt eine Scrollbar anzuzeigen.
`maxSize`		
`maximumSize`		
`minimumSize`		Alias. Siehe `minSize`
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	Gibt die minimale Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`missingPropertiesPolicy`		
`name`		
`password`	Boolean: true, **false**	Passwortfeld statt normalem Text.
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`preferredSize`		Alias. Siehe `prefSize`
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`roundingFormat`		
`rows`	int: **4**	

Name	Erlaubte Werte	Beschreibung
`selectAllWhenFocused`	Boolean: true, *false*	Wenn das Form-Element den Fokus bekommt wird der gesamte Inhalt selektiert
`syncOnWait`		
`syncOnWaitDelay`		
`tabSize`	int: **4**	
`translationAvailable`		
`wrapStyleWord`	Boolean: **true**, false	Wenn Text umgebrochen wird, dann möglichst an Whitespace-Grenzen.

Image

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`displayProperty` DEPRECATED		siehe `property`
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`scaleToFit`	Boolean: true, **false**	

Label

Name	Erlaubte Werte	Beschreibung
`asyncRefresh`	Boolean: true, false, **null**	Explizit synchrones oder asynchrones Verhalten erzwingen. Synchrones Verhalten ist hilfreich, falls das Label variierenden Platzbedarf hat. (Üblicherweise bei mehrzeiliger HTML-Ausgabe.)
`background`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'
`class`		
`clickable`	true, **false**	Wenn `true` und es ist eine `property` gesetzt, die auf ein BO zeigt, wird dieses BO in einem Formular geöffnet.
`disabledIcon`		
`displayFormat` DEPRECATED		siehe `format`
`displayProperty` DEPRECATED		siehe `property`
`font`		
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	

Name	Erlaubte Werte	Beschreibung
`foreground`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'
`format`	String (CBOFormat). Bsp.: `property="." format="'Auflage: 'Auflage.Nr"`	Ermöglicht dynamisches Erzeugen des Labeltextes durch CBOFormat. Voraussetzung, um, wie im Beispiel, auf einen Attributswert zugreifen zu können, ist es, dass die Property gesetzt ist (siehe `property`).
`gradientStartColor`	Farbangabe. Bsp: `gradientStartColor="160 160 255"`	Hinterlegt das Label mit einem Farbverlauf von links nach rechts. Kann zusammen mit `gradientStopColor` verwendet werden.
`gradientStartPosition`	String: `NORTH`, `SOUTH`, `WEST`, `EAST`	Verlegt den Anfang des Farbverlaufs.
`gradientStopColor`	Farbangabe. Bsp: `gradientStopColor="160 160 255"`	Hinterlegt das Label mit einem Farbverlauf von rechts nach links. Kann zusammen mit `gradientStartColor` verwendet werden.
`gradientStopPosition`	String: `NORTH`, `SOUTH`, `WEST`, `EAST`	Verlegt das Ende des Farbverlaufs.
`hAlign`	String: `LEFT`, `CENTER`, `RIGHT`	Horizontale Ausrichtung.
`hTextPosition`	String: `LEFT`, `CENTER`, `RIGHT`, `LEADING`, `TRAILING`	

Name	Erlaubte Werte	Beschreibung
`html`	Boolean: true, **false**	Erlaubt es, im `text`-Attribut HTML zu benutzen, ohne auf spitze Klammern verzichten zu müssen. Enthält das html eine URL (`<a href="` ...), wird das Label implizit `clickable`, und die URL wird bei Klick evaluiert und geöffnet.
`icon`	String: z.Bsp. icon="20x20/New.gif", icon="image/remove_red_eye.svg" oder icon="image/remove_red_eye.svg@5085dc" (mit Farbangabe in hex; nur für SVGs verfügbar)	Pfad zum gewünschten icon. Das icon erscheint vor dem in `text` angegebenen Text.
`iconColor`	Farbangabe, z.Bsp. #ffffff	Ermöglicht es, dem Icon explizit eine Farbe zu geben, vorausgesetzt es handelt sich um ein SVG icon.
`iconTextGap`	Bsp: `iconTextGap="10"`	Angabe des Abstandes zwischen dem Icon und Text (als Ganzzahl)
`openProperty`		Ermöglicht es, eine andere Property des angezeigten BOs zu öffnen als angezeigt bzw. formattiert wird. Bei Benutzung dieses Attributs wird `clickable` automatisch auf **true** gesetzt.
`padding`	Vier positive ganze Zahlen, getrennt durch Komma oder Leerzeichen, die den jeweiligen Abstand in Pixeln zum oberen, linken, unteren und rechten Rand spezifizieren: `padding="20,30,20,30"`	Außenabstand in Pixeln.
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.

Name	Erlaubte Werte	Beschreibung
`text`	String	Text des Labels. Kann HTML beinhalten, wenn `html` Attribut `true` ist.
`textWhileLoading`		
`toolTipText`	String.	Text, der angezeigt wird, wenn man den Mauszeiger über das Element hält.
`vAlign`	String: TOP, CENTER, BOTTOM, z. B. vAlign="TOP"	Bestimmt die vertikale Ausrichtung des Textes innerhalb des Elements. Die Höhe des Elements muss größer sein als eine normale Zeilenhöhe, was z. B. durch Setzen des Attributs prefSize erreicht werden kann.
`vTextPosition`	String: `TOP`, `CENTER`, `BOTTOM`	

FPanel (abstrakt)

Superklasse für viele Elemente. Bringt die unten stehenden Attribute und Sub-Elemente mit.

Skriptvariablen

In den Skripten sind diese Bindings verfügbar:

Variablenname	Klasse/Interface	Definition	Beschreibung
`ctx`	ClientContextI	ftx.getCtx()	Client-weiter Kontext, wird z.Bsp. zum Öffnen von Dialogen oder Formularen benutzt
`user`	Benutzer	ftx.getCtx().getCurrentUser()	der im Client angemeldete Benutzer
`ftx`	FormContextI	ftx	Kontext des Formulars, in dem das FPanel eingesetzt ist. Wird gebraucht, um andere Formularelemente anzusprechen.
`bo`	BO	ftx.getBO()	Das zugrundeliegende BO des FPanels. (Kann vom rootBO abweichen)
`tx`	Transaction	ftx.getRoot().getTransaction()	Die Transaction, mit der das rootBO geladen wurde, wenn `bol instanceof Transaction`
`fe`	FormElementI	fe	Das FPanel (als FormElementI)
`bol`	BOLoaderI	ftx.getRoot().getBOLoader()	Der Loader, mit dem die Daten des Strukturelements geladen wurde. Im Fall von Formularen gilt: `bol instanceof Transaction`
`rootBO`	BO	ftx.getRoot().getBO()	Das zugrundeliegende BO des Formulars.

Subelemente

onAfterSelectValue

Ermöglicht es, ein Skript zu definieren, das nach dem Setzen eines Werts ausgeführt wird.

```
<Text property="Beschreibung">
  <onAfterSelectValue
language="groovy">ftx['v_lbl_beschreibung_fehlt'].ftx.refreshForms()</onAfterSelectVal
ue>
</Text>
```

Attribute

Name	Erlaubte Werte	Beschreibung
`language`	String: groovy, **beanshell**	Die zu verwendende Skriptsprache.

editableIf

Ergebnis des Ausdrucks (hier `bo.kannEditiertWerden`) bestimmt, ob das Elternelement editiert werden kann.

```
<Text property="Beschreibung">
  <editableIf language="groovy">bo.kannEditiertWerden</editableIf>
</Text>
```

Attribute

Name	Erlaubte Werte	Beschreibung
`language`	String: groovy, **beanshell**	Die zu verwendende Skriptsprache.

visibleIf

Ergebnis des Ausdrucks (hier `bo.istSichtbar`) bestimmt, ob das Elternelement angezeigt wird.

```
<Text property="Beschreibung">
  <visibleIf language="groovy">bo.istSichtbar</visibleIf>
</Text>
```

Attribute

Name	Erlaubte Werte	Beschreibung
`leftEntity`	String: Entitätsname	Die Entität, an der das `property` des Elternelements hängt, muss eine Subklasse der hiermit angegebenen Entität sein. Kann Skript-Inhalt dieses Tags ersetzen oder ihn als Konjunktion ergänzen.
`language`	String: groovy, **beanshell**	Die zu verwendende Skriptsprache.
`leftClass`	DEPRECATED	siehe `leftEntity`
`never`	Boolean: true, **false**	Wenn **true**, wird das Elternelement niemals angezeigt.
`notForLeftEntity`		
`notForRightEntity`		
`property`		
`rightEntity`		

OnDrop

OnDrop ist ein Tag, dass benutzt werden kann um Dateien zu behandeln die auf das Element gedropt wurden:

Beispiel:

```
<onDrop language="groovy">
  def result = Datei.importFileFromDnD(ftx, tx, files, rootBO)
  ftx.toast("${result.size()} Dateien importiert.")
  ftx.getRoot().refreshForms()
</onDrop>
```

Attribute: FIXME

onFocusGained

Eventhandler. Wird beim Setzen des Eingabe-Focus auf das Elternelement ausgeführt.

```
<Text property="Beschreibung">
  <onFocusGained language="groovy">ftx.toast('Das Textfeld "Beschreibung" hat jetzt
den Fokus.')</onFocusGained>
</Text>
```

onFocusLost

Eventhandler. Wird ausgeführt, wenn das Elternelement den Fokus verliert.

```
<Text property="Beschreibung">
  <onFocusLost language="groovy">ftx.toast('Tschüss!')</onFocusLost>
</Text>
```

onRefresh

Eventhandler. Wird beim Übertragen von Model-Daten in die View aufgerufen.

```
<Text property="Beschreibung">
  <onRefresh language="groovy">ftx.toast('Lade Beschreibung.')</onRefresh>
</Text>
```

onSync

Eventhandler. Wird beim Übertragen von View-Daten ins Model aufgerufen.

```
<Text property="Beschreibung">
  <onSync language="groovy">ftx.toast('Speichere Beschreibung.')</onSync>
</Text>
```

FInputPanel (abstrakt)

Leitet sich von FPanel ab.
Alle Komponenten, die einen Input erwarten leiten sich daraufhin von FInputPanel ab.

Zentralisiert die Mandatory-Berechnung, sodass alle Subklassen ein gleiches Verhalten besitzen. Außerdem wird noch ein neues Skript-Tag - `alsoMandatoryIf` - hinzugefügt.

alsoMandatoryIf

Kann benutzt werden um ein Eingabeelement zum Pflichtfeld zu machen.

```
<alsoMandatoryIf language="groovy">
   ctx.currentUser.istMitgliedVon("Pflichtgruppe")
</alsoMandatoryIf>
```

Per default ist das angegebene Skript gecached (wird also nur beim Öffnen des Formulars ausgewertet), da man von komplexerem Code ausgegangen ist.

Möchte man aber, dass bei einem Refresh das Skript erneut evaluiert wird, kann man das Caching mittels `cached="false"` deaktivieren.

```
<alsoMandatoryIf cached="false" language="groovy">
   !rootBO.KundeWillDatenNichtNennen
</alsoMandatoryIf>
```

Falls das Schema vorgibt, dass ein Element mandatory ist, kann dies NICHT über `alsoMandatoryIf` ausgehebelt werden.

PDFViewer

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`annotationColorEdit`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH Standard: **#000000**	
`annotationColorSaved`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH Standard: **FF0000**	
`annotationFont`	String: **Helvetica-18**	
`bufferedImage`	Boolean: true, **false**	**false** = ein volatiles Bild wird direkt in den Grafik(karten)-Speicher geladen, was je nach Treiber(/Karte/System) länger dauern kann. **true** = ein gepuffertes (aber speicher-intensiveres) Bild wird in den Grafik(karten)-Speicher geladen. Diese Vorgehensweise ist mitunter unter Windows etwas schneller
`e-no-label`	Boolean: true, **false**	Unterdrückt die Anzeige des zugehörigen Labels
`enableAnnotations`	Boolean: true, **false**	
`fitOnPage`	Boolean: true, **false**	
`fitWidth`	Boolean: **true**, false	
`forceAntialiasing`	Boolean: **true**, false	
`lowQuality`	Boolean: true, **false**	
`onAnnotationAdded`		
`onAnnotationChanged`		
`onAnnotationRemoved`		
`onAnnotationSelected`		

Name	Erlaubte Werte	Beschreibung
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`scaleBicubic`	Boolean: true, **false**	
`scaleToFit`	Boolean: true, **false**	Alias für `fitOnPage`.
`zoomValue`	Double: **0.05**	

Popup

Ein Popup-Element mit einem Eingabefeld und einem weißen Blatt-Icon auf der rechten Seite zeigt an, dass Sie hier ein bereits vorhandenes Objekt auswählen und verknüpfen können. Wenn daneben ein gelber Stift erscheint, können Sie an dieser Stelle direkt ein neues Objekt erstellen. Das Doppelblatt-Icon kopiert den aktuellen Inhalt in ein neues Objekt und verknüpft dieses. Diese Funktion ist hilfreich, wenn das ursprüngliche verknüpfte Objekt unverändert bleiben soll, Sie jedoch an dieser Stelle eine angepasste Version benötigen.

Attribute:

Name	Erlaubte Werte	Beschreibung
`align`	String: `LEFT`, `CENTER`, `RIGHT`, `LEADING`, `TRAILING`	

Name	Erlaubte Werte	Beschreibung
`autoEdit`	Boolean: true, **false**	Attribute in der Detailview eines Popup Elements sind normalerweise schreibgeschützt, d.h. Wert 'false'. Grund ist, dass ein Popup mit Detailview oft allgemeinere Informationen im Formular anzeigt, das BO des Popups jedoch nicht nur von dem aktuellen `RootBO` abhängt. Eine Änderung eines Attributs im Kontext dieses RootBOs könnte demnach nicht allgemein genug sein um überall zu passen. Dies hindert den Benutzer daran, ausversehen einen solchen Wert zu ändern. Es gibt jedoch Fälle, in denen es im Formular explizit gewünscht ist, die Attribute ändern zu dürfen. Zum Beispiel wenn das BO 'dependent' vom RootBO ist. In diesem Fall kann dem Benutzer das Editieren per Definition von `autoEdit="true"` erlaubt werden. Soll der Benutzer die Attribute des BOs ändern dürfen, jedoch das eigentliche BO nicht sehen können, dann ist eine Umsetzung über eine Attributkette in der property-Definition bzw. ein Wrapper-`<Element..></Element>` vielleicht sinnvoller.
`columns`		
`debug`		
`displayFormat` DEPRECATED		siehe `format`
`displayFormatDivider`		
`displayFormatPostfix`		
`displayFormatPrefix`		
`displayProperty` DEPRECATED		siehe `property`
`displaySort`		

Name	Erlaubte Werte	Beschreibung
`editable`	Boolean: **true**, false	`false` verhindert Editieren des Feldes. Das Feld wird ausgegraut.
`fallBackProperty`		
`font`		
`fontsize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	
`foreground`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'
`format`	String (CBOFormat).	Legt fest, wie das BO für die Anzeige im Feld formatiert ist. Bsp: "Id': 'Name"
`implied`		
`lazy`		
`lookupCaseSensitive`	Boolean: true, **false**	Definiert ob die Eingabe mit oder ohne Berücksichtigung der Gross-/Kleinschreibung gesucht werden soll

Name	Erlaubte Werte	Beschreibung
`lookupProperty`	String	Komma-separierte Liste von Attributen/Attribut-Ketten, die bei der "Schnelleingabe" durchsucht werden sollen; bei einem eindeutigen Ergebnis wird automatisch selektiert
`lookupStartingWith`		
`lookupSubstring`	Boolean: **true**, false	
`maximumSize`		
`maxSize`		
`minimumSize`		Alias. Siehe `minSize`
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	Gibt die minimale Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`missingPropertiesPolicy`		
`name`	String	Um das Form-Element innerhalb des Formulars referenzieren zu können
`nullChoiceTitle`		
`offerCopyBeforeEdit`		
`openFormTid`	String	Tid des Formulars, das benutzt werden soll, um Objekte aus diesem Popup zu öffnen.
`openProperty`		
`popupAlign`		
`popupHeight`	Integer mit Einheit **px**, c, em oder dlu. Bsp.: popupHeight="40c"	Die Höhe des resultierenden Popups. Wird keine Einheit angegeben, wird von einem px-Wert ausgegangen.

Name	Erlaubte Werte	Beschreibung
`popupSize`	Komma-separierte Integer mit Einheit **px**, c, em oder dlu. Bsp.: popupSize="960, 20c"	Kurzform für `popupWidth` und `popupHeight`. Der erste Wert bestimmt die Breite, der zweite Wert die Höhe. Einheiten können gemischt angegeben werden. Wird keine Einheit angegeben, wird von einem px-Wert ausgegangen.
`popupWidth`	Integer mit Einheit **px**, c, em oder dlu. Bsp.: popupWidth="40c"	Die Breite des resultierenden Popups. Wird keine Einheit angegeben, wird von einem px-Wert ausgegangen.
`preferredSize`		Alias. Siehe `prefSize`
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`showEntityName`	Boolean: **true**, false	
`showNewAction`	Boolean: true, **false**. Bsp.: `showNewAction="true"`	Bestimmt, ob im Feld ein Icon (Blatt mit Stern) angezeigt wird, das beim Anklicken ein neues Fenster zum Erstellen eines neuen Objektes für die jeweilige Relation öffnet.
`showSelectAction`	Boolean: **true**, false. Bsp.: `showSelectAction="false"`	Bestimmt, ob im Feld ein Icon (Blatt Papier) angezeigt wird, das beim Anklicken eine Tabelle bereits existierender Objekte öffnet, die für die Relation ausgewählt werden können.
`subentitiesToExclude`		

Name	Erlaubte Werte	Beschreibung
`templateSource`		
`usePolymorphySelectionTree`		

Scheduler

Der Scheduler ist eine spezielle Komponente, die es ermöglicht, Objekte, die eine Zeitspanne darstellen und zu einer Resource zugeordnet werden, zu planen und zu verwalten.

Denkbare Einsatzgebiete wären: *Dienst- und Schichtplanung, Produktionsplanung, allgemeine Terminplanung*

Die minimale Konfiguration des Schedulers benötigt die Attribute `property`, `itemClass` und ein `dataMapper` script (siehe weiter unten).

Border / Toolbar

Sofern sich der Scheduler nicht bereits in einer `Border` befindet, wird implizit eine `Border` um die Komponente erzeugt, damit gewisse Standardaktionen immer verfügbar sind. XML-Attribute können vom Scheduler über den Prefix `b-` an die implizite `Border` übergeben werden.

DetailView

Wie die `Table` kann auch der Scheduler eine DetailView haben, welche das jeweils selektierte Item anzeigt. Es ist jedoch nicht möglich, neue Items über die DetailView zu erzeugen.

Terminologie und Rollen

Diese Komponente verwendet bis zu drei verschiedene Arten von Objekten in unterschiedlichen Rollen:

Begriff	Beispiele	Verwendung
Item	ZeiterfassungsEintrag, Produktion, Termin	Dies sind die entlang einer Zeitachse anzuordnenden Objekte.
Contact	Mitarbeiter, AbstraktePerson, Benutzer, Maschine	Items werden nach Contacts gruppiert. Um den Scheduler benutzen zu können, muss in der Many-Relation, die als `property` gesetzt wird, eine Instanz vorhanden sein.
Group	Abteilung, Niederlassung, Mandant, Halle	Groups ermöglichen es, Contacts visuell zu gruppieren.

Attribute:

Name	Erlaubte Werte	Beschreibung
`b-*`	siehe Border	Prefix zum Durchleiten von XML-Attributen an die Border, welche den Scheduler implizit umgibt, sofern keine explizite Border vorhanden ist.
`background`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	**Optional.** Farbe, die im Scheduler auf die oberen beiden "Zeit-"Header (Jahr und Monat) angewandt wird, und das sonstige Farbschema bestimmt, wenn `headerColor` und `gridColor` nicht gesetzt sind.
`datatipFormat`	**date time duration** date time date time time duration duration none	Bestimmt, ob und in welchem Format datatips für Items angezeigt werden
`foreground`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	**Optional.** Zu verwendende Schriftfarbe im Scheduler.
`gridColor`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	**Optional.** Wenn nicht angegeben, wird eine aufgehellte `headerColor` verwendet.
`groupClass`	Name einer Entität oder Klasse (z.Bsp. `String`, `Integer`)	**Optional.** Name des Objekttyps, der die Rolle der "Groups" einnimmt.
`headerColor`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	**Optional.** Wenn nicht angegeben, wird ein aufgehellter `background` verwendet.
`itemClass`	Name einer Entität bzw. deren Klasse (z.Bsp. `ZeiteintragMitMitarbeiter`)	Name der Entität, die die Rolle der "Items" einnimmt. Neu erstellte Items sind per Default von diesem Typ.

Name	Erlaubte Werte	Beschreibung
`noNavigation`	true, **false**	Bewirkt, dass keine Buttons o.ä. Elemente auftauchen, mit denen man durch den Kalendar navigieren kann. Nützlich, wenn es im Kontext nur einen festen Zeitraum gibt.
`property`	Attributname einer Many-Relation	Many-Relation, welche die "Contacts" ausgibt. Damit der Scheduler effektiv bedienbar ist, muss in der Relation mindestens ein Objekt enthalten sein.
`range`	Week, **Month**	Steuert, welchen Zeitraum die Zeitachse darstellt, und in welchen Zeiteinheiten die Zeitachse vor und zurück geschaltet werden kann, bzw. Daten nachlädt.
`snapUnit`	**Day**, Hour, Minute	Steuert die Mindestgröße neu angelegter Items und um welche minimale Zeiteinheit ein Item bewegt / verändert werden darf. (Default: 1 Day)
`type`	**Resources**, Timetable, SingleMonth	Steuert den Kalendartyp bzw. -ansicht. Timetable: Eine Spalte pro Tag und Contact, eine Reihe pro 30 Minuten. Resources: Eine Spalte pro Tag oder Stunde, eine Reihe pro Contact. SingleMonth: Zeigt alle Tage eines Monats auf einmal, von Items aber nur den Titel. (*experimentell*)
`viewOnly`	**true**, false	Steuert, ob Items in der Ansicht angelegt und verändert werden dürfen. Da eine editierbare Ansicht eine etwas ausführlichere Konfiguration benötigt, ist der Default **true**.

Name	Erlaubte Werte	Beschreibung
workHours	**6-22**, [0-23]-[0-23]	Nur zur Verwendung mit type="Timetable". Die Arbeitszeiten werden farbig hervorgehoben und beim Öffnen des Schedulers wird automatisch zur Beginn der Arbeitszeit gescrollt.

Subelemente:

dataMapper

Das dataMapper Skript beschreibt, welche BOs geladen werden, welche Daten angezeigt, und wie sie verknüpft werden. Darüber hinaus können auch Verhalten und Darstellung beeinflusst werden.

Als Parameter wird ein mapper vom Typ FSchedulerBOMapper<ItemClass, ContactClass, GroupClass übergeben, wobei die "GroupClass" optional ist und Void sein kann, wenn keine Gruppen genutzt werden. "ContactClass" entspricht hierbei dem Typ der Relation, die im XML-Attribut "property" gesetzt wurde. "ItemClass" muss explizit via XML-Attribut "itemClass" angegeben worden sein angegeben werden.

Zusätzlich verfügbare Parameter sind: Transaction tx, ClientContextI ctx, FormContextI ftx und FScheduler fe.

Am mapper existieren die Methoden itemMapper, contactMapper und groupMapper, um die entsprechenden Rollen zu konfigurieren.
An diesen "Sub-Mappern" wiederum können wahlweise Attribute oder Funktionen hinterlegt werden.

Beispiel:

```
<dataMapper><![CDATA[
   mapper.itemMapper()
       .displayText('Lohnart.L10nName')
       .start('KommtGueltig') // !Mandatory! ①
       .end('GehtGueltig') // !Mandatory! ②
       .contacts('Mitarbeiter') // !Mandatory!
       .groups('Mitarbeiter.Abteilung')
       .lockedIf { it.getLohnart()?.istNormal() } ③
       .resizingAllowedIf { false } ④
       .background { Color.decode('#f2e3a6') } ⑤

   mapper.contactMapper()
       .format("(AbstraktePerson.Name2|left(1))(AbstraktePerson.Name1|left(1))") ⑥
       .groups('Abteilung') ⑦

   mapper.groupMapper()
       .name('L10nName')
]]></dataMapper>
```

① **itemMapper**: Damit Items bei Bedarf aus der Datenbank nachgeladen werden können, müssen `start` und `end` auf persistente Attribute verweisen.

② **itemMapper**: Um neue Items zu erzeugen und bestehende zu editieren, müssen die Attribute oder Relationen von `start`, `end` und `contacts` beschreibbar sein.

③ **itemMapper**: Über `lockedIf` kann das editieren einzelner Items blockiert werden. Es ist auch möglich, ein Boolean-Attribut zu übergeben.

④ **itemMapper**: Über `resizingAllowedIf` kann gesteuert werden, ob es möglich ist, die Dauer von Items zu bearbeiten oder ob die Dauer nach der Neuanlage fix bleibt. Per default ist resizing möglich.

⑤ **itemMapper**: Die Methoden `background` und `foreground` steuern die Farbgebung von Text und Hintergrund einzelner Items

⑥ **contactMapper**: Um zu bestimmen, wie ein Contact angezeigt werden soll, kann entweder über `format` ein CBOFormat oder über `name` ein Attribut oder eine Funktion angegeben werden.

⑦ **contactMapper**: Werden "Groups" verwendet, bestimmt `groups` welche Gruppen geladen werden. Neue Items müssen ihrem "Contact" und ggfs. dessen "Group" korrekt zugeordnet sein, oder sie sind **nicht sichtbar**.

Es ist möglich, an dem Mapper über `itemMapper().query()` eine Funktion zu hinterlegen, die alle "Items" im Zeitraum der Parameter `LocalDateTime start` und `LocalDateTime end` für die "Contacts" mit `Set<Long> contactID` lädt. Dies kann sinnvoll oder sogar notwendig sein, wenn mit nicht-persistenten Objekten gearbeitet wird, um trotzdem effizient Daten zu laden. Wird ein expliziter Query implementiert, und die Ansicht ist editierbar, so ist es notwendig, neu erzeugte und ungespeicherte BOs im Rahmen des Query-Scripts mit dem Query-Resultat zusammen zurückzugeben, damit diese beim "Umblättern" nicht verloren gehen.

Alle weiteren hier aufgelisteten Subelemente enthalten Groovy-Scripte und sind mit Ausnahme des

`dataMapper`-Skripts optional.

Name	Variablen / Rückgabetyp	Beschreibung	Default

`validIf`	`ItemClass item`
	`ContactClass contact`
	`LocalDateTime start`
	`LocalDateTime end`
	Rückgabetyp : `boolean`
		Wird aufgerufen, unmittelbar nachdem der Benutzer versucht hat, ein neues Item anzulegen oder ein Item zu modifizieren, aber bevor diese Neuanlage oder Modifikationen in einer Transaktion aufgezeichnet bzw. auf ein BO angewandt werden.
		Im Falle einer Neuanlage ist `item = null`.
		Gibt das Skript `false` zurück, wird die Modifikation abgebrochen, ohne einen Fehler zu werfen.
			Per Default unimplementiert. Es wird angenommen, dass jede Modifikation / Neuanlage zulässig ist.

Name	Variablen / Rückgabetyp	Beschreibung	Default
`newItem`	`FormContextI` ftx `Transaction` tx `ContactClass` contact `LocalDateTime` start `LocalDateTime` end Rückgabetyp : `ItemClass`	Wird aufgerufen, wenn der Benutzer die Neuanlage eines Items in der Ansicht via Mouse Drag abgeschlossen hat und `validIf` `true` war oder übersprungen wurde. Die Werte des von `newItem` erzeugten Objekts werden zurück in die Ansicht gesynct, d.h. die übergebenen Zeiten müssen nicht zwangsweise die finalen Zeiten sein. **CAUTION:** Es ist wichtig, dass das erzeugte Item mit dem übergebenen Contact (und dessen Group, wenn verwendet) verknüpft wird, da andernfalls das Item aus der Ansicht verschwindet.	Per Default wird versucht, ein neues Objekt von dem Typ, der in `itemClass` angegeben wurde, zu erzeugen. Start, Ende und Contact werden über die im Mapper definierten Attribute oder Setter-Funktionen gesetzt und müssen daher beschreibbar sein.
`onItemClick`	`ItemClass` item	Das Script wird bei Doppelklick auf ein Item ausgeführt.	Per Default öffnet sich das angeklickte Item in einem für den Benutzer verfügbaren und präferierten Formular.
`onContactClick`	`ContactClass` contact	Das Script wird bei Doppelklick auf einen Contact ausgeführt.	Per Default öffnet sich der angeklickte Contact in einem für den Benutzer verfügbaren und präferierten Formular.

Ausführliches Beispiel:

Das nachfolgende Beispiel zeigt eine Verwendung des Schedulers mit der nicht-persistenten Entität `ZEKumuliertFuerTag`. Da nicht-persistente Entitäten nicht in eine Transaction included und nicht neugeladen werden können, ist es notwendig, Skripte für `newItem` und `onItemClick` zu verwenden.

Das `validIf` Skript verhindert, dass Zeiteinträge an Feiertagen und Wochenende angelegt oder

dorthin bewegt werden.

```
<Scheduler property="Mitarbeiter" range="Week" viewOnly="false"
itemClass="ZEKumuliertFuerTag" groupClass="Abteilung" background="#587ac2"
foreground="#ffffff">
  <dataMapper><![CDATA[
    import java.awt.Color

    mapper.itemMapper()
        .displayText { i -> i.getBasierendAufZEs().values().findAll { !it.isDeleted()
}.collect { it.lohnart?.l10nName?:'Public Holiday' }.unique().join(', ') }
        .start('Start') // writeable vattr on ZEKumuliertFuerTag
        .end('Ende') // writeable vattr on ZEKumuliertFuerTag
        .contacts('Mitarbeiter')
        .groups('Mitarbeiter.Abteilung')
        .lockedIf { i -> i.getBasierendAufZEs().values().any {
it.istExplizitGestempelt() || it.lohnart?.istNormal() } }
        .resizingAllowedIf { false }
        .background { ZEKumuliertFuerTag ze ->
          def salaryType = ze.getLohnart()
          if (salaryType == null) {
            return null
          } else if (ze.getBasierendAufZEs().values().any { it.istExplizitGestempelt()
|| it.lohnart?.istNormal() }) {
            return Color.decode('#eeeeee') // grey
          } else if (salaryType.istUrlaub()) {
            return Color.decode('#f2e3a6') // yellow
          } else if (salaryType.istKrank()) {
            return Color.decode('#f2b9a6') // red
          } else if (salaryType.istHomeOffice()) {
            return Color.decode('#b7c1e1') // blue
          }
          return null
        }
        .query({ start, end, contactIDs ->
           // use the query option here, since we can't load non-persistent objects
from database, but we can load the persistent objects they are based on
           // thus, the most efficient solution is to load the underlying objects from
database and transform them
           def startDate = DateTimeToolsNG.toDateOfSystem(start)
           def endDate = DateTimeToolsNG.toDateOfSystem(end)
           final def query = 'ONLY ZeiterfassungsEintrag ze WHERE ze.Mitarbeiter.Id IN
LIST($1) AND ErsterZEFuerTag = NULL' +
               ' AND COALESCE(ze.Kommt, ze.KommtKorrigiert) <= $3 AND' +
               ' (COALESCE(ze.Geht, ze.GehtKorrigiert) >= $2 OR COALESCE(ze.Geht,
ze.GehtKorrigiert) = NULL)'

           def zes = tx.queryBO(query, [contactIDs, startDate, endDate] as Object[])
as Collection<ZeiterfassungsEintrag>
           def kumul = zes.collect { it.getZEKumuliertFuerTag() }
```

```
          // we need to re-add the new entries that were previously created in the tx
(if any)
          for (BO newBO : tx.getNewBOs()) {
            if (newBO instanceof ZeiterfassungsEintrag && !(newBO instanceof
MultiZeiterfassungsEintrag)) {
              if (((ZeiterfassungsEintrag) newBO).ersterZEFuerTag == null) {
                kumul.add(newBO.getZEKumuliertFuerTag())
              }
            }
          }
          return kumul
        })

    mapper.contactMapper()
        .format("(AbstraktePerson.Name1', ')(AbstraktePerson.Name2)")
        .groups('Abteilung')

    mapper.groupMapper()
        .name('L10nName')
  ]]></dataMapper>
  <newItem><![CDATA[
      import de.ipcon.db.core.BO
      import de.ipcon.tools.date.DateTimeTools
      import java.time.ZoneId
      import java.util.concurrent.TimeUnit

      def e = (Mitarbeiter) contact
      def d = Date.from(start.atZone(ZoneId.systemDefault()).toInstant())
      def salaryType = Lohnart.forHomeOffice(tx)

      createNewZEsForDay = { Mitarbeiter employee, Date day, Lohnart lohnart ->
        // [imagine some code creating persistent time entries for the given day]
      }

      def newItem = createNewZEsForDay(e, d, salaryType)
      return newItem.getZEKumuliertFuerTag()
  ]]></newItem>
  <onItemClick><![CDATA[
      import de.ipcon.form.MDIManagerI

      def ntx = ctx.getNewFormTransaction()
      // load the persistent entity which builds the np-entity so we have something to
frap
      def frappedEntry = ntx.getBO(item.getBasierendAufZEs().values().find().getId())
      // open the np-object now that it's been built with the right tx
      def cumul = frappedEntry.getZEKumuliertFuerTag()
      def form = ctx.getFormByTid('MCS_ZEITERFASSUNGSEINTRAG_S_TAG',
cumul.getClass().simpleName)
      // and finally open the form
      ctx.openForm(form, ntx, cumul, null, null, null, false,
```

```
MDIManagerI.VIEWTYPE_WIZARD, null, false, false, false, /*
doNotIncludeNotIncludedNewBOsWithTxAsLoader = */ true)
  ]]></onItemClick>
  <validIf><![CDATA[
      import java.time.DayOfWeek
      import java.time.ZoneId

      // This isValid check checks if the employee is allowed to work on the day we
try to set
      def employee = (Mitarbeiter) contact
      def d = Date.from(start.atZone(ZoneId.systemDefault()).toInstant())
      def contract = employee.getGueltigerArbeitsvertrag(d)

      def weekDay = start.getDayOfWeek()
      if (weekDay == DayOfWeek.SATURDAY) {
      return contract == null || contract.istSamstag()
      } else if (weekDay == DayOfWeek.SUNDAY) {
      return contract == null || contract.istSonntag()
      }

      // holidays
      def handler = contract?.getNiederlassung()?.getBundesland() ?:
contract?.getNiederlassung()?.getLand()
      if (handler != null && contract != null && !contract.istFeiertag()) {
      return !handler.isFeiertag(d)
      }

      return true
 ]]></validIf>
</Scheduler>
```

SimpleTimespanChooser

Wenig benutzte Komponente für Attribute vom Typ `Timespan` (= Dauer, z.Bsp. `1h 30m 42s`). Kombiniert ein Eingabefeld für eine Zahl mit einer Combobox für die Zeiteinheit (Minute, Stunde, Tag, etc.).

Leitet sich ab von TextInputComponent und erbt daher auch einen Teil der XML-Attribute.

Attribute:

Name	Erlaubte Werte	Beschreibung
`align`	String: `LEFT`, `CENTER`, `RIGHT`, `LEADING`, `TRAILING`	
`defaultUnit`	Erlaubte Werte: `years`, `months`, `weeks`, `days`, `hours`, `minutes`, `seconds`	Bestimmt die vorausgewählte Zeiteinheit, falls noch nichts eingegeben wurde.
`enabled`	true, **false**	Aktuell unbenutzt?
`fallBackProperty`		
`fontsize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	
`format`	???	Aktuell unbenutzt?
`rows`	int: **4**	
`selectAllWhenFocused`	Boolean: true, *false*	Wenn das Form-Element den Fokus bekommt wird der gesamte Inhalt selektiert

Tab

Attribute:

Name	Erlaubte Werte	Beschreibung
`background`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'
`debug`		
`editable`	Boolean: **true**, false	Bei false kann innerhalb dieses Elements kein Feld mehr editiert werden.
`foreground`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'

Name	Erlaubte Werte	Beschreibung
`grabFocus`	Boolean: **true**, false	
`implied`		
`lazy`	Boolean: **true**, false	Daten dieses Tabs erst beim Öffnen des Tabs laden, nicht schon beim Öffnen des Formulars.
`maximumSize`		
`maxSize`		
`minimumSize`		Alias. Siehe `minSize`
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	Gibt die minimale Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`missingPropertiesPolicy`		
`name`	String	Um das Form-Element innerhalb des Formulars referenzieren zu können
`preferredSize`		Alias. Siehe `prefSize`
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`scrollable`	Boolean: true, false; Oder Richtungsbeschränkung, z. B. scrollable="VERTICAL_ONLY"	Scrollbar für dieses Element ein- oder ausschalten bzw. auf eine Richtung beschränken.
`title`	String	Wird als Titel des Tabs angezeigt.
`toolTipText`	String.	Text, der angezeigt wird, wenn man den Mauszeiger über das Element hält.

Subelemente:

Name	Erlaubte Werte	Beschreibung
`autoSelectObject`		
`Column`		
`Columns`		
`DetailView`		
`onAfterSelectValue`		
`onShowingTab`		Wird ausgeführt, sobald der Tab aktiviert (angezeigt) wird
`onHidingTab`		Wird ausgeführt, sobald der Tab deaktiviert (versteckt) wird
`Query`		
`View`		

TabbedView

Attribute:

Name	Erlaubte Werte	Beschreibung
`antiAlias`	Boolean: true, **false**	
`conflictPolicy`	String: '**IGNORE**', 'ASK'	Gibt an, wie Konflikte beim Speichern durch veraltete BOs behandelt werden sollen. Nur in der äussersten TabbedView / View erlaubt. IGNORE: Ignoriere Konflikte und übernehme/merge die Änderungen, ASK: Prüfe auf Konflikte und Frage den Benutzer wie damit umgegangen werden soll.
`debug`		
`editable`	Boolean: **true**, false	Bei false kann innerhalb dieses Elements kein Feld mehr editiert werden.
`height`	Integer mit Einheit px, c, em oder dlu	Initiale Höhe der View in Pixeln oder der angegebenen Einheit.
`ignoreOtherLocalTransactionSaves`	Boolean: true, **false**	Nur für Formular-Roots. Änderungen aus lokalen Speichervorgängen im Client in diesem Formular nicht nachziehen
`implied`		
`l10nBundle`		
`maximumSize`		
`maxSize`	Tupel: (horizontal, vertikal). Beispiele: maxSize="4c, 5c"; maxSize="4c,"; maxSize=",5c"	
`minimumSize`		Alias. Siehe `minSize`

Name	Erlaubte Werte	Beschreibung
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	Gibt die minimale Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`missingPropertiesPolicy`		
`name`	String	Um das Form-Element innerhalb des Formulars referenzieren zu können
`preferredSize`		Alias. Siehe `prefSize`
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`rotateLabels`	Boolean: **true**, false	Gibt an, ob bei tabPlacement `LEFT` oder `RIGHT` die Beschriftung der Tabs gedreht werden soll oder nicht.
`tabLayoutPolicy`		
`tabPlacement`	String: `BOTTOM`, `TOP`, `LEFT`, `RIGHT`	Die Reiter der TabbedView unten, oben, links oder rechts anzeigen.
`useMaximumHeight`	Boolean: true, **false**, z. B.: useMaximumHeight="true"	
`useMaximumWidth`	Boolean: true, **false**, z. B.: useMaximumWidth="true"	
`width`	Integer mit Einheit px, c, em oder dlu	Initiale Breite der View in Pixeln oder der angegebenen Einheit.

Subelemente:

Name	Erlaubte Werte	Beschreibung
`Tab`		

Table

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`alternateCellBackground`	Farbe, z.Bsp. `#ffffff` oder `255 255 255`	Setzt die Default-Hintergrundfarbe aller geraden Zeilen
`asyncModel`	Boolean: **true**, false	
`autoRefresh`	Boolean: true, **false**	Änderungen an Objekten des Typs, welche in der Tabelle angezeigt werden, führen zu einer Aktualisierung der Tabelle
`additionalAutoRefreshEntities`	String	komma-separierte Liste von Entitätsnamen; Änderungen an Objekten dieser Typen führen zusätzlich zum in der Tabelle angezeigten Typ zu einer Aktualisierung der Tabelle; funktioniert nur zusammen mit `autoRefresh="true"`; Unter-Entitäten werden automatisch mitüberwacht
`autoSelectFirst`	Boolean: **true**, false	
`autoSelectLast`	Boolean: true, **false**	
`autoSelectNone`	Boolean: true, **false**	
`cellBackground`	Farbe, z.Bsp. `#ffffff` oder `255 255 255`	Setzt die Default-Hintergrundfarbe aller ungeraden Zeilen

Name	Erlaubte Werte	Beschreibung
`columns`	String	Definition der Spalten einer Tabelle im Kurzformat. Bsp.: `columns="Attribut1, 25c, desc3 | Attribut2, desc | Attribut3, desc2"` Einzelne Spaltendefintionen werden durch `|` getrennt. `desc` und `desc2` bestimmen eine absteigende Sortierung. Die Numerierung legt die Priorität der Spalte beim Sortieren fest, wobei `desc` vor `desc2` berücksichtigt wird und `desc2` vor höheren Ziffern. Durch Verwendung von `asc` kann auch aufsteigend sortiert werden. Spalten werden ausgeblendet, indem sie einfach nicht deklariert werden. Die Reihenfolge kann geändert werden. `25c` definiert hier eine Spaltenbreite von 25 Buchstaben. Weitere erlaubte Einheiten sind `px`, `em` und `dlu`. Bei polymorphem Tabelleninhalt kann es hilfreich sein, den Typ des jeweiligen Objektes mit `Bot.Name '$R{BOTyp}'` anzuzeigen.
`columnSelectionAllowed`	Boolean: **true**, false	False: Bei Mausklick (oder `.selectObject()`) wird die komplette Zeile ausgewählt
`createInDetailView`	Boolean: true, **false**	
`dependent`	Boolean: true, false	Die anzuzeigenden Elemente sind nicht eigenständig und leben nur in diesem View.
`displayClass` DEPRECATED		

Name	Erlaubte Werte	Beschreibung
`displayProperty` DEPRECATED		siehe `property`
`easyEdit`	Boolean: **true**, false	Selektion per Doppelklick
`editableDetailView`	Boolean: true, **false**	Bewirkt, dass die DetailView der Tabelle das Bearbeiten der dort angezeigten Werte zulässt. Nützlich, wenn es sich bei diesen Werten bspw. um Attribute aus Subrelationen eines virtuellen Attributs handelt, die eigentlich gesperrt wären.
`editable`	Boolean: true, **false**	
`entity`	String: Entitätsname	Bestimmt die Entität, deren Objekte diese Tabelle darstellen soll.
`explicitStart`	Boolean: true, **false**	Die Daten im Lesezeichen werden erst vom Server geladen, wenn man das explizit mit F5 oder RETURN "anfordert". Man kann also erstmal alle nötigen Filter setzen und das Lesezeichen beginnt nicht bereits nach dem ersten gesetzten Filter mit dem Laden. Nützlich bei Entitäten mit sehr vielen Objekten, bei denen es angeraten ist erst mal ein bisschen vorzufiltern.
`filter`	String	
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`freeSearch`	Boolean: **true**, false	
`height`	Integer	

Name	Erlaubte Werte	Beschreibung
`horizontalScrollBarPolicy`	String: `ALWAYS`, `AS_NEEDED`, `NEVER`	`ALWAYS`: Zeigt eine horizontale Scrollbar an, sobald ein Objekt in der Tabelle nicht mehr vollständig in das aktuelle Fenster passt. `AS_NEEDED`: Zeigt eine horizontale Scrollbar an, sobald ein Objekt in der Tabelle nicht mehr vollständig in das aktuelle Fenster passt. `NEVER`: Es wird niemals eine Scrollbar angezeigt, auch nicht dann, wenn nicht alle Objekte der Tabelle in das aktuelle Fenster passen.
`ignoreInitialFocus`	Boolean: true, **false**	
`implied`		
`intercellSpacingX`	Integer	Zusätzlicher freier Raum in Zellen horizontal.
`intercellSpacingY`	Integer	Zusätzlicher freier Raum in Zellen vertikal.
`itemProperty`	String	Zeiger auf die Property, die in den Elementen die Postennummer darstellt. Posten-Modus. Fuehrt dazu, dass zwei Aktionen, naemlich Posten-rauf und Posten-runter gebaut werden, und die ITable standardmaessig die Postenspalte aufsteigend sortiert.
`linkOnly`	Boolean: true, **false**	Es können nur bestehende Objekte verknüpft werden und keine neue erstellt
`loadImmediate`	Boolean: true, **false**	Bestimmt, ob die Objekte in der Tabelle sofort geladen werden sollen. Wenn `true`, ist ein Drücken von Enter im Suchbalken zur Anzeige der Inhalte nicht erforderlich.
`maxRowHeight`	Integer	

Name	Erlaubte Werte	Beschreibung
`maxRows`	Integer: **100000**	Definiert die maximale Anzeige von Objekten (vergleichbar mit dem von SQL bekannten "LIMIT 100000")
`maximumSize`	siehe `maxSize`	
`maxSize`	Tupel: (horizontal, vertikal). Beispiele: maxSize="4c, 5c"; maxSize="4c,"; maxSize=",5c"	
`minimumSize`	siehe `minSize`	
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	
`missingPropertiesPolicy`	String: **error**, ignore, log	Wie soll mit nicht existierenden oder falsch geschriebene Attributen in der Spaltendefinition umgegangen werden
`name`	String	Um das Form-Element innerhalb des Formulars referenzieren zu können
`openFormTid`	String	Tid des Formulars, das benutzt werden soll, um Objekte aus dieser Tabelle zu öffnen.
`openProperty`	String	Name der Property für welche beim Doppelklick das Standardformular geöffnet wird.
`parentClass` DEPRECATED		
`parentEntity`	String	
`preferredSize`		Alias. Siehe `prefSize`
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`preferredVisibleRows`	Integer: **5**	Definiert die bevorzugte Anzahl angezeigter Zeilen

Name	Erlaubte Werte	Beschreibung
`property`	String	
`readOnly`	Boolean: true, **false**	
`reloadBOsWhenOpening`	Boolean: true, **false**	True: Beim Öffnen von BOs über dieses Lesezeichen werden die BOs immer neu vom Server neu geladen, anstatt die Instanz aus dem Lesezeichen zu verwenden.
`resizeMode`	String: `ALL_COLUMNS`, `LAST_COLUMN`, `NEXT_COLUMN`, `SUBSEQUENT_COLUMNS`, `OFF`	Bestimmt das Verhalten der Spaltenbreite beim Vergrößern/Verkleinern der Tabelle.
`rowHeight`		Steuert die Default-Höhe der Zeilen, z.Bsp `2c`
`rowSelectionAllowed`	Boolean: **true**, false	False: Bei Mausklick wird die komplette Spalte ausgewählt
`showEntityName`	Boolean: **true**, false	
`showHorizontalLines`	Boolean: **true**, false	
`showVerticalLines`	Boolean: **true**, false	
`singleClickEdit`	Boolean: true, **false**	
`sortForSelected`	Boolean: true, **false**	
`subentitiesToExclude`	String: Kommaseparierte Liste von Entitätsnamen	Subentitäten der in der Tabelle dargestellten Entität, die nicht angezeigt werden sollen.
`templateSource`	String	
`useMaximumHeight`	Boolean: true, **false**, z. B.: useMaximumHeight="true"	
`useMaximumWidth`	Boolean: true, **false**, z. B.: useMaximumWidth="true"	
`usePolymorphySelectionTree`	Boolean: true, **false**	

Name	Erlaubte Werte	Beschreibung
`verticalScrollBarPolicy`	String: `ALWAYS`, `AS_NEEDED`, `NEVER`	`ALWAYS`: Zeigt eine vertikale Scrollbar an, sobald ein Objekt in der Tabelle nicht mehr vollständig in das aktuelle Fenster passt. `AS_NEEDED`: Zeigt eine vertikale Scrollbar an, sobald ein Objekt in der Tabelle nicht mehr vollständig in das aktuelle Fenster passt. `NEVER`: Es wird niemals eine Scrollbar angezeigt, auch nicht dann, wenn nicht alle Objekte der Tabelle in das aktuelle Fenster passen.
`viewOnly`	Boolean: true, **false**	Es darf nur geschaut werden, verlinken bestehender Objekte, etc ist nicht möglich
`width`	Integer	

Column

Attribute:

Name	Erlaubte Werte	Beschreibung
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`displayProperty` DEPRECATED		siehe `property`
`format`	String (CBOFormat).	Legt fest, wie das BO aus der property für die Anzeige in der Zelle formatiert wird. Bsp: "Id': 'Name"; funktioniert nur, wenn das letzte Attribut in der property auf eine Relation verweist
`displayFormat` DEPRECATED		siehe `format`

Name	Erlaubte Werte	Beschreibung
`toolTipFormat`	String (CBOFormat).	Legt fest, wie das BO der aktuellen Zeile als ToolTip für diese Spalte formatiert wird. Bsp: "Id': 'Name"
`title`	String	Überschrift für die Spalte.
`width`	Integer mit Einheit px, c, em oder dlu	Initiale Breite der Spalte in Pixeln oder der angegebenen Einheit.
`vAlign`	String: TOP, CENTER, BOTTOM, z. B. vAlign="TOP"	Bestimmt die vertikale Ausrichtung des Textes innerhalb des Elements. Die Höhe des Elements muss größer sein als eine normale Zeilenhöhe, was z. B. durch Setzen des Attributs prefSize erreicht werden kann.
`cellBackground`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	Bestimmt die Hintergrundfarbe der Spalte für ungerade Zeilennummern.

Name	Erlaubte Werte	Beschreibung
`alternateCellBackground`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	Bestimmt die alternative Hintergrundfarbe der Spalte für gerade Zeilennummern.
`cellForeground`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	Bestimmt die Schriftfarbe der Spalte für ungerade Zeilennummern.
`alternateCellForeground`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	Bestimmt die Schriftfarbe der Spalte für gerade Zeilennummern.
`deferredRendering`	Boolean: true, **false**	**FIXME** aktuell ohne Funktion?
`debug`	Boolean: true, **false**	Aktiviert Info-Level Logging beim Rendern von Zellen.
`caching`	Boolean: **true**, false	Aktiviert oder deaktiviert das Caching von angezeigten Werten.
`style`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	
`justification`	String: **`LEFT`**, `CENTER`, `RIGHT`	
`sort`	`ASC`, `DESC`, `NONE`	Gibt an, in welcher Richtung die Spalte sortiert werden soll. NONE verbietet das Sortieren.

Name	Erlaubte Werte	Beschreibung
`sortLevel`	int: **0**	Gibt die Sortier-Reihenfolge der Spalten an.
`rendererClass`	String	Hier kann der Name einer TableCellRenderer-Klasse angegeben werden, die statt des Default Renderers benutzt werden soll.

headerRenderer und renderer

Hier können spezielle Renderer konfiguriert werden um z.B. die Farbe abhängig vom Inhalt der Zelle zu ändern. Sie müssen auf Spaltenebene definiert werden:

```
<Column property="Enabled">
   <renderer>
      import java.awt.Color
      renderer.setHorizontalAlignment(javax.swing.SwingConstants.CENTER)
      if (!value) { ①
         renderer.setBackground((row&amp;amp;1) == 0 ? new Color(234, 176, 176) : new Color(240, 200, 200)) ②
         renderer.setText('\u2717') ③
      } else {
         renderer.setBackground((row&amp;amp;1) == 0 ? new Color(199, 234, 176) : new Color(207, 226, 186))
         renderer.setText('\u2714')
      }
   </renderer>
</Column>
```

① value beinhaltet den Wert aus dem Property. In diesem Fall also `Enabled` als Boolean.

② Falls der Wert von `Enabled` false/null ist wird der Hintergrund abhängig von der Zeilen-Nummer in einem unterschiedlichen Rot-Ton eingefärbt.

③ Der Inhalt wird auf das Unicode-Zeichen gesetzt.

Wenn man am Background rumfuhrwerkt, muss man natürlich auch den "isSelected"-Status abfragen und den Background selbst entsprechend setzen (z.B. das gesetzte Rot bei markierten Zellen etwas bläulich einfärben).

DetailView

Die DetailView wird als Subelement der Table verwendet, um Details zum jeweils gerade innerhalb der Tabelle angeklickten Objekt anzuzeigen bzw. um das Bearbeiten der Attribute des Objekts zu ermöglichen. Sind von den Änderungen innerhalb der DetailView Inhalte der Tabelle betroffen, werden diese automatisch aktualisiert.

Achtung: Falls die DetailView für eine Tabelle in einem Lesezeichen verwendet wird, muss im `<table>` Element das Attribut `viewOnly="true"` gesetzt sein.

Attribute: FIXME

Name	Erlaubte Werte	Beschreibung
`adjustableSplit`	Boolean: true, **false**	Falls Wert auf **true**, kann der Bereich der DetailView mit der Maus verschoben, sprich vergrößert/verkleinert werden.
`position`	String: `NORTH`, **`SOUTH`**, `WEST`, `EAST`	
`resizeWeight`	Float: **0.0**	
`scrollable`	Boolean: true, false; Oder Richtungsbeschränkung, z. B. scrollable="VERTICAL_ONLY"	Scrollbar für dieses Element ein- oder ausschalten bzw. auf eine Richtung beschränken.

Sollen Attribute von Objekten aus Relationen des Tabellenobjekts geändert werden, müssen für diese jedoch `onAfterSetValue`-Hooks implementiert werden, der die Version des eigentlichen Objekts anstößt, um ein Aktualisieren der Tabelle zu erzwingen.

Beispiel mit `onAfterSetValue`:

```
<Table property="Buecher" columns="Titel | Erscheinungsjahr | Autor.Familienname">
  <DetailView name="autor">
    <Border etched="true" title="Details zum Autor">
      <View scrollable="true">
        <Element label="Familienname">
          <Text property="Autor.Familienname">
            <onAfterSetValue
language="groovy">ftx['autor'].getBO().bumpVersion()</onAfterSetValue>
          </Text>
        </Element>
      </View>
    </Border>
  </DetailView>
</Table>
```

MultipleChoiceFilterGUI

Hierbei handelt es sich um einen Filter, der via `<filter type="multipleChoice">` in eine FTable (Lesezeichen, Popup) eingebaut werden kann.
Für Hilfe beim Anlegen eines Filters die Solstice User-Dokumentation lesen.

Name	Erlaubte Werte	Beschreibung
preselectIdx	int	Gibt an, welcher Index in dem Dropdown standardmäßig ausgewählt ist.
sort	`ASC`, `DESC`, `NONE`	Gibt an, in welcher Reihenfolge die Queryergebnisse sortiert werden sollen

Text

Attributes:

Name	Erlaubte Werte	Beschreibung
`align`	String: `LEFT`, `CENTER`, `RIGHT`, `LEADING`, `TRAILING`	
`class`		Angabe der Klasse zur Textdarstellung, z.B. `ITextArea`
`disabled`	Boolean: true, **false**	Das Feld wird ausgegraut und kann nicht editiert werden.
`displayFormat` DEPRECATED		siehe `format`
`displayProperty` DEPRECATED		siehe `property`
`font`		
`foreground`	Farbangabe. Bitte entweder als "#rrggbbaa" oder "r,g,b,a", "r g b a" oder eine Farbkonstante der java.awt.Color, z.B. YELLOW angeben. Farbnamen mit Postfix "ISH" werden in Richtung weiss verschoben (Mittelwert der einzelnen Farbwerte und 255). Der Alphawert ist optional. Die einzelnen Werte sind bei den beiden letzteren Varianten entweder Float-Werte von 0.0..1.0 (bei 1 bitte 1.0 angeben!) oder Integer-Werte von 0..255. bitte nur die eine Sorte Werte verwenden. Oder fuer Random-Farbe: random.'	
`format`		
`lineWrap`	Boolean: **true**, false	Bricht Text an der rechten Kante um, statt eine Scrollbar anzuzeigen.
`password`	Boolean: true, **false**	Passwortfeld statt normalem Text.
`roundingFormat`		
`rows`	int: **4**	

Name	Erlaubte Werte	Beschreibung
`selectAllWhenFocused`	Boolean: true, *false*	Wenn das Form-Element den Fokus bekommt wird der gesamte Inhalt selektiert
`syncOnWait`		
`syncOnWaitDelay`		
`tabSize`	int: **4**	
`translationAvailable`		
`wrapStyleWord`	Boolean: **true**, false	Wenn Text umgebrochen wird, dann möglichst an Whitespace-Grenzen.

TimeSelector

Eingabe-/Bearbeitungsmöglichkeit für ein oder mehrere Zeitspannen innerhalb eines Zeitraums.

FIXME TT 2019-05-08: Wird praktisch nie benutzt.

Name	Erlaubte Werte	Beschreibung
`startingHour`	int: **0**-23	Erste mögliche auszuwählende Stunde des Tages
`rangeHours`	int: 1-n (**24**)	Anzahl der von `startingHour` ausgehend anzuzeigenden Stunden
`interval`	int: 5,10,**15**,20,30,60	Länge der Intervall-Aufteilung innerhalb einer Stunde
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`onRelease`	Element FIXME ???	FIXME ???

ToggleButton

Knopf der zwischen zwei Zuständen wechselt; Eingabe-/Bearbeitungsmöglichkeit für Boolean-Werte.

Name	Erlaubte Werte	Beschreibung
`action`	String	`cmd`-Attribut der Action, die bei Klick auf den Button ausgeführt werden soll.
`displayProperty` DEPRECATED		siehe `property`
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`class`		Angabe eigener Klasse für das GUI Control Element. Diese muss vom Typ `javax.swing.JToggleButton` sein.
`background`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	Hintergrundfarbe des Buttons im "nicht-gedrückten" Zustand.
`foreground`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	Vordergrundfarbe (d.h. Schriftfarbe).
`selectedColor`	Farbangabe. #rrggbbaa, r,g,b,a, r g b a. Alpha optional. Beispiele: #14f900, 255,255,0,0, 0.5 0.4 0.3 0.2, GREEN, YELLOWISH	Hintergrundfarbe des Buttons im gedrückten Zustand. Falls `selectedColor` nicht angegeben ist, wird `background` in einer anderen Schattierung als `selectedColor` verwendet.
`fontSize`	String: +X%	Gibt an, um wieviel Prozent die Schrift vergrößert werden soll.
`fontStyle`	String: z. B. fontStyle="bold", fontStyle="italics" oder fontStyle="BOLD", fontStyle="italics"	

Name	Erlaubte Werte	Beschreibung
`hAlign`		
`vAlign`	String: TOP, CENTER, BOTTOM, z. B. vAlign="TOP"	Bestimmt die vertikale Ausrichtung des Textes innerhalb des Elements. Die Höhe des Elements muss größer sein als eine normale Zeilenhöhe, was z. B. durch Setzen des Attributs prefSize erreicht werden kann.
`multiClickThreshold`	Integer (750ms)	Spezifiziert innerhalb welches Zeitraums mehrfaches Klicken des Buttons als ein einfaches Klicken interpretiert wird
`trueIcon`	String: z.Bsp. icon="20x20/New.gif", icon="image/remove_red_eye.svg" oder icon="image/remove_red_eye.svg@5085dc" (mit Farbangabe in hex; nur für SVGs verfügbar)	Pfad zum gewünschten icon. Icon für den Toggle-Button im "gedrückten" Zustand.
`falseIcon`	String: z.Bsp. icon="20x20/New.gif", icon="image/remove_red_eye.svg" oder icon="image/remove_red_eye.svg@5085dc" (mit Farbangabe in hex; nur für SVGs verfügbar)	Pfad zum gewünschten icon. Icon für den Toggle-Button im "nicht-gedrückten" Zustand.
`text`	String	Beschriftung des Toggle-Buttons. Falls `falseText` angegeben ist, wird dieser Text nur im "gedrückten" Zustand angezeigt.
`trueText`	String	Alias für `text`.
`falseText`	String	Text für den Toggle-Button im "nicht-gedrückten" Zustand.

Tree

Auswahl von Einträgen für eine Relation, deren mögliche Einträge hierarchisch in einer Baumstruktur angeordnet sind.

Attributes:

Name	Erlaubte Werte	Beschreibung
`childrenProperty`	String. Bsp.: `childrenProperty="Kinder"`	Kinder-Attribut. Bei Subentitäten von Querbaum einfach "Kinder."
`displayClass` DEPRECATED		
`displayProperty` DEPRECATED		siehe `property`
`entity`	String: Entitätsname. Bsp.: `entity="Verzeichnis"`	Die Entität, deren Instanzen als Baum dargestellt werden sollen. Obligatorisch.
`filter`		ACHTUNG: Dieses Attribut wird im Moment noch nicht vom Tree unterstützt.
`format`	String: Attributname, z. B. `format="Bezeichnung"`	Das Attribut, das die Beschriftung der Knoten im Baum stellen soll. Default: `ui description`
`height`	Integer. Bsp.: `height="3"`	
`parentProperty`	String. Bsp.: `parentProperty="Elter"`	Eltern Attribut. Bei Subentitäten von Querbaum einfach "Elter."
`property`	String: Property accessor. Beispiele: property="Buch.Autor.Alter"; property="."	Das Attribut der aktuell betrachteten Entität, das im Kontext dieses Elementes verwendet werden soll. Im zweiten Beispiel wird das BO des aktuellen Formkontexts als Property gesetzt.
`restrictToEntity`		
`usePolymorphySelectionTree`	Boolean: true, **false**	
`width`	Integer. Bsp.: `width="15"`	

Uri

Attributes:

Name	Erlaubte Werte	Beschreibung
`autoaddProtocol`	Boolean: **true**, false	

View

Attributes:

Name	Erlaubte Werte	Beschreibung
`autoHideElements`	Boolean: **true**, false	
`border`		
`columns`	Integer, z. B.: `columns="3"`	Teilt die Inhalte der View auf die angegebene Anzahl Spalten auf.
`conflictPolicy`	String: '**IGNORE**', 'ASK'	Gibt an, wie Konflikte beim Speichern durch veraltete BOs behandelt werden sollen. Nur in der äussersten TabbedView / View erlaubt. IGNORE: Ignoriere Konflikte und übernehme/merge die Änderungen, ASK: Prüfe auf Konflikte und Frage den Benutzer wie damit umgegangen werden soll.
`debug`		
`defaultRightFill`	Double: **1**. Wert zwischen `0` und `1`. `defaultRightFill="1"`: maximale Breite; 0-1: Prozentuale Streckung, bevor die Streckung aufgegeben wird. Fallback ist `0`. Bsp.: `defaultRightFill="0.5"`	Verhindert Streckung von Feldern und benutzt `preferredSize`.
`delegateToParent`	Boolean: true, **false**	Die View übernimmt die Anzeigeeinstellungen (z.B. Anzahl columns) der übergeordneten View. Dadurch kann z.B. einheiltiche Anzeige der Formularelemente in verscheidenen Views erreicht werden.
`editable`	Boolean: **true**, false	Wenn false werden Kindelemente gesperrt, sodass diese nicht editiert werden können.
`externalHGap`	Integer mit Einheit px, c, em oder dlu. Bsp.: `externalHGap="3c"`	Erzeugt links und rechts der View den angegebenen Abstand.

Name	Erlaubte Werte	Beschreibung
`externalVGap`	Integer mit Einheit px, c, em oder dlu. Bsp.: `externalVGap="3c"`	Erzeugt über und unter der View den angegebenen Abstand.
`height`		
`hideElementsForNullBO`	Boolean: **true**, false	
`ignoreOtherLocalTransactionSaves`	Boolean: true, **false**	Nur für Formular-Roots. Änderungen aus lokalen Speichervorgängen im Client in diesem Formular nicht nachziehen
`implied`		
`internalHGap`	Integer mit Einheit px, c, em oder dlu. Bsp.: `internalHGap="3c"`	Fügt den angegebenen Abstand zwischen Labels und Feldern ein.
`internalVGap`	Integer mit Einheit px, c, em oder dlu. Bsp.: `internalVGap="3c"`	Fügt den angegebenen Abstand zwischen Zeilen ein.
`l10nBundle`		
`maximumSize`		Alias. Siehe `maxSize`
`maxSize`	Tupel: (horizontal, vertikal). Beispiele: maxSize="4c, 5c"; maxSize="4c,"; maxSize=",5c"	
`minimumSize`		Alias. Siehe `minSize`
`minSize`	Tupel: (horizontal, vertikal). Beispiele: minSize="4c, 5c"; minSize="4c,"; minSize=",5c"	Gibt die minimale Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`missingPropertiesPolicy`		
`name`	String	Um das Form-Element innerhalb des Formulars referenzieren zu können
`preferredSize`		Alias. Siehe `prefSize`

Name	Erlaubte Werte	Beschreibung
`prefSize`	Tupel: (horizontal, vertikal). Beispiele: prefSize="8c, 6c"; prefSize="8c,"; prefSize=",6c"	Gibt die bevorzugte Größe des Elements mit horizontalem und vertikalem Wert an und ersetzt die ansonsten automatische Größenberechnung. Diese würde das Element auf das Minimum an Platz für dessen Inhalt setzen. Beide Werte sind jeweils optional.
`scrollable`	Boolean: true, false; Oder Richtungsbeschränkung, z. B. scrollable="VERTICAL_ONLY"	Scrollbar für dieses Element ein- oder ausschalten bzw. auf eine Richtung beschränken.
`tint`	<Farbwert>[@<Intensität>], Intensität ist default 0.1; Beispiele: tint="#ff0000 @ 0.1"; tint="GREENISH"	Färbt den Hintergrund des Views in der angegebenen Farbe ein
`useMaximumHeight`	Boolean: true, **false**, z. B.: useMaximumHeight="true"	
`useMaximumWidth`	Boolean: true, **false**, z. B.: useMaximumWidth="true"	
`width`		

Datenaustausch

MyTISM bietet diverse Möglichkeiten des Imports als auch Export von Daten.

Import

FIXME

Export

Excel

Für den Datenaustausch nach Excel bzw. kompatiblen Tabellenkalkulationsprogrammen kommt intern die Apache POI Bibliothek zum Einsatz (https://poi.apache.org).

Etwas schöner benutzen lässt sich das mit dem XlsHandler, der in dem ein oder anderen Projekt bereits zur Anwendung gekommen ist (einfach über das Repository nach XlsHandler.nrx suchen).
Dort gibt es drei Methoden: . Bauen von Header . Befüllen des Sheets . Den eigentlichen Export erstellen

Die Features, die man nutzen kann, findet man auf der Seite von Apache POI beschrieben. Grob gesagt geht "fast" alles:

- mehrere Sheets
- Styles
- Formeln
- Bilder
- ...

(Siehe die Tabelle unten hier: https://poi.apache.org/components/spreadsheet/index.html)

Bekannte Einschränkungen sind:

- wenige Charts
- keine Makros
- kein Pivot
- gewisse Größenbeschränkungen der Files

(Siehe auch https://poi.apache.org/components/spreadsheet/limitations.html)

Für das, was normalerweise benötigt wird, sollte es aber hoffentlich ausreichend sein.

Ein Beispiel für den Export findet man im Repository, wenn man nach der Klasse AngebotsgruppeGas.nrx sucht (Methode "exportSummenlastgaenge").

Natürlich kann man das auch in einem Groovy-Skript direkt im Client / einem Lesezeichen / einem Formular machen.

Zu überlegen ist, ob bei entsprechend häufiger Verwendung (u.a. auch direkt durch den Kunden), wir die interne API möglichst benutzerfreundlich gestalten, damit die Skripte nicht zu kompliziert werden.

Lokale autoritative sowie synchronisierende Instanzen zum Entwickeln aufsetzen

Manchmal benötigt man beim Entwickeln bestimmter Features - meist bei Core-Arbeiten - eine Konstellation von autoritativem Server und synchronisierendem/n Servern.

Hier eine stichwortartige Anleitung wie man solche eine Konstellation auf seinem lokalen Entwickersystem aufsetzen kann. Ausgegangen wird davon, dass eine Instanz - die autoritative Instanz - bereits existiert. Außerdem wird davon ausgegangen, dass beide Instanzen immer denselben Codestand nutzen sollen - falls nicht muss für die synchronisierende Instanz ein eigenes "deploy"-Verzeichnis angelegt werden, statt das der autoritativen Instanz zu verlinken.

Im folgenden wurde überall der Zusatz "syncnode" für Namen von zur synchronisierende Instanz gehörigen Dingen gewählt. Dieser ist aber prinzipiell beliebig bzw. das Verzeichnis könnte generell theoretisch beliebig benannt werden. Der besseren Zuordnung halber sollte man allerdings die hier beschriebene Variante wählen.

Folgendes ist dann zum Aufsetzen der zugehörigen synchronisierende Instanz zu tun:

- Verzeichnis `/.<projektkuerzel>-syncnode` anlegen (muss normalerweise als root/via sudo gemacht werden, um das /-Verzeichnis modifizieren zu dürfen)
- Besitzer korrekt setzen (wie vom Verzeichnis der autoritativen Instanz `/.<projektkuerzel>`)
- `/.<projektkuerzel>/deploy` nach `/.<projektkuerzel>-syncnode/deploy` verlinken
- `/.<projektkuerzel>/filesRoot` nach `/.<projektkuerzel>-syncnode/filesRoot` verlinken
- `/.<projektkuerzel>/lib/mytism` nach `/.<projektkuerzel>-syncnode/lib/mytism` verlinken
- Dateien `/.<projektkuerzel>mytism.ini` etc. nach `/.<projektkuerzel>-syncnode` kopieren
- `/.<projektkuerzel>-syncnode/mytism.ini` anpassen:
  - `nodeNumber` anpassen, falls Id der zugehörigen BN bekannt; oder Zeile komplett löschen, dann wird automatisch eine neue angelegt
  - (DB-)`url` anpassen: `<dbname>` → `<dbname>-syncnode`
  - `filesRoot` anpassen → `/.<projektkuerzel>-syncnode/filesRoot`
  - `authoritative=1` → `authoritative=0`
  - SyncService*-Einträge aktivieren:

```
[Sync]
url=socket://localhost:4242?compress=zlib9
user=Admin
pass=
```

Statt "Admin" und leerem Passwort bitte die korrekten Zugangsdaten aus der Datenbank eintragen.

  - `port` und `tlsPort` anpassen → z.B. statt 4242 und 4243 Ports 14242 und 14243 nutzen
  - `DeploySite`-Einstellungen anpassen oder neu eintragen, falls noch nicht vorhanden:

```
[DeploySite]
host=localhost
port=18080
--tlsHost=
--tlsPort=
requireAuthentication=0
--reauthEveryXDays=90
```

- Skript `mytism` anpassen:
  - `export PRJDIR="/.<projektkuerzel>-syncnode"`
  - `export DBNAME="<dbname>-syncnode"`

Danach kann, wie in der Admin-Doku unter "Aufsetzen eines syncenden MyTISM-Servers aus einem Backup des autoritativen Servers" beschrieben, ein Datenbank-Backup der autoritativen Instanz eingespielt und die beiden Instanzen genutzt werden.