MyTISM - Ein Datenbank- und Anwendungs-Framework

Inhaltsverzeichnis

Einleitung . . . 1
MyTISM: Ein starkes Fundament für Ihre Anwendung . . . 2
Vorstellung von MyTISM . . . 3
Was bedeutet der Name "MyTISM"? . . . 4
Warum MyTISM? . . . 5
Was ist MyTISM genau? . . . 6
Was bringt die Zukunft? . . . 7
SOLSTICE - der Client . . . 8
Grundlagen . . . 9
Ansicht der Benutzeroberfläche . . . 9
Bereiche des Hauptfensters . . . 9
Mehrfachfenstermodus . . . 10
Navigationsbaum . . . 10
Aussehen und Position von Elementen . . . 10
Sichtbarkeit von Elementen . . . 10
Strukturelemente . . . 10
Arbeiten mit Strukturelementen . . . 15
Anzeige von Objekten (BOs) . . . 15
Export der Daten aus einem Lesezeichen . . . 15
Kopieren eines Objektes aus einem Lesezeichen . . . 16
Anordnen und Organisieren von Strukturelementen . . . 16
Erstellen und Bearbeiten von Strukturelementen . . . 17
Glossar . . . 17
Referenz Tastaturkürzel . . . 18
Sichern und Wiederherstellen von Strukturelementen . . . 18
Ausführung von Skripts bei Server-Ereignissen . . . 20
Lesezeichen . . . 21
Sortierung . . . 21
Sortierung nach einer Spalte . . . 21
Sortierung nach mehreren Spalten . . . 22
Vordefinierte Sortierung . . . 22
Suchmöglichkeiten . . . 22
Volltextsuche . . . 23
Interaktive Filter . . . 23
Definition von Filtern allgemein . . . 23
Texteingabefelder (type="string") . . . 24
Eingabefelder für Zahlen (type="decimal") . . . 26
Eingabefelder für Datumswerte (type="date") . . . 26

Checkboxen zur Ja/Nein/Egal-Auswahl . . . 27
Auswahlboxen zur Auswahl aus mehreren Optionen . . . 28
Statische Multiple-Choice-Filter . . . 28
Dynamische Multiple-Choice-Filter mit choiceQuery . . . 29
Dynamische Multiple-Choice-Filter mit choiceScript . . . 31
Trenner . . . 31
OQL-Klauseln . . . 32
Beispiele . . . 32
Volltextsuche auf zusätzliche Felder ausdehnen . . . 33
Fest eingestellte Filter . . . 34
Eigene Query-Schablone . . . 35
Bedingungsgruppen ("constraint groups") . . . 35
Massenänderungen / Skripting . . . 37
"Transform Scripts" für die Abfrageresultate . . . 39
Sonstiges der Lesezeichen-XML-Definition . . . 40
Das Query-Element . . . 40
Formulare . . . 42
Eingabemöglichkeiten nach Datentypen . . . 42
Timespan (Zeitspanne) . . . 42
Altes Standardformat . . . 42
"Doppelpunkt"-Format(e) . . . 43
"Marker"-Format(e) . . . 44
Diverses . . . 44
Schablonen . . . 46
Erzeugen des neuen Objektes . . . 46
Reports . . . 48
Grundlagen . . . 48
Was ist ein Report überhaupt? . . . 48
Erstellung eines neuen Reports . . . 49
(Eingabe-)Parameter für Reports . . . 53
Die Anker-Definition oder: Wie komme ich an die Daten? . . . 54
virtualProperties in Reports . . . 55
Das CBOFormat und seine Verwendung im Report . . . 56
Troubleshooting . . . 57
Seitenwechsel / Überlappende Felder / "wachsende" Felder bei dynamischem Text . . . 57
Codebausteine . . . 58
Einbinden von Codebausteinen . . . 58
Reiter "CookedParameter" und "Codebausteine" . . . 60
Pfadangaben für Codebausteine . . . 60
Benamsung von Codebausteinen . . . 60
Inhalt von Codebausteinen . . . 61

`hideComment` beim Einbinden eines Codebausteines ... 62
Argumente für Codebausteine ... 63
Core-Codebausteine ... 64
jahrMonatTag.filter ... 64
Problembehebung ... 65
IllegalArgumentException: Invalid parameter "xyz" given... ... 65
Benachrichtigungen ... 66
Alarme ... 67
Grundlagen ... 68
Vorbereitung und Konfiguration ... 70
Alarmsystem-Lizenz einspielen ... 70
Alarmsystem aktivieren ... 70
Sync-Events behandeln ... 70
Benachrichtigungssystem aktivieren ... 70
Anlegen und Verwalten von Alarmen ... 71
Gruppe "Admins Alarmsystem" ... 71
Alarme aktivieren und deaktivieren ... 71
Testmodus für Alarme ... 71
Gemeinsame Eigenschaften aller Alarme ... 73
Erster Reiter ... 73
Reiter "Erweitert" ... 74
Einfacher Termin ... 75
Allgemeine Eigenschaften festlegen ... 75
Wann soll der einfache Termin stattfinden? ... 75
Vorwarnzeit ... 76
Wer soll Benachrichtigungen erhalten und wie sollen diese aussehen? ... 76
BO-basierter Termin ... 78
Allgemeine Eigenschaften festlegen ... 78
Welche Objekte sollen "überwacht" werden? ... 78
Exkurs: Vor- und Nachteile der verschiedenen BOMasken-Typen ... 78
Skript ... 79
Grooql-BOMasken ... 80
OQL-BOMasken ... 80
Wann soll der BO-basierte Termin (für ein Objekt) ausgelöst werden? ... 80
Auslösedatum aus Objekt-Attribut auslesen ... 81
Auslösedatum mit Skript berechnen ... 81
Wer soll Benachrichtigungen erhalten und wie sollen diese aussehen? ... 82
Automatische Neuterminierung nach Auslösung ... 83
Anhängen von (weiteren) Objekten ... 84
BOBasierterTermin-Status ... 85
Hinweise ... 86

Allgemeine Eigenschaften festlegen . . . 86
Welche Objekte sollen "überwacht" werden? . . . 87
Wann soll der Hinweis ausgelöst werden? . . . 87
Ignorierte BTs/Änderungen . . . 87
Auslösung bei beliebiger Änderung, Erstellen oder Löschen von Objekten (Unter-Reiter "Einfach") 88
Auslösung mittels Auslösekriterien (Unter-Reiter "Erweitert") . . . 88
Auslösung mittels Auslöseskript (Unter-Reiter "Skript") . . . 90
Mindestens eines oder alle gleichzeitig? . . . 91
Wer soll Benachrichtigungen erhalten und wie sollen diese aussehen? . . . 91
Von wem muss die Änderung stammen? . . . 91
Ab wann ist der Hinweis aktiv? . . . 92
Wiedervorlagen . . . 93
Allgemeine Eigenschaften festlegen . . . 94
Welche Objekte sollen "überwacht" werden? . . . 94
Wann soll die Wiedervorlage ausgelöst werden? . . . 94
Wer soll Benachrichtigungen erhalten und wie sollen diese aussehen? . . . 95
Wiedervorlage-Status . . . 95
Benachrichtigung bei Alarm-Auslösung . . . 96
Hartkodierte trigger()-Methode . . . 96
Benachrichtigungsskript "Sende Benachrichtigungen mittels dieses Skripts", Reiter "Erweitert" 96
Standard-Mechanismus . . . 98
Logging/Historie und `AlarmAusloesungen`-Objekte . . . 101
Sonstige Infos . . . 102
"Verpasste" bzw. "Verspätete" Auslösung . . . 102
Neuinitialisierung der Objekt-Status für BO-basierten Terminen und Wiedervorlagen . . . 102
CBOFormat . . . 103
Was ist CBOFormat? . . . 104
Abweichendes Attribut aus der Attributkette als Label verwenden . . . 106
Datum und Zeitwert-Formatierung . . . 107
Zahlen-Formatierung . . . 109
Funktionsaufrufe . . . 111
Script-Verwendung . . . 113
Wo kann man das CBOFormat nun überhaupt einsetzen? . . . 114
MEX - Makros und erweiterte Query-Funktionen . . . 115
Definition MEX . . . 116
Support auf Queryseite . . . 118
Support in Solstice . . . 119
Volltextsuche . . . 121
Vorbereitung und Konfiguration . . . 122

Volltextsuche aktivieren . . . 122
Einstellungen . . . 122
PostgreSQL: max_locks_per_transaction . . . 122
Betriebssystem: Mögliche Anzahl gleichzeitig offener Dateien . . . 122
indexAllByDefault . . . 122
indexDeletedBOs . . . 123
spellcheck . . . 123
fetchSize . . . 123
maxFieldLength und unlimitedFieldLength . . . 124
indexPath . . . 124
maxThreads . . . 125
directoryWrapper . . . 126
compassConfig . . . 126
Der Index . . . 127
Initiale Erstellung . . . 127
Erneute Erstellung / Re-Indexierung . . . 127
Verteilen des Index für synchronisierende Server . . . 128
Konfiguration für die in den Index aufzunehmenden Daten . . . 128
Benutzung der Volltextsuche . . . 129
Standard-Abfragen . . . 129
Einschränkungen der Entität . . . 129
Grooql (Groovy Object Query Language) . . . 130
Sprachumfang . . . 131
Beispiele für Filterskripte . . . 132
Einstellungen-Variablen . . . 133
Definition der vorhandenen/verfügbaren Variablen . . . 134
Abfrage von Einstellungen-Variablen in Skripten . . . 135
Setzen von abweichenden Werten für Benutzer oder Gruppen . . . 136
Lesezeichen und Anzeige in Benutzer- und Gruppen-Formularen . . . 137
Scripted Attributes . . . 138
Beispiele für Virtual Properties . . . 139
Caching . . . 142
Mögliche Cachemodi . . . 142
Neuberechnung bei `true` oder `VERSIONED` . . . 142
`cached`-Angabe direkt im Schema . . . 143
Positiv-Beispiel . . . 143
Negativ-Beispiel . . . 143
Standard-Werte . . . 145
Initialisierungsskript . . . 146
Probleme beim Start des Clients . . . 147
FAQ - Immer wiederkehrende Fragen und deren Beantwortung . . . 148

Benutzer-Passwort ändern / Change user password / Changer mot de passe 149
Benutzer-Passwort ändern 149
Change user password 149
Changer mot de passe 149
JavaWebstart-Cache löschen unter Windows 150
Anzeige der Symbole auf SVGs umstellen 151
Der Windows-Task-Manager zeigt mehr verwendeten Speicher an als der About-Dialog von MyTISM 152

Einleitung

MyTISM ist ein leistungsstarkes Framework zur Entwicklung und Verwaltung von Datenbankanwendungen. Es ist plattformunabhängig, objektorientiert, dezentral, multiuserfähig, individuell anpassbar und quelloffen. Mit MyTISM erhalten Sie ein 3-Tier-System inklusive GUI und Web-Application-Server, das Ihnen die Arbeit erheblich erleichtert. Es bietet eine umfassende Sammlung von Tools und Funktionen, die dabei helfen, komplexe Anwendungen effizient zu erstellen und zu verwalten. MyTISM wird entwickelt und betreut von der OAshi S.à r.l.

Dieses Handbuch führt Sie in die Grundlagen von MyTISM ein und zeigt Ihnen, wie Sie das Framework optimal nutzen und die damit erstellten Anwendungen bedienen.

Dieses Handbuch befindet sich noch in der Entwicklung. Wir arbeiten kontinuierlich daran, es zu vervollständigen und zu verbessern.

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

MyTISM: Ein starkes Fundament für Ihre Anwendung

Stellen Sie sich vor, Sie bauen ein Haus. MyTISM ist wie das Fundament, die Wände und das Dach, die Ihrem Haus Stabilität und Flexibilität geben. Es ist ein Framework, das Entwicklern hilft, Anwendungen zu erstellen, die **zuverlässig, anpassungsfähig und einfach zu warten** sind.

MyTISM teilt die Anwendung in drei Bereiche auf:

1. **Das Aussehen (Frontend):** Hier geht es um alles, was der Benutzer sieht und mit dem er interagiert, wie z.B. Buttons, Menüs und Formulare.
2. **Die Funktionen (Middleware):** Hier wird festgelegt, was die Anwendung **tut**, z.B. Daten verarbeiten, Berechnungen durchführen oder Informationen anzeigen.
3. **Die Daten (Backend):** Alle wichtigen Daten werden hier sicher gespeichert und verwaltet.

Was sind die Vorteile von MyTISM?

- **Übersichtlich und organisiert:** Wie ein gut aufgeräumtes Haus ist der Code der Anwendung strukturiert und leicht verständlich.
- **Flexibel und anpassbar:** Änderungen an einem Teil der Anwendung haben keine großen Auswirkungen auf andere Teile. So kann die Anwendung leichter an neue Anforderungen angepasst werden.
- **Stabil und zuverlässig:** MyTISM sorgt dafür, dass Ihre Anwendung robust und wartungsfreundlich ist.

Mit MyTISM bauen Entwickler Anwendungen, die wie ein solides Haus stabil, flexibel und zukunftssicher sind.

Vorstellung von MyTISM

Was bedeutet der Name "MyTISM"?

MyTISM steht für "**My T**ool **Is M**y...".

Sie können den Satzanfang mit dem ergänzen, was Ihnen am wichtigsten ist.
Zum Beispiel:

- My Tool Is My *Solution*: Mein Werkzeug ist meine Lösung.
- My Tool Is My *Key To Success*: Mein Werkzeug ist mein Schlüssel zum Erfolg.
- My Tool Is My *Inspiration*: Mein Werkzeug ist meine Inspiration.

MyTISM versteht sich dabei als universelles Software-Werkzeug.

Warum MyTISM?

Die Idee zu MyTISM entstand schon im August 2000. Doch dazu später mehr.

Stellen Sie sich vor, Sie möchten ein Haus bauen. Sie könnten natürlich jeden einzelnen Ziegelstein selbst formen und jeden Nagel von Hand schmieden. Aber es ist viel einfacher, fertige Ziegel, Nägel und Werkzeuge zu verwenden, oder?

Genauso ist es bei der Softwareentwicklung. Es gibt viele fertige "Bausteine" (Frameworks), die man verwenden kann, um Programme zu erstellen. Warum also haben wir uns die Mühe gemacht, MyTISM, unser eigenes Framework, zu entwickeln?

Ganz einfach: Weil wir keine passenden "Bausteine" gefunden haben, die all unsere Anforderungen erfüllt hätten. Damals haben wir festgestellt, dass die herkömmliche Art, Datenbankanwendungen zu entwickeln, sehr umständlich und fehleranfällig ist. Wir wollten einen besseren Weg finden, um Daten zu speichern und zu verwalten. Wir brauchten etwas, das flexibel, leistungsstark und einfach zu bedienen ist. Also haben wir angefangen, unsere eigenen "Bausteine" zu bauen.

Nach vielen Experimenten und Tests haben wir schließlich MyTISM entwickelt. Es basiert auf dem Prinzip der Objektorientierung und ermöglicht es uns, Daten als Objekte zu behandeln. Dadurch wird die Entwicklung von Datenbankanwendungen viel einfacher und intuitiver.

Und das Ergebnis kann sich sehen lassen! MyTISM ist das Herzstück unserer Softwareentwicklung. Es ist ein robustes und flexibles Framework, mit dem wir schnell und effizient maßgeschneiderte Software für unsere Kunden entwickeln können.

Was ist MyTISM genau?

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.

Was bringt die Zukunft?

MyTISM ist nicht stehen geblieben! Wir haben es ständig verbessert, neue Funktionen hinzugefügt und es noch leistungsstärker gemacht. Es hat sich schon in Projekten aus verschiedensten Bereichen vom Einzelhandel bis hin zur Industrieproduktion bewährt.

Und keine Sorge, wir haben noch viele Ideen, wie wir MyTISM in Zukunft noch besser für Sie machen können!

SOLSTICE - der Client

Solstice ist ein Frontend bzw. eine Benutzeroberfläche für MyTISM - oder besser gesagt, *das* Frontend, auch wenn, dank der modularen Bauweise von MyTISM, andere Frontends ohne weiteres möglich sind.

Grundlagen

FIXME TODO Solstice Client starten

Ansicht der Benutzeroberfläche

Die folgende Abbildung zeigt die Solstice-Oberfläche für den Benutzer "ERPTest".

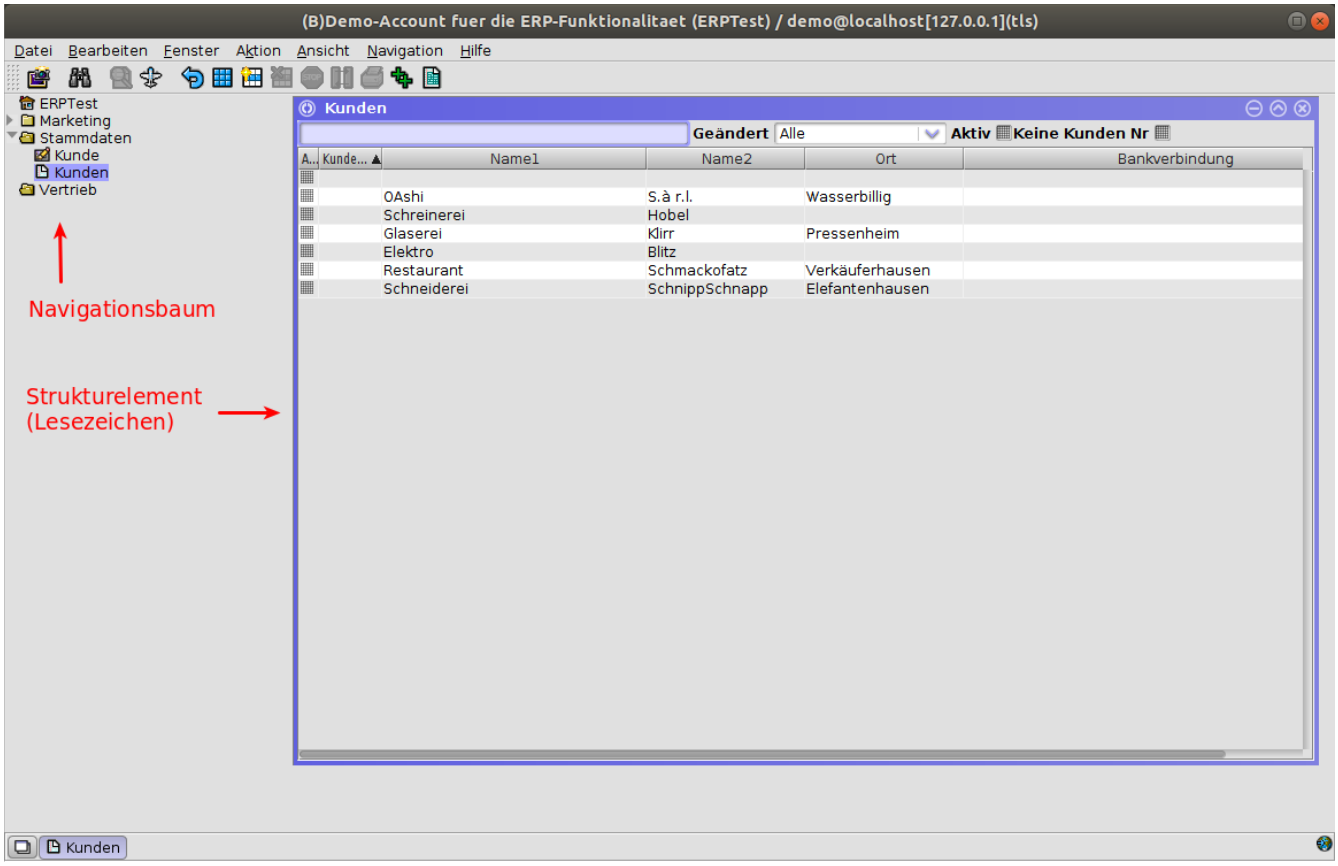


Abb. 1: Ansicht des Hauptfensters des Solstice-Clients im Einfenstermodus

Bereiche des Hauptfensters

Die Menüleiste befindet sich am oberen Rand des MyTISM-Solstice-Fensters. Nach Auswahl einer Menükategorie öffnet sich ein Untermenü mit weiteren Einträgen. (Ein schwarzer Pfeil zeigt an, dass das Menü noch weiter geschachtelt ist.) Die einzelnen Menüpunkte lassen sich entweder durch Klicken anwählen oder durch Tastaturkürzel aufrufen.

In jedem Hauptmenüpunkt ist ein Buchstabe unterstrichen. Tippt man diesen Buchstaben mit gedrückter "ALT"-Taste ein, öffnet sich das Untermenü. Die Kürzel zum Öffnen der Unterpunkte werden am rechten Rand der Menüpunktzeile angezeigt.

FIXME TODO weitere Beschreibung der Menüleiste (Kürzel, "2. Reihe" m. bildl. Symbolen beschreiben).

Unterhalb der Menüleiste auf der linken Seite ist der Navigationsbaum zu finden. Auf der freien

Fläche rechts davon werden die geöffneten sogenannten Strukturelemente angeordnet.

Mehrfachfenstermodus

Neben dem in der Abbildung gezeigten klassischen Einzelfenstermodus kann Solstice über die Menüleiste über den Menüpunkt (`Datei zum Mehfachfenstermodus wechseln`) alternativ im Mehrfachfenstermodus geöffnet werden. So wird jedes Element in einem eigenen Fenster geöffnet und kann frei angeordnet werden (u.a. lassen sich so die Elemente über mehrere Monitore verteilen und dort beliebig vergrößern).

Navigationsbaum

Der Navigationsbaum stellt eine wichtige Komponente der Solstice-Benutzeroberfläche dar, indem er für den jeweiligen Benutzer den Zugriff auf die für ihn verfügbaren Elemente strukturiert und somit eine benutzerspezifische Systemübersicht bietet.

Angezeigt werden im Navigationsbaum generell:

- Strukturelemente (`Ordner`,`Lesezeichen`, `Schablonen`, `Formulare`, `Reports` und `Aliase` darauf)
    - virtuelle Ordner
        - ein virtueller Ordner für den angemeldeten Benutzer
        - für Administratoren ein virtueller Ordner mit allen Benutzern
        - und nach dem Suchen von Strukturelementen ein virtueller Ordner mit Unterordnern für die Suchergebnisse.

Aussehen und Position von Elementen

Im Normalfall werden Elemente in Ordnern alphabetisch sortiert; es ist jedoch möglich, eine gewünschte Reihenfolge manuell festzulegen, indem man für das Element eine gewünschte Position einträgt. Elemente mit Position werden in der dadurch angegebenen Reihenfolge und vor allen Elementen ohne Position angezeigt.

Es ist außerdem möglich, Elemente durch zuweisen einer Hintergrundfarbe besonders hervorzuheben. Die Farbe muss HTML-kodiert angegeben werden.

Sichtbarkeit von Elementen

Welche Strukturelemente im Navigationsbaum für einen angemeldeten Benutzer sichtbar sind, wird von mehreren Faktoren gesteuert; u.a. im Zusammenspiel mit den von der [Rechteverwaltung] vergebenen Rechten.

Strukturelemente

`Strukturelement` ist der Oberbegriff für alle Elemente der Benutzeroberfläche, mit denen man Daten anzeigen und manipulieren kann. Dies sind Lesezeichen, Formulare, Schablonen, Codebausteine, Reports sowie Aliase und Ordner.

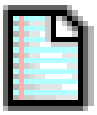
technischer Hintergrund:
MyTISM speichert die Daten in einer objektorientierten Datenbank. Datenelemente eines Typs werden jeweils in einer Datenbanktabelle zusammengefasst. Dabei können die Daten in einer Eltern-Kind Hierarchie angeordnet werden, so dass die Eigenschaften der übergeordneten Struktur auch für die 'Kindtabelle' gelten.
Beispiel: mögliche (Tabellen-)hierarchie für Belege:

```
Beleg    DebitorenBeleg    Rechnung    Endabrechnung
 Beleg    DebitorenBeleg    Auftrag
```

Wird für Objekte (- in MyTISM auch [BO]s genannt -) des Typs `Beleg` die Eigenschaft `Adressat` festgelegt, haben automatisch auch beispielsweise Datensätze des Typs `Endabrechnung` und `Auftrag` jeweils einen Adressaten.

Für Endanwender sind von den im folgenden beschriebenen Strukturelementen möglicherweise nur Lesezeichen, Formulare (bzw. die Aliase hierauf) und Reports interessant, während Schablonen und Codebausteine nur für diejenigen Anwender relevant sind, die selbst Strukturelemente (weiter-)entwickeln möchten.

Lesezeichen

Symbol	Beschreibung
	Lesezeichen zeigen in Tabellen- bzw Listenform eine Menge von Objekten (BOs) an. Die angezeigten Daten kann man mittels der Query-Zeile noch weiter einschränken / filtern (siehe "[Suchfunktion]").  Technisch gesprochen, handelt es sich bei einem Lesezeichen um eine gespeicherte Abfrage. Angezeigt werden alle Objekte aus einer Tabelle, die *nicht* als gelöscht markiert sind.

Formular

Symbol	Beschreibung
	Bei einem Formular handelt es sich um die Definition bezüglich der Darstellung von Daten: In einem Formular werden einzelne Objekte angezeigt, können dort aber auch bearbeitet oder neu angelegt werden. Durch das Formular wird festgelegt, in welchen Feldern die einzelnen Werte angezeigt werden, wie diese Felder angeordnet sind, usw.

Schablone

Symbol	Beschreibung
	Bei einer Schablone handelt es sich um die "Bauanleitung" für ein neues Objekt: Die Schablonendefinition legt fest, von welchem Typ das neu erzeugte Objekt sein soll und mit welchem Formular es dargestellt und bearbeitet werden soll.

Codebaustein

Symbol	Beschreibung
	Bei einem Codebaustein handelt es sich um ein eher technisches Strukturelement für Entwickler, das für den reinen Endbenutzer eher uninteressant ist, da es nicht direkt angezeigt wird: Ein Codebaustein ist im Prinzip ein Stück XML-Quellcode, welches man mit einer entsprechenden Anweisung in den Quellcode eines anderen Strukturelements einbinden kann. Dies dient dazu, doppelten Code zu vermeiden und gleiche, oft benötigte Quelltext-Teile zentral verwalten und ändern zu können.

Report

Symbol	Beschreibung
	Reports bieten Daten in einer druckbaren Form an. Möchte man z.B. eine Rechnung drucken, dann muss man das Aussehen und die Anordnung der Rechnungsdaten in Form eines Reports einmal definieren und kann fortan diesen für den Ausdruck (oder die Erstellung eines PDFs) verwenden. Reports werden in einem eigenen Kapitel ausführlicher beschrieben.

Alias

Symbol	Beschreibung
[solstice	*Definition:* Bei einem Alias handelt es sich um einen Verweis auf ein Strukturelement. *Verwendung:* Aliase können im Kontextmenü eines Elements im Navigationsbaum mittels dem Befehl `Verlinken` bzw. dem Tastaturkürzel `STRG+L` erzeugt werden und mittels `Einfügen` bzw. `STRG+V` an einer anderen Stelle im Navigationsbaum eingefügt werden. In der Praxis werden Aliase beispielsweise dazu genutzt, benutzer- bzw. rollenspezifische Ordner zu füllen. Ein 'Benutzerordner' dient dem jeweiligen Benutzer als zentrale, schnell zugreifbare Ansicht für die für ihn freigegebenen Strukturelemente, während das Originalstrukturelement an zentraler Stelle abgelegt ist. Aliase sind nicht zu verwechseln mit vollwertigen Kopien - ein Alias verweist immer auf ein Original und erbt dessen Rechte. Beim Doppelklick auf den Alias öffnet sich das Original, während eine Kopie natürlich ein eigenständiges Objekt ist, das separat gepflegt werden muss. (Ähnlich wie eine Kopie, aber das Objekt wird nicht wirklich kopiert; es wird lediglich ein "Verweis" auf das Originalobjekt. Alle Änderungen, die an einem der beiden vorgenommen werden, wirken sich auf "das andere" aus.

Arbeiten mit Strukturelementen

alias

Anzeige von Objekten (BOs)

Ein typischer, einfacher Arbeitsablauf, um ein Objekt anzusehen, beginnt häufig mit der Auswahl eines Lesezeichens im Navigationsbaum. Ein Doppelklick auf einen im Lesezeichen angezeigten Listeneintrag öffnet das dort beschriebene Objekt im zugehörigen Formular.

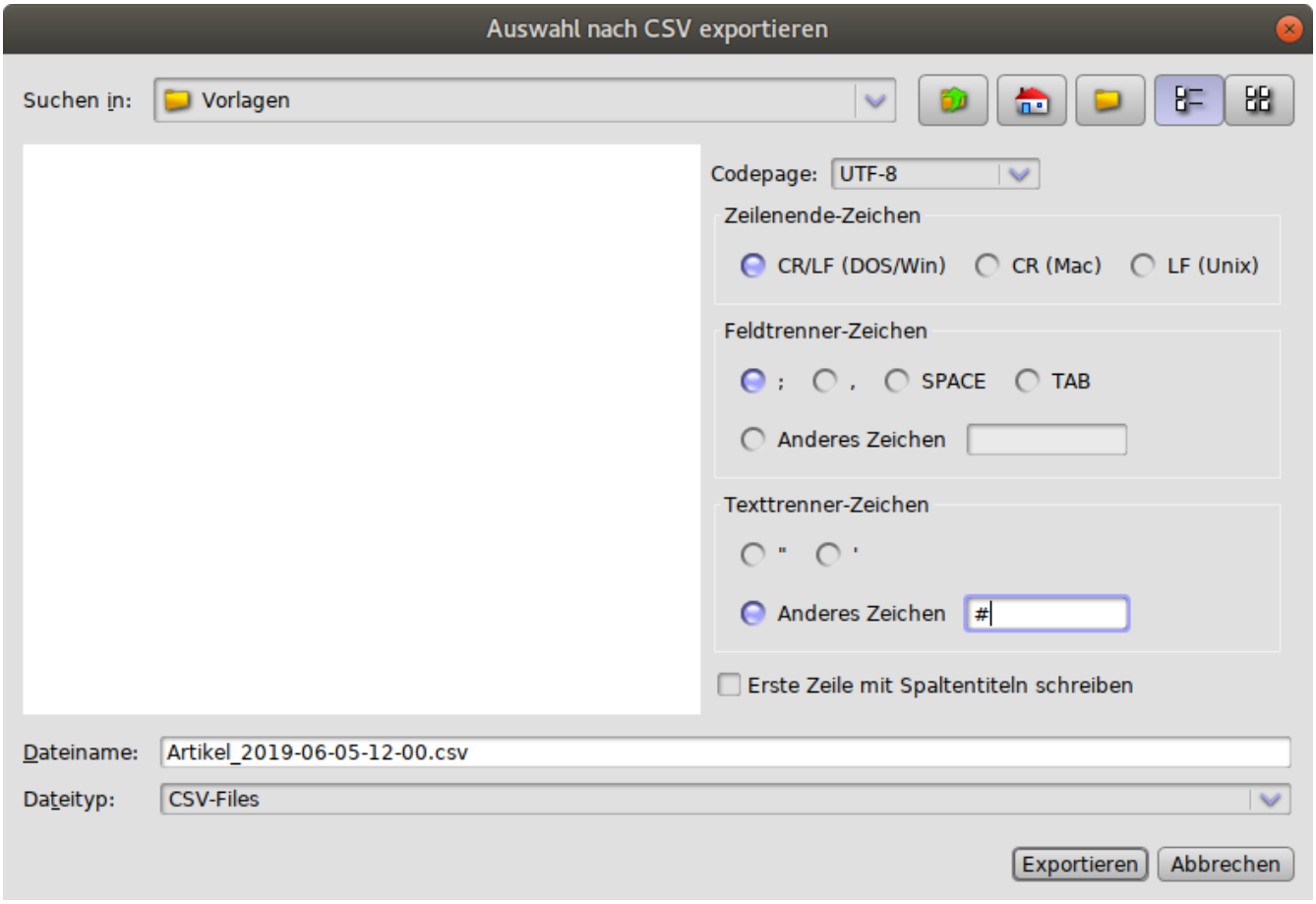
Es können mehrere Formulare für Objekte eines Typs existieren. Jedem Formular ist eine Priorität und ein BO-Typ zugewiesen. Sollten also mehrere Formulare existieren, mit denen das Öffnen des ausgewählten Objekts möglich ist, wird das Formular mit der höchsten Priorität gewählt. Haben mehrere passende Formulare die gleiche Priorität, wird das Formular bevorzugt, das vom BO-Typ her besser auf das zu öffnende Objekt passt, d.h. einem spezielleren passenden BO-Typen zugeordnet ist. Sollte es danach immer noch mehrere Formulare geben, die passen, wird zuerst nach Name und bei Gleichheit nach Id sortiert, um ein eindeutiges Formular zu bestimmen. Mit Hilfe der rechten Maustaste kann man sich alle (auf Grund der jeweiligen Berechtigung sichtbaren) zur Verfügung stehenden Formulare anzeigen lassen. So ist es auch möglich, ein Formular mit einer niedrigeren Priorität oder für einen allgemeineren BO-Typen auszuwählen.

Export der Daten aus einem Lesezeichen

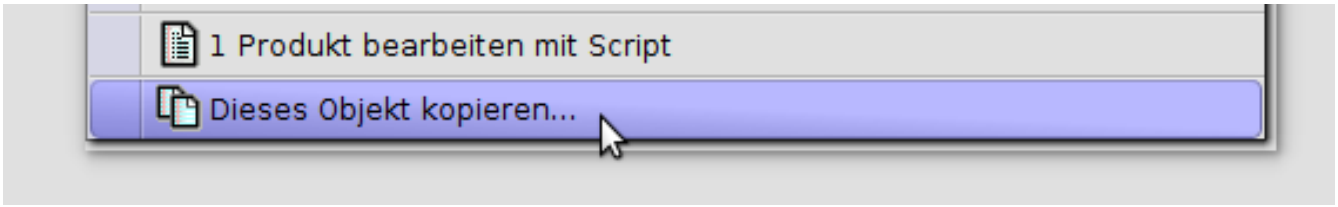
Es ist möglich die selektierten Daten aus einem Lesezeichen in eine Datei zu exportieren. Zur Auswahl steht das CSV- oder das XLS Format. Dazu werden zuerst die gewünschten Daten selektiert. Nach einem Klick auf die rechte Maustaste erscheint das entsprechende Kontextmenü.

Neben diversen Einstellungsmöglichkeiten bietet der CSV Export noch folgende Features:

- Die zur Auswahl stehenden Codepages können durch eine Einstellungsvariable vorgegeben werden. Hierzu existiert eine Variable mit dem Namen `csvExport.codepages` (diese wird vom Server implizit beim Start angelegt, sofern noch nicht vorhanden). Als Wert erhält diese Variable eine Liste von Codepages, welche durch ein Komma getrennt sind, z.B.: `UTF-8,Windows-1252,ISO-8859-1,ISO-8859-15`. Der erstgenannte Wert ist der Default.
- Die Einstellungen des CSV Exportes werden lokal für den jeweiligen Benutzer gespeichert. Beim nächsten Mal sind diese standardmäßig vorgewählt.
- Zum Abspeichern wird ein Dateiname vom System vorgeschlagen. Dieser besteht aus dem Entität-Namen der zu exportierenden Tabelle, der aktuellen Uhrzeit und der Dateiendung `.csv`.



Kopieren eines Objektes aus einem Lesezeichen



Aus dem Kontextmenü eines Lesezeichen kann ein BO kopiert werden.
Damit der Menüpunkt zur Verfügung steht, muss dem jeweiligen Benutzer (bzw. einer Gruppe des Benutzers) eine Schablone für den zu kopierenden Objekttyp zugewiesen sein.

Anordnen und Organisieren von Strukturelementen

Strukturelemente können in Solstice zwischen verschiedenen Ordnern verschoben und kopiert werden, und es können sog. [alias]>Aliase (Verknüpfungen) angelegt werden. Dies geschieht üblicherweise über den Navigationsbaum, indem man mit der linken Maustaste auf das Strukturelement anwählt und es - mit weiterhin gehaltener Maustaste - an die gewünschte Stelle "zieht".

Hält man beim Loslassen keine weitere Taste gedrückt, wird eine Verknüpfung erstellt. Hält man beim Loslassen die Taste STRG gedrückt, so wird das Objekt verschoben; hat man die Taste ALT gedrückt, so wird das eine Kopie des Objekts an dieser Stelle angelegt. Automatik-Elemente können nur kopiert werden; einen Alias zu erstellen oder das Element zu verschieben wird komplett ignoriert.

Erstellen und Bearbeiten von Strukturelementen

Die Lesezeichen-, Formular- und Schablonen selbst können bearbeitet werden, indem man entweder das Strukturelement anwählt und ALT+EINGABE drückt oder im Kontextmenü des Objekts (Objekt anwählen, rechte Maustaste drücken) den Menüpunkt Information wählt. Voraussetzung zum Editieren sind Schreibrechte, die durch den Systemadministrator für jeden Benutzer pro BO vergeben werden können.

Glossar

FIXME Verschieben ans Ende der User-doku. Es handelt sich hier nicht (nur) um Solstice-spezifische Begriffe, sondern um solche, die für das Verständnis von MyTISM im allgemeinen wichtig sind. Eigenes .ad-Dokument hieraus erzeugen

BO / CBO / SBO

BO ist die Abkürzung für "Business Object" - jedes Objekt ("Datensatz") in MyTISM ist ein BO. Jedes BO hat einen BO-Typ, welcher die Eigenschaften des BOs definiert.
Beispiel
Eine "Person" ist ein BO vom Typ "Person" und hat z.B. die Felder "Vorname", "Nachname", "Geschlecht", ...
BOs werden der Übersichtlichkeit halber nochmal unterteilt in "Complex Business Object" (CBO) und "Simple Business Object" (SBO). Quertabellen wie z.B. "Geschlecht", die nur wenige Einträge (wie in diesem Fall "männlich" und "weiblich") haben, sind typische Vertreter für ein SBO. Eine "Rechnung" ist da schon was komplexeres und demzufolge vom Typ "CBO".

Quertabelle

Quertabellen sind Nachschlagetabellen, die hauptsächlich vorinitialisierte Daten enthalten. Es handelt sich hierbei häufig um für das System zentrale Daten, die sich selten ändern und deren Werte bereits bekannt sind. Ein typisches Beispiel für solche Daten sind Einheiten. Zentrale Einheiten wie bestimmte Maßeinheiten und Gewichte (Gramm, Kilogramm etc.) werden bereits durch das ERP-Modul bereitgestellt. Solche vordefinierten Quertabellen können aber prinzipiell durch berechtigte Benutzer jederzeit erweitert werden.

Schema

FIXME (Entscheiden, ob dieser Begriff hier aufgenommen wird; evtl zu technisch u. eher f. Developer-Doku relevant)

Virtual Attributes / Scripted Attributes

MyTISM bietet die Möglichkeit, im laufenden Betrieb Datenfelder in Formulare, Lesezeichen und Reports nachzubauen. Diese nennt man `Virtual Attributes` oder auch `Scripted Attributes`.

Für alle im Schema der jeweiligen MyTISM-Installation definierten BOs werden beim Start des Servers automatisch jeweils ein Lesezeichen (das alle BOs der entsprechenden Klasse anzeigt) sowie ein Formular und eine Schablone erstellt. Daneben existieren für manche Klassen auch noch angepasste, "schönere" vorgebaute Strukturelemente, die ebenfalls automatisch eingespielt werden.

Referenz Tastaturkürzel

to be continued

F2

Funktion: Speichern
Wo: Formular

F3

Funktion: Speichern und Schliessen
Wo: Formular

F4

Funktion: Popup aufklappen
Wo: Formular

F5

Funktion: Aktualisierung der Daten/Anzeige
Wo: Lesezeichen, Menü-Baum

ESC

Funktion: Ansicht schliessen
Wo: Formular, Lesezeichen

STRG-F

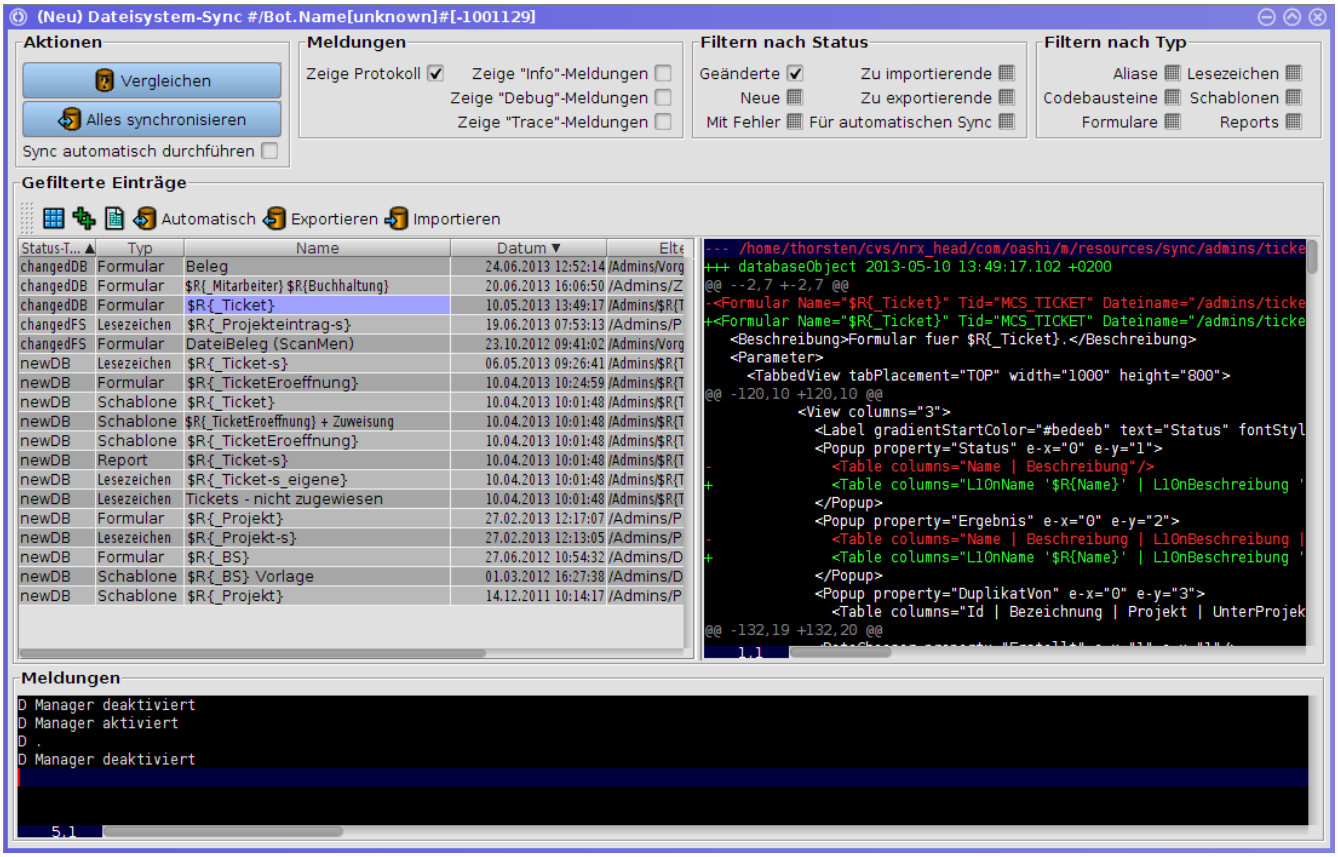
Funktion: Suchen (Strukturelemente: Formular, Lesezeichen, Report, ...)
Wo: überall

STRG-S

Funktion: Speichern
Wo: Formular

Sichern und Wiederherstellen von Strukturelementen

Unter dem Menüpunkt `Entwicklung` gibt es die Funktion `Struktur-Synchronisation` . Hiermit werden alle Strukturelemente (Formulare, Lesezeichen, Schablonen, Reports, etc.), bei denen ein (im Prinzip frei wählbarer) Dateiname definiert ist als XML-Dateien in einem Verzeichnis gespeichert bzw. Strukturelemente aus diesen Dateien wieder in die Datenbank eingespielt.



Die Bedienung sollte größtenteils selbsterklärend sein.

- Mit den diversen `Filtern` ist es möglich, die Liste nach vorgegebenen Kriterien auszudünnen.
- Unter `Meldungen` kann man die Anzeige der Log-Meldungen aktivieren und angeben, wie genau man dort über die Vorgänge informiert werden will.
- Der Knopf `Vergleichen` erlaubt es, die Liste manuell zu aktualisieren.
- Der Knopf `Alles synchronisieren` exportiert bzw. importiert automatisch alle Strukturelemente, abhängig von ihrem Status und speichert danach auch automatisch die entstandenen Änderungen ab.
- `Sync automatisch durchführen` überwacht Datenbank und Verzeichnis selbsttätig auf Änderungen und synchronisiert diese automatisch. ***FIXME: Es kann sein, dass das noch nicht ganz korrekt funktioniert - Funktion wird fast nie benutzt.***

Damit die exportierten Objekte auch einigermassen geordnet in Unterverzeichnissen liegen, die ihrem Ordnernamen in Solstice entsprechen, sollte man dies im Dateinamen mit angeben. So würde man für das Formular "MeinFormular", welches im Ordner "EigeneFormulare" liegt z.B. folgenden Dateinamen eintragen: `EigeneFormulare/MeinFormular`. Die vorgebauten Formulare für Strukturelemente bieten einen Knopf "Dateiname vorschlagen" mit welchem man einen aus dem Elterpfad generierten Dateinamen automatisch eintragen lassen kann.

Die exportierten Objekte enthalten je nach Typ folgende Kürzel:

- `bkm`: Lesezeichen (für engl. "Bookmark")
- `frm`: Formular (für engl. "Form")
- `tpl`: Schablone (für engl. "Template")

- `rpt`: Report (für Reports werden aus technischen Gründen übrigens zwei Dateien abgespeichert, die zweite der beiden Dateien hat gar kein "Mittelkürzel")

uieleme

- `bst`: Codebaustein

Beim Sync der AnkerDefinition von Reports werden mehrfache Leerzeichen zwischen XML-Attributen von Tags nicht beim Diff beachtet. Außerdem werden Kommentare außerhalb des Wurzelknotens (ganz am Anfang oder ganz am Ende des XML-Dokuments) ignoriert.

Ausführung von Skripts bei Server-Ereignissen

Im Normalfall werden bei Server-Ereignissen, wie Herunterfahren oder Systemnachrichten voreingestellte Aktionen ausgeführt; meist wird (nur) eine Nachricht angezeigt. Mittels im Benutzer-Profil definierter Skripts kann man jedoch auch in anderer Weise auf diese Ereignisse reagieren. Beispiel:

```
<Configuration>
  <Profile name="default">
    <onSystemMessage>_client.log.warn("Systemmessage: " + _msg +
".")</onSystemMessage>
    <onShutdownInitiated>_client.log.warn("Shutdown initiated: " + _msg + " in " +
_cSecsDelay + " seconds.")</onShutdownInitiated>
    <onShutdownStopped>_client.log.warn("Shutdown stopped.")</onShutdownStopped>
    <onShutdown>_client.log.warn("Server has been shut down.");
_client.close()</onShutdown>
    <!-- Sonstiger Profil-Code -->
  </Profile>
</Configuration>
```

Folgende Möglichkeiten stehen zur Verfügung:

- onSystemMessage: Wird aufgerufen, wenn eine Systemnachricht angekommen ist. Die Variable `_msg` enthält den Nachrichtentext.
- onShutdownInitated: Wird aufgerufen, wenn die Bennachrichtigung über ein bevorstehendes Herunterfahren des Servers angekommen ist. Die Variable `_cSecsDelay` enthält die Anzahl der Sekunden, die das Herunterfahren noch entfernt ist; `_msg` enthält ggf. den Text einer zusätzlichen Information zum Herunterfahren, sofern einer mitgeliefert wurde.
- onShutdownStopped: Wird aufgerufen, wenn das Herunterfahren aus irgendeinem Grund abgebrochen wurde.
- onServerLocked: Wird aufgerufen, wenn der Server gesperrt (keine Anmeldungen mehr erlaubt) wurde.
- onServerUnlocked: Wird aufgerufen, wenn der Server wieder entsperrt wurde.

Lesezeichen

"Lesezeichen" ist die MyTISM-Bezeichnung für vordefinierte Datenbank-Abfragen in der Solstice-Benutzeroberfläche.

Im einfachsten Fall können in einem Lesezeichen bestehende Objekte aus der Datenbank mit einer (eingeschränkten) Volltextsuche gesucht und in Tabellenform angezeigt werden.

Neben der Suche mit Suchbegriffen können auch zusätzliche sog. *Filter* definiert werden, die z.B. über eine Auswahlliste eine weitere Einschränkung der Suchergebnisse ermöglichen.

Für jedes Lesezeichen ist festgelegt, welcher Typ von Objekten damit abgefragt werden kann, wobei im Normalfall dann natürlich auch alle ggf. definierten Untertypen eingeschlossen sind.

Die gefundenen Objekte können dann aus dem Lesezeichen heraus zur Detailansicht oder Bearbeitung geöffnet werden.

Außerdem können mittels sog. *Massenänderung* Datenänderungen an mehreren oder allen der gefundenen Objekte gleichzeitig vorgenommen werden.

Die in der Tabelle angezeigten Daten der Objekte können als CSV- oder Excel-Datei exportiert oder in die Zwischenablage kopiert werden.

Weiterhin können spezielle *Aktionen* definiert werden, die dann mit mehreren oder allen der Objekte vorprogrammierte Dinge tun.

Bei den meisten Werten für XML-Elemente und -Attribute für die XML-Definition des Lesezeichens ist die Groß-/Kleinschreibung wichtig und sie sollten genau so eingegeben werden, wie hier aufgeführt. "Historisch gewachsen" ist die Schreibweise leider nicht einheitlich und so müssen manche Werte groß und manche klein geschrieben werden.

Die meisten der hier beschriebenen Möglichkeiten sind nicht nur in Lesezeichen, sondern allgemein in allen `Table`-XML-Elementen verfügbar; so insb. z.B. auch in der Auswahlliste von GUI-Auswahlboxen (`<Popup`).

Sortierung

Die in Lesezeichen angezeigten Ergebnisse können nach Wunsch sortiert werden.

Sortierung nach einer Spalte

Soll nur nach einer Spalte sortiert werden, kann man hierzu einfach mit der Maus auf den Titel der Spalte klicken. Ein weiterer Klick kehrt die Sortierreihenfolge um. Ein weiterer Klick hebt dann wieder die Sortierung dieser Spalte auf.

Sortierung nach mehreren Spalten

Auch eine Sortierung nach mehreren Spalten ist möglich; hält man beim Klick auf einen Spaltennamen die STRG/CTRL-Taste gedrückt, so werden bisher definierte Sortier-Spalten beibehalten.

Die Ergebnisse werden zuerst nach der zuerst ausgewählten Spalte sortiert; wenn für Objekte der Wert dieser Spalte gleich ist, werden diese Objekte dann nach der als zweites ausgewählten Spalte sortiert; usw.

Die Reihenfolge, in der die Spalten sortiert werden, ist anhand der Größe der Symbole zu erkennen; nach der Spalte mit dem größten Symbol wird zuerst sortiert.

Vordefinierte Sortierung

Es ist möglich, eine Sortierung dauerhaft bzw. als Standard-Einstellung im Lesezeichen zu definieren.

Hierzu wird die Spaltendefinition innerhalb der Tabellendefinition erweitert. Wird die Kurznotation der Tabellenspalten genutzt, können die Schlüsselwörter `ASC` (für aufsteigende Sortierung, also kleiner → größer, älter → jünger oder A → Z) und `DESC` (für absteigende Sortierung, also größer → kleiner, jünger → älter oder Z → A) durch Komma getrennt hinter den Attributnamen geschrieben.

Die Reihenfolge bzw. Priorität der Sortierung kann als Zahl direkt hinter `ASC` oder `DESC` geschrieben werden und muss innerhalb der Spaltendefinitionen eindeutig sein.

Beispiel für Standard-Einstellung Sortierung erst absteigend nach Belegdatum, dann aufsteigend nach Kunde

```
<Table entity="Rechnung">
  <Query type="Text"/>
  <View>
    <Columns>
      Kunde, ASC2
      Belegdatum, DESC1
      Netto 'Netto-Betrag'
      Brutto 'Brutto-Betrag'
      Bankeinzug
    </Columns>
  </View>
</Table>
```

In der ausführlichen Notation für die Spalten werden die XML-Attribute `sort` und `sortLevel` in ähnlicher Weise benutzt.

Suchmöglichkeiten

Volltextsuche

In der Eingabezeile oberhalb der Tabelle können Suchbegriffe eingegeben und durch Drücken von Return/Enter die Volltextsuche gestartet werden. Es werden alle Objekte des vordefinierten Typs gefunden, bei denen einer der eingegebenen Begriffe in einem der Text- oder Zahlenfelder des Objekts vorkommt.

Neben der vordefinierten Menge an Feldern können außerdem zusätzliche Felder definiert worden sein, die dann ebenfalls durchsucht werden.

Wenn man direkt vor einem Suchbegriff ein Pluszeichen `+` eingibt bedeutet das, dass der Begriff vorkommen *muss*.

Ein direkt vorangestelltes Minuszeichen `-` bedeutet umgekehrt, dass der Begriff *nicht vorkommen darf*.

Interaktive Filter

Neben der Volltextsuche können Lesezeichen mit zusätzlichen Eingabemöglichkeiten ausgestattet werden, mit denen weitere Einschränkungen für die Ergebnismenge definiert werden.

Es gibt mehrere Typen dieser sogenannten *Filter*:

- Texteingabefelder
- Eingabefelder für Zahlen
- Eingabefelder für Datumswerte
- Checkboxen zur Ja/Nein/Egal-Auswahl
- Auswahlboxen zur Auswahl aus mehreren Optionen

Definition von Filtern allgemein

Alle diese Filter können mittels `<filter>`-Kindelementen des `<Query>`-Elements erzeugt werden und werden dann als Eingabefeld, Checkbox, usw. im Lesezeichen angezeigt.

Beispiel für ein Texteingabefeld zur Einschränkung der Ergebnismenge auf Dokumente mit einer bestimmten Nummer:

```
<Query type="Text">
  <filter type="string" title="Dokumentnummer" cols="30">
    <clause>Dokumentnummer = "{}"</clause>
  </filter>
</Query>
```

Folgende XML-Attribute sind dabei für jeden Filtertyp verfügbar:

type

Verpflichtend - Was für ein Typ von Filter erzeugt werden soll; Mögliche Werte sind `string` für Texteingabefelder, `decimal` für Eingabefelder für Zahlen, `date` für Eingabefelder für

Datumswerte, `bool` für Checkboxen zur Ja/Nein/Egal-Auswahl und `multipleChoice` für Auswahlboxen zur Auswahl aus mehreren Optionen.

nts/solst

title

(Meist) Optional - Wird als Beschriftung der Filterkomponente benutzt; falls nicht angegeben wird stattdessen die `clause` (s.u.) als Titel verwendet.

name

Optional - Wird für einige Fehlermeldungen und im Zusammenhang mit "dependent"-Filtern benutzt FIXME

variable

Optional - FIXME

group

Optional - Mit diesem Attribut ist es möglich, den Filter einer sog. Bedingungsgruppe zuzuordnen. Wird es nicht angegeben gehört der Filter zur Standardgruppe, die in der Query-Schablone mit "{=constraints}" angesprochen wird.

grabFocus

Optional - Kann "true" oder "false" (der Standard) sein; der in der Reihenfolge der Definitionen erste Filter mit `grabFocus="true"` erhält nach dem Öffnen des Lesezeichens direkt den Eingabefokus.

dependsOn

Optional - (FIXME Filter können voneinander abhängen)

Texteingabefelder (type="string")

In "string"-Filtern können Zeichenketten angegeben werden, die in der Suche verwendet werden sollen.

Folgende XML-Attribute können speziell für "string"-Filter benutzt werden:

cols

Optional - Die bevorzugte Breite des Eingabefeldes, in Zeichen.

Folgende XML-Kindelemente können speziell für "string"-Filter benutzt werden:

clause

Verpflichtend - Die OQL-Klausel, die in die Datenbankabfrage eingefügt wird, wenn in diesem Filter ein Wert angegeben wird. Innerhalb dieser Klausel kann `{}` (zwei geschweifte Klammern, ohne Inhalt) als Platzhalter verwendet werden; hier wird dann bei der Abfrage der im Filterfeld eingegeben Wert eingesetzt. Bei Verwendung von Bedingungsgruppen können auch mehrere dieser `clause`-Elemente für einen Filter verwendet werden, nähere Erklärungen im Abschnitt Bedingungsgruppen.

ifEmpty

Optional - Eine OQL-Klausel, die in die Datenbankabfrage eingefügt wird, wenn im Filter *kein*

Wert eingegeben wurde oder *keine* `clause` definiert wurde.

ice-alias

inputPreprocessor

Optional - In diesem XML-Element kann ein Groovy-Skript angegeben werden, dass die Benutzereingabe im Filter noch modifizieren kann, bevor sie für die Datenbankabfrage benutzt wird. Das Skript wird ausgeführt *bevor* der Platzhalter `{}` ersetzt wird und muss eine Zeichenkette zurückliefern, die dann anstelle der ursprünglichen Benutzereingabe verwendet wird.

Im Skript stehen zwei vordefinierte Variablen zur Verfügung:

- `input` - Der im Filterfeld eingegebene Wert als String
- `bol` - Ein BOLoaderI

Beispiel für einen "string"-Filter, in dem ein oder mehrere durch Komma getrennte Dokumentnummern eingegeben werden können um Dokumente mit einer der eingegebenen Nummern zu finden:

```
<Query type="Text">
  <filter type="string" title="Dokumentnummern" cols="12">
    <clause>Nummer IN LIST({})</clause>
    <inputPreprocessor>
      // Leerzeichen von den Nummern entfernen, in Hochkommata einschliessen und
wieder als Komma-getrennte Liste zurückgeben.
      input.split(',').collect{ "'${it.trim()}'" }.join(',')
    </inputPreprocessor>
  </filter>
</Query>
```

Beispiel für einen "string"-Filter, in dem aus der User Eingabe die Nummer extrahiert und mit einem Prefix für die Suche versehen wird:

```
<Query type="Text">
  <filter type="string" title="Dokumentnummern" cols="12">
    <clause>Nummer = "{}"</clause>
    <inputPreprocessor>
        // RegEx für Nummern
        def number = /\d+/
        def matcher = (input =~ number)
        // Falls eine Nummer eingegeben wurde, diese extrahieren und mit dem Prefix
'D' versehen für die Eingabe zurückgeben
        matcher.find() ? "D ${matcher[0]}" : input
        // Alternativ als Einzeiler.
        // (input =~ /\d+/).findResult{ "D $it" } ?: input
    </inputPreprocessor>
  </filter>
</Query>
```

Eingabefelder für Zahlen (type="decimal")

In "decimal"-Filtern können Zahlen angegeben werden, die in der Suche verwendet werden sollen. Im Gegensatz zu "string"-Filtern, bei denen die Eingabe unverändert und ungeprüft übernommen wird, wird bei "decimal"-Filtern versucht, die Eingabe als Zahl zu interpretieren; falls das nicht möglich ist, wird eine Fehler angezeigt.

Folgende XML-Attribute können speziell für "decimal"-Filter benutzt werden:

cols

Optional - Die bevorzugte Breite des Eingabefeldes, in Zeichen.

Folgende XML-Kindelemente können speziell für "decimal"-Filter benutzt werden:

clause

Verpflichtend - Die OQL-Klausel, die in die Datenbankabfrage eingefügt wird, wenn in diesem Filter ein Wert angegeben wird. Innerhalb dieser Klausel kann `{}` (zwei geschweifte Klammern, ohne Inhalt) als Platzhalter verwendet werden; hier wird dann bei der Abfrage der im Filterfeld eingegeben Wert eingesetzt. Bei Verwendung von Bedingungsgruppen können auch mehrere dieser `clause`-Elemente für einen Filter verwendet werden, nähere Erklärungen im Abschnitt Bedingungsgruppen.

ifEmpty

Optional - Eine OQL-Klausel, die in die Datenbankabfrage eingefügt wird, wenn im Filter *kein* Wert eingegeben wurde oder *keine* `clause` definiert wurde.

Eingabefelder für Datumswerte (type="date")

In "date"-Filtern können Datumswerte eingeben werden, die in der Suche verwendet werden sollen. Ähnlich wie bei "decimal"-Filtern wird auch hier geprüft, ob die Eingabe, ausgehend von einem definierten Eingabeformat, als ein korrekter Datumswert interpretiert werden kann.

Folgende XML-Attribute können speziell für "decimal"-Filter benutzt werden:

cols

Optional - Die bevorzugte Breite des Eingabefeldes, in Zeichen.

replace

Optional - Wenn "true" (der Standardwert), wird die Benutzereingabe im Eingabefeld (FIXME wann?) ersetzt durch das angegebene Datum, aber formatiert mit dem Datumsformat wie es bei `format` spezifiziert ist. Mit "false" wird die Benutzereingabe beibehalten exakt wie sie eingegeben wurde.

format

Optional - Wenn angegeben wird hierdurch eine Standardformatierung definiert, in der Datumswerte - zusätzlich zu allen sonstigen Formaten (FIXME erklären) - im Feld eingegeben werden können. Ist `replace` aktiviert, wird die Benutzereingabe umformatiert, so dass sie dem hier gegebenen Format entspricht.

quickLookup

FIXME

folder.gif

strictFormat

Ähnlich wie bei `format` wird hier Standardformatierung definiert, in der Datumswerte im Feld eingegeben werden können. Wird allerdings `strictFormat` benutzt, wird *nur* diese Formatierung unterstützt, *kein* anderes Format ist dann erlaubt. Ist `replace` aktiviert, wird die Benutzereingabe umformatiert, so dass sie dem hier gegebenen Format entspricht.

Folgende XML-Kindelemente können speziell für "string"-Filter benutzt werden:

clause

Verpflichtend - Die OQL-Klausel, die in die Datenbankabfrage eingefügt wird, wenn in diesem Filter ein Wert angegeben wird. Innerhalb dieser Klausel kann `{}` (zwei geschweifte Klammern, ohne Inhalt) als Platzhalter verwendet werden; hier wird dann bei der Abfrage der im Filterfeld eingegeben Wert eingesetzt. Bei Verwendung von Bedingungsgruppen können auch mehrere dieser `clause`-Elemente für einen Filter verwendet werden, nähere Erklärungen im Abschnitt Bedingungsgruppen.

ifEmpty

Optional - Eine OQL-Klausel, die in die Datenbankabfrage eingefügt wird, wenn im Filter *kein* Wert eingegeben wurde oder *keine* `clause` definiert wurde.

format

FIXME (weitere, zusätzliche Eingabeformate definieren)

Checkboxen zur Ja/Nein/Egal-Auswahl

BoolFilterGUI when 'ifTrue' then do when 'ifFalse' then do when 'ifNull' then do

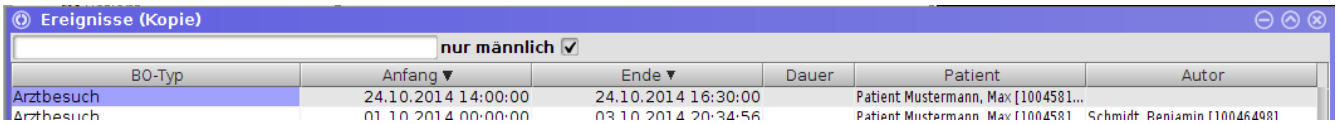
Ein Boolescher Filter erscheint als Checkbox.

Beispiel Checkbox-Filter:

```
<Query type="Text">
  <filter type="bool" title="nur männlich">
    <ifTrue>
      Geschlecht.Tid = "MAENNLICH"
    </ifTrue>
    <ifFalse>
      Geschlecht.Tid = "WEIBLICH" or Geschlecht.Tid = "NA"
    </ifFalse>
    <ifNull>
      Geschlecht = null
    </ifNull>
  </filter>
</Query>
```

Das `Query`-Tag enthält hier einen Filter, der auf Wunsch alle weiblichen (eigentlich: alle nicht-

männlichen) Personen herausfiltert.



Auswahlboxen zur Auswahl aus mehreren Optionen

MultipleChoiceFilterGUI when 'nullable' then when 'sort' then when 'preselectIdx' then do

```
when 'clause'       then
when 'choice'       then
when 'choiceScript' then
when 'choiceQuery'  then
when 'setupScript'  then
```

Ein Multiple-Choice-Filter erscheint in seinem Formular als Combo-Box.

Statische Multiple-Choice-Filter

Beispiel statischer MultipleChoice-Filter:

```
<Query type="text">
  <filter type="multipleChoice" title="Auswahl">
    <choice title="Alle"></choice>
    <choice title="Nur Rechnungen">Bot.Name = "Rechnung"</choice>
    <choice title="Nur Direktverkaeufe">Bot.Name = "Direktverkauf"</choice>
  </filter>
</Query>
```

Beispiel statischer MultipleChoice-Filter mit vordefinierter identischer WHERE-Klausel:

```
<Query type="text">
  <filter type="multipleChoice" title="Auswahl">
    <clause>Bot.Name="{}"</clause>
    <choice title="Alle"></choice>
    <choice title="Nur Rechnungen">Rechnung</choice>
    <choice title="Nur Direktverkaeufe">Direktverkauf</choice>
  </filter>
</Query>
```

Hier agiert `{}` in der `clause` als Platzhalter für einsetzbare Werte, die in `choice`-Tags angegeben sind. Bei "Alle" (leeres Tag) erhält es eine Wildcard-Funktion.

Beispiel MultipleChoice-Filter mit SQL-Funktionen (hier Datumsberechnung):

```
<Query>
    <filter type="multipleChoice" title="$R{Geschrieben}">
      <choice title="$R{seitHeute}"><![CDATA[
        age(date_trunc("day", BuchungsDatum))<"1 days"
     ]]></choice>
      <choice title="$R{seitGestern}"><![CDATA[
        age(date_trunc("day", BuchungsDatum))<"2 days"
     ]]></choice>
      <choice title="$R{letzteWoche}"><![CDATA[
        age(date_trunc("day", BuchungsDatum))<"7 days"
     ]]></choice>
      <choice title="$R{letztenMonat}"><![CDATA[
        age(date_trunc("day", BuchungsDatum))<"30 days"
     ]]></choice>
      <choice title="$R{irgendwann}"/>
    </filter>
</Query>
```

Dynamische Multiple-Choice-Filter mit choiceQuery

Es ist auch möglich, dynamische Multiple-Choice-Filter mit Hilfe einer Query anzugeben.

Beispiel dynamischer MultipleChoice-Filter:

```
<Query>
  <!-- Liste enthaelt alle Filialen mit gesetzter Tid und zeigt in der Liste den
Kurznamen der Filiale an -->
  <filter type="multipleChoice" title="Filiale">
    <choiceQuery query="Filiale a WHERE Not Ldel And Not is_undefined(Tid)"
                 format="Kurzname">
      Filiale.Kurzname = "{Kurzname}"
    </choiceQuery>
  </filter>
</Query>
```

Das Resultat der choiceQuery wird Wert für Wert als Filtereinträge im Formular angezeigt. Statt der Angabe eines "choice-title" werden die Resultate mittels des "format"-Attributs formatiert und als Auswahlwerte angezeigt.

Abhängigkeiten für dynamische Multiple-Choice-Filter mit choiceQuery

Multiple-Choice-Filter, die ihre Werte per choiceQuery ermitteln, können Abhängigkeiten zu anderen Filtern definieren und aufgrund der darin gesetzten Werte ihre eigene Auswahl modifizieren.

Die Abhängigkeiten werden mittels des Attributs `dependsOn` angegeben. Es können ein oder durch Komma getrennt auch mehrere andere Filter über ihren Namen als Abhängigkeiten definiert werden.

Ändert sich der Wert in einem Filter, von dem man abhängig ist, so werden die Werte automatisch aktualisiert.

Per Attribut `dependsOnQuery` wird die Query angegeben, mit der die Werte ermittelt werden, inklusive der aktuell gesetzen Werte in den Filtern, von denen man abhängig ist. In die dependsOnQuery können die Werte aus den anderen Filtern über ihren Namen eingesetzt werden, indem man den Namen des Filters in geschweifte Klammern {...} schreibt.

Das gleiche gilt für das Attribut `dependsOnDefaultQuery`, das zur Ermittlung des Default-Wertes inklusive der aktuell gesetzen Werte in den Filtern dient.

Beispiele dynamischer MultipleChoice-Filter mit Abhängigkeiten:

```
<Table entity="Lagerplatz">
  <Query type="Text">
    <filter name="Halle" type="multipleChoice" title="$R{Halle}">
      <choiceQuery query="Halle bo WHERE Not Ldel ORDER BY Name">Regal.Halle.Id =
{Id}</choiceQuery>
    </filter>
    <filter type="multipleChoice" title="$R{Regal}" dependsOn="Halle">
      <choiceQuery query="Regal bo WHERE Not Ldel ORDER BY Nummer"
dependsOnQuery="Regal bo WHERE Not Ldel AND Halle.Id = {Halle} ORDER BY Nummer">Regal
= {Id}</choiceQuery>
    </filter>
```

```
<Query type="Text">
  <filter name="Maschine" type="multipleChoice" title="$R{Maschine}">
    <choiceQuery query="Maschine a where not Ldel order by Name">
      exists (within MaschinenPositionen p where p.Maschine.Id = {Id})
    </choiceQuery>
  </filter>
  <filter type="multipleChoice" title="$R{MaschinenFehlercode}" dependsOn="Maschine">
    <choiceQuery query="MaschinenFehlercode a where not Ldel
                        and (Inaktiv = null or not Inaktiv)
                        and MaschinenUnabhaengig
                        order by Name"
        dependsOnQuery="MaschinenFehlercode a where not Ldel
                        and (Inaktiv = null or not Inaktiv)
                        and (exists (within Maschinen m where m.Id={Maschine})
                             or MaschinenUnabhaengig)
                        order by Name">
      MaschinenFehler = {Id}
    </choiceQuery>
  </filter>
```

Momentan wird für die gesetzten Werte in Multiple-Choice-Filtern, von denen man abhängig ist, nur die Id und nicht das BO selbst eingesetzt. Dies wird sich noch ändern.

Die Notation ist noch verläufig und kann sich nochmals ändern. Insbesondere fehlt die Möglichkeit, für gesetzte NULL-Werte in anderen Filtern abweichende Klauseln angeben zu können, was oftmals jedoch nötig ist. Oftmals hilft eine Konstruktion, bei der man in der dependsOnQuery, die den Wert eines anderen Filters benutzt, zusätzlich eine Klausel "or '{Maschine}' = 'NULL' hinzufügt.

Dynamische Multiple-Choice-Filter mit choiceScript

Beispiel dynamischer MultipleChoice-Filter mit Skript:

```
<Query>
  <!-- Liste soll nur Kunden zur Auswahl enthalten, von denen es auch eine Rechnung
gibt -->
  <filter type="multipleChoice" title="Kunde">
    <clause>Kunde.AbstraktePerson.Name1 = "{}"</clause>
    <choiceScript language="groovy">
def erg = new TreeSet()
_bol.queryBO("SELECT a.Kunde.AbstraktePerson.Name1 FROM Rechnung a WHERE NOT Ldel
ORDER BY Kunde.AbstraktePerson.Name1").each{
  erg.add(it)
}
return new ArrayList(erg)
    </choiceScript>
  </filter>
</Query>
```

Weiteres Beispiel dynamischer MultipleChoice-Filter mit Skript:

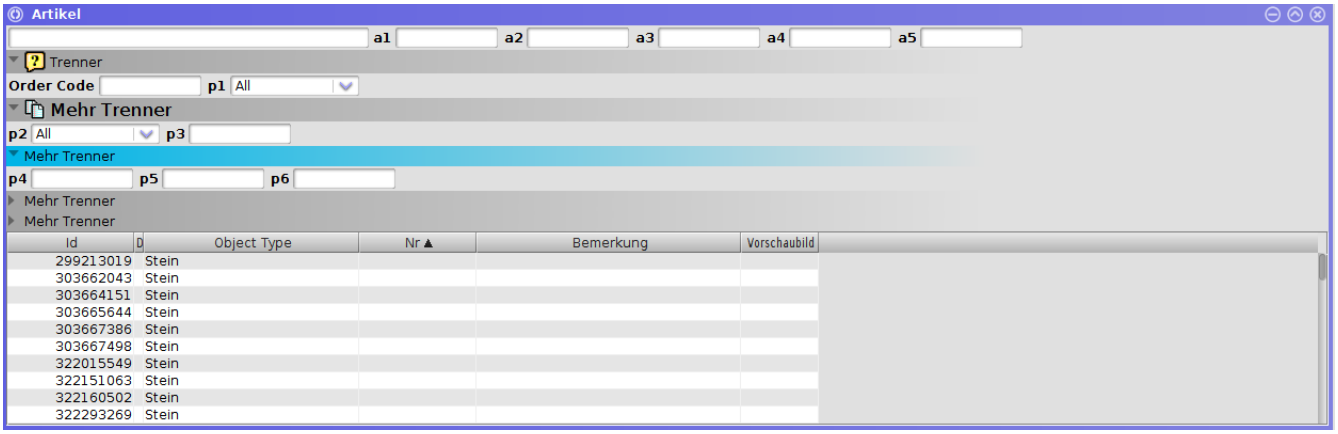
```
<Query>
  <!-- Liste enthaelt immer die letzten 10 Jahre -->
  <filter type="multipleChoice" title="Jahr">
    <clause>date_part("year",$IP{attrDatum})={}</clause>
    <choiceScript language="groovy">
def cal = Calendar.getInstance()
cal.setTime(new Date())
def year = cal.get(Calendar.YEAR)
def list = []
(0..9).each{ list.add(String.valueOf(year - it)) }
return list
    </choiceScript>
  </filter>
</Query>
```

Trenner

Bei Trennern handelt es sich um ein GUI-Element um verschiedene interaktive Filter optisch zu gruppieren und voneinander abzugrenzen. Trenner dienen lediglich dem Layout und haben auf die Abfrage keinen Einfluss.

Trenner werden mit dem `seperator`-XML-Element definiert und können wie ein `Label` (siehe Sektion zu Formularen) konfiguriert werden. Die Standard-Konfiguration vergrößert die Schrift um 10% und hinterlegt das Label mit einem grauen Farbverlauf.

Alle nach einem `seperator`-XML-Element definierten Filter werden optisch zusammengefasst; durch einen Klick auf den Trenner können alle zugehörigen Filter-Komponenten dann nach Wunsch zusammen aus- und eingeblendet werden.




```
<separator text="Artikeleigenschaften" icon="/20x20/Box.gif"/>
<filter...
...
<separator text="Verkauf2" collapsed="true"/>
<filter...
...
<separator prefSize="200c"
           text="Verkauf"
           fontSize="+10%"
           gradientStartColor="160 160 255"
           gradientStopPosition="SOUTH"/>
```

OQL-Klauseln

Hierbei handelt es sich um eine Funktionalität für fortgeschrittene Benutzer, die über die internen Datenstrukturen der Objekte und die Möglichkeiten von OQL Bescheid wissen.

Neben einfachen Suchbegriffen können in der Eingabezeile auch direkt OQL-Klauseln eingegeben werden, welche dann in die letztendlich auf der Datenbank ausgeführte OQL-Abfrage integriert werden.

Solche Suchanfragen werden mit `[` (einer offenen eckigen Klammer) eingeleitet.

Beispiele

Wo: In einem Lesezeichen für "Personen"
Fragestellung: Personen, die nach einem bestimmten Datum (und Uhrzeit) geboren sind

```
[Geburtsdatum >= "1999-12-24 08:00"
```

Wo: In einem Lesezeichen für "Kunden"
Fragestellung: Kunden, die im Land "Deutschland" residieren

```
[Land.Name = "Deutschland"
```

Wo: In einem Lesezeichen für "Kunden"
Fragestellung: Kunden, die in "Deutschland" oder "Luxemburg" residieren

```
[Land.Name = "Deutschland" OR Land.Name = "Luxemburg"

Alternativ

[Land.Name In List ("Deutschland", "Luxemburg")
```

Wo: Lesezeichen für "Länder"
Fragestellung: Dopplersuche nach Ländern mit gleichem Namen

```
[exists(Land b where not b.Ldel and b != a and b.Name = a.Name)
```

Wo: Lesezeichen für "Länder"
Fragestellung: Dopplersuche nach Ländern mit gleichem Namen, "Original" (was zuerst angelegt wurde) nicht anzeigen

```
[exists(Land b where not b.Ldel and b != a and b.Name = a.Name AND b.Crea < a.Crea)
```

Volltextsuche auf zusätzliche Felder ausdehnen

Hierbei handelt es sich um eine Funktionalität für fortgeschrittene Benutzer, die über die internen Datenstrukturen der Objekte Bescheid wissen.

Die Volltextsuche kann erweitert werden, so dass sie nicht nur die direkten Felder (Attribute) der Objekte durchsucht, sondern auch Felder von weiteren Objekten, die wiederum direkt mit den Objekten aus dem Lesezeichen verknüpft sind.

Hierzu muss in der XML-Definition des Lesezeichens das `<Query>`-Element erweitert werden. Mit Hilfe des Elements `<addProperty>` kann ein weiteres, "indirektes" Attribut in die Suche einbezogen werden.

Beispiel, das zusätzlich die Suche nach dem Namen des Objekttyps ermöglicht:

```
<Query type="Text">
  [...]
  <addProperty>Bot.Name</addProperty>
  [...]
</Query>
```

Fest eingestellte Filter

Manche Lesezeichen haben bereits vordefinierte, fest eingestellte Filterbedingungen die zusätzlich zu allen manuell eingegebene Filterbedingungen *immer* greifen. Diese sind immer aktiv und können nur durch Bearbeiten der XML-Definition des Lesezeichens deaktiviert werden.

Diese Filter werden definiert in `<filter>`-Kindelementen (ohne das Attribut `type`) des `<Query>`-Elements. Ähnlich wie bei der Suche mit OQL-Klauseln werden auch hier zusätzliche OQL-Schnipsel definiert, welche dann in die letztendlich auf der Datenbank ausgeführte OQL-Abfrage integriert werden.

Beispiel: Zeige auf jeden Fall nur Einträge, die beim Öffnen des Lesezeichens nicht älter als einen Monat sind:

```
<Query type="Text">
  [...]
  <filter>
    <![CDATA[  age(Crea) < "1 month"  ]]> ①
  </filter>
  [...]
</Query>
```

① Weil im Filter ein Kleiner-Zeichen `<` vorkommt, steht er in einer CDATA-Sektion. Alternativ könnte man auch `<filter>age(Crea) &lt; "1 month"</filter>` schreiben. Dies ist lediglich aufgrund der allgemeinen Erfordernisse von XML notwendig und hat nichts mit dieser Filterdefinition speziell zu tun.

Diese Filterdefinition unterstützt nur ein einziges (optionales) XML-Attribut zur Konfiguration: Wie bei anderen Filtern auch kann der Filter mit dem `group`-Attribut einer Bedingungsgruppe zugeordnet werden.

Als Alternative (ebenfalls optional) können, wie z.B. bei den "string"-Filtern, stattdessen auch mehrere `clause`-XML-Kindelemente benutzt werden, falls der Filter in unterschiedlichen Bedingungsgruppen verwendet werden soll.

Falls eine eigene Query-Schablone verwendet wird, könnten statt eines solchen Filters natürlich die entsprechenden OQL-Klauseln auch direkt in der Schablone angegeben werden. Durch Benutzung des `filter`-Elements wird die Schablone aber übersichtlicher gehalten und das Bearbeiten der Filterdefinition ist u.U. etwas einfacher.

Eigene Query-Schablone

Hierbei handelt es sich um eine Funktionalität für fortgeschrittene Benutzer, die über die internen Datenstrukturen der Objekte und die Möglichkeiten von MEX und OQL Bescheid wissen.

Im Normalfall verwenden die Lesezeichen zum Abfragen der Objekte eine Standard-OQL-Query der Form `SELECT a FROM <Typ> WHERE <Constraints>`.

Es ist jedoch auch möglich, komplexere Query-Formen zu definieren, z.B. um die erweiterten MEX-Möglichkeiten von Subqueries zu nutzen und z.B. die Objekte von zwei verschiedenen Untertypen mit leich veränderten Klauseln abzufragen.

Hierzu kann mit dem `template`-Kindelement eine eigene MEX-Query-Schablone definiert werden.

Beispiel: Von SubtypA nur Objekte mit Name "EinName" anzeigen aber von SubtypB nur solche bei denen in der Beschreibung "entfernt" vorkommt

```
<Query type="Text">
  [...]
  <template>
    {=select} SubtypA {=where} {=constraints} AND Name = "EinName"
    {Union {=select} SubtypB {=where} {=constraints} AND Beschreibung ilike
"%entfernt%"
  </template>
  [...]
</Query>
```

Bedingungsgruppen ("constraint groups")

Lesezeichen (bzw. eigentlich generell die MyTISM-eigene Tabellenkomponente) bieten die Möglichkeit, die durch Filter etc. definierten Bedingungen unterschiedlichen Bedingungsgruppen zuzuordnen. Dies ist insb. bei Benutzung des "Union"- bzw. "UnionAll"-MEX-Konstrukts nützlich, bei dem im Prinzip zwei oder gar mehr einzelne OQL-Queries zu einem gemeinsamen OQL-Query zusammengefügt werden. Bedingungsgruppen ermöglichen hier, einzelne Bedingungen nur in einem bzw. nur in einer Untermenge dieser Queries anzuwenden, in den anderen nicht.

Bedingungsgruppen werden einfach dadurch definiert, dass sie in einem Filter angegeben werden und dann bei Bedarf automatisch angelegt.

Beispiel: Union-Query der - neben gemeinsamen Bedingungen ("{=constraints}") die in beiden Teilqueries gelten - unterschiedliche Bedingungen ("{=constraintsForA}" bzw. "{=constraintsForB}") in den beiden Teilqueries benutzt

```
<Query type="Text">
  [...]
  <template>
    {=select} SubtypA {=where} {=constraints} AND {=constraintsForA}
    {Union {=select} SubtypB {=where} {=constraints} AND {=constraintsForB}
  </template>
  [...]
</Query>
```

Die Zuordnung von Bedingungen, z.B. aus Filtern, zu einer Gruppe erfolgt mit dem `group`-Attribut, das an entsprechenden Stellen angegeben wird.

Bei Filtern besteht die Möglichkeit, entweder den gesamten Filter selbst einer Bedingungsgruppe zuzuordnen oder aber das Filterkriterium in unterschiedlicher Weise in verschiedenen Bedingungsgruppen zu verwenden.

Um den ganzen Filter einer Gruppe zuzuordnen wird das `group`-Attribut im `filter`-Element direkt angegeben.

Beispiel in dem die im Filter "Dokumentnummer" eingetragene Bedingung nur für die Bedingungsgruppe "constraintsForA" benutzt wird

```
<Query type="Text">
  <filter type="string" title="Dokumentnummer" cols="30" group="ForA">
    <clause>Dokumentnummer = "{}"</clause>
  </filter>
</Query>
```

Wenn die vorher schon genannte Query-Schablone mit "Union" benutzt wird, wird der in diesem Filter eingegeben Wert nur bei der Abfrage für Objekte vom SubtypA berücksichtigt, da nur diese (Teil-)Abfrage die Bedingungsgruppe "constraintsForA" benutzt; die Abfrage für Objekte vom SubtypB filtert nicht nach diesem Kriterium.

Es ist auch möglich, das Filterkriterium in mehreren Bedingungsgruppen und Abfragen zu verwenden, aber z.B. in abgewandelter Weise. Hierzu müssen mehrere `clause`-Kindelemente definiert werden; für jedes dieser `clause`-Kindelemente wird dann bestimmt, in welcher Gruppe es verwendet werden soll.

Beispiel in dem die im Filter "Dokumentnummer" eingetragene Bedingung für beide Bedingungsgruppen benutzt wird

```
<Query type="Text">
  <filter type="string" title="Dokumentnummer" cols="30">
    <clause group="ForA">Dokumentnummer = "{}"</clause>
    <clause group="ForB">DokumentHeaderinfo.Nummer = "{}"</clause>
  </filter>
</Query>
```

Das obige Beispiel geht davon aus, dass die Dokumentnummer in SubtypA im Attribut `Dokumentnummer` zu finden ist, für den SubtypB dagegen in dem Attribut `Nummer` eines aus dem Dokument referenzierten DokumentHeaderinfo-Objekts. Bei der Suche nach passenden Objekten müssen also die Abfragen leicht unterschiedlich formuliert werden; dies ist mit diesem Verfahren möglich.

In der (Teil-)Abfrage für SubtypA in der Query-Schablone wird für diesen Filter die Klausel `Dokumentnummer = "{}"` in die Bedingungsgruppe "constraintsForA" eingebaut; in der (Teil-)Abfrage für SubtypB wird dagegen die Klausel `DokumentHeaderinfo.Nummer = "{}"` in der Bedingungsgruppe "constraintsForB" benutzt.

Falls für eine in der Query-Schablone benutzte Gruppe kein entsprechender `clause`-Eintrag dieser Gruppe zugeordnet wurde, wird für diese Gruppe auch keine Klausel in die Abfrage eingefügt. Die Bedingungsgruppen können allerdings trotzdem *immer* in der Form "... AND {=constraints<Gruppenname>}" in der Schablone verwendet werden; sollten für eine Bedingungsgruppe keine Bedingungen definiert worden sein - weil kein Filter dieser Gruppe zugeordnet wurde oder keiner der zugeordneten Filter vom Benutzer genutzt wurde - so wird automatisch eine Dummy-Klausel "(1 = 1)" eingefügt, die in diesem Fall syntaktisch korrektes OQL garantiert aber keine wirklichen Auswirkungen auf die Abfrage hat.

Massenänderungen / Skripting

Aus jeder Tabellenansicht (also Lesezeichen und Table-Popups bzw. Anzeigen in Formulare mit 1:n-Beziehung) heraus kann man sehr einfach Massenänderungen durchführen, d.h. eine oder mehrere Eigenschaften mehrerer BOs auf einmal ändern.

Hierfür markiert man in der Tabellenansicht die zu ändernden Datensätze (oops, sorry, Objekte) und ruft mit der rechten Maustaste das Kontextmenü auf.

Man hat nun die Möglichkeit, die Änderung mit dem Formular oder per Skript durchzuführen. Gibt man in einem oder mehreren Feldern des Formulars einen Wert bzw. Werte ein, werden diese beim Speichern auf alle markierten Objekte angewendet. So werden z.B. auch bei hinzugefügten und neu angelegten Objekten diese kopiert und an jedes markierte Objekt angehangen.

Beispiel: an mehrere Rechnungen soll ein Artikel als Rechnungsposten angehangen werden. Die betroffenen Rechnungen werden markiert und mittels des Massenänderungsformulars wird der besagte Artikel als Rechnungsposten angehangen. Nach abgeschlossener Massenänderung findet sich dieser Rechnungsposten als jeweils eigener Datensatz (Objekt) an allen markierten Rechnungen.

Mit dem Skript eröffnen sich per BeanShell-Programmierung weitaus grössere und komplexere Möglichkeiten. Neben den BeanShell-Befehlen stehen noch Funktionen aus den automatisch generierten Klassen zur Verfügung (zu finden in .PROJEKT/classes/de/PROJEKT/bo/).

Beispiel (um Projekteinträge an ein anderes Projekt zu hängen, aus der OAshi-Applikation "OAshi.Venice")

```
// Bitte modifizieren Sie dieses vorgefertigte Script nach Ihren Wuenschen
// bo.Id = (Long) ;
// bo.Crea = (Datetime) ;
// bo.Lmod = (Datetime) ;
// bo.Ldel = (Boolean) ;
// bo.Bot = // (BOT) ;
// bo.addDateien((Datei));
// bo.removeDateien((Datei));
// bo.Tid = (String) ;
// bo.Mitarbeiter = // (Mitarbeiter) ;
// bo.Datum = (Datetime) ;
// bo.Dauer = (Integer) ;
// bo.Kunde = // (Kunde) ;
prjs = ctx.queryBO("select bo from de.m.bo.Projekt bo where bo.Kuerzel = \"tapla\"");
bo.Projekt = prjs.get(0);
// bo.Beschreibung = (String) ;
// bo.BemerkungIntern = (String) ;
// bo.Kostenstelle = // (Kostenstelle) ;
// bo.InRechnungStellen = (Boolean) ;
```

Wenn man Objekte in Relation bringt oder aus der Relation entfernt mittels add/remove, so werden diese Änderungen "blind" geloggt. D.h. es wird nicht vorher überprüft, ob das BO, was man in Relation bringen will, bereits Teil der Relation ist, bzw. ob das BO, was man aus der Relation entfernen will, gar nicht Teil der Relation ist.

Dies birgt zwei Probleme:

1. Die Historie / das Log des BO weist "falsche" Einträge auf und erweckt ggfs. den Eindruck, dass der alte Zustand vor der Änderung ein anderer gewesen sei. Man muss also genau hinschauen, ob der suggerierte Zustand vor der Massenänderung wirklich so vorgelegen hatte, oder ob da effektiv "nichts" hinzugefügt oder entfernt wurde.
2. Bei einem undo einer solchen Massenänderung wird ein falscher Zustand hergestellt, der nicht dem Zustand vor der Massenänderung entspricht, da ggfs. ein Objekt in Relation gebracht wird, das gar nicht entfernt worden war, da es vorher gar nicht in Relation stand.

Das gleiche Problem besteht natürlich bei allen programmatisch vorgenommenen add/remove Änderungen an BOs.

Eine Lösung des Problems kann momentan nicht ohne erhebliche Performance-Einbußen wegen Unlazying programmiert werden, wird aber im Zuge des Projekts berücksichtigt werden, in dem wir `Castor` durch ein neues, besseres, selbst entwickeltes JDO-Data-Binding-Framework ersetzen werden.

"Transform Scripts" für die Abfrageresultate

Es ist mögliche Skripte zu definieren, die während des Ladens auf bestimmte oder alle zurückgelieferten Resultate (Objekte) der Abfrage angewendet werden.

In diesen Skripten können dann z.B. virtuelle Attribute gesetzt werden, deren Wert dann wiederum in Filtern benutzt werden kann. FIXME really?

Skripte werden mit `transform-script`-Kindelementen des `Query`-Elements definiert. Es können beliebig viele dieser Skripte definiert werden.

Da das Skript für ggf. sehr viele oder gar alle Einträge ausgeführt wird, kann das die Performance beeinträchtigen. Lange, aufwändige Berechnungen oder Ähnliches sollten in solchen Skripten also keinesfalls durchgeführt werden.

Folgende Variablen sind in diesen Skripten vorbelegt:

bo

Das aktuell betrachtete Objekt aus der Ergebnismenge, für welches das Skript gerade ausgeführt wird.

tag

Der sog. Tag, der für das aktuelle Resultat (Objekt) definiert wurde. Kann - nur? - mittels des `{Union @MeinTag }`-MEX-Query-Konstrukts definiert werden; alle Resultate, die dann von diesem Teil des Queries zurückgeliefert werden, bekommen den nach `@` genannten Tag zugewiesen.

Beispiel in dem sich jedes Objekt im Hilfsattribut "Badge" merkt, aus welchem Union-Query es zurückgeliefert wurde

```
<Query type="Text">
  [...]
```

```
  <template>
       only BO a where 'dummy'!='for Badge'
{Union @Eingeloest a.AusgezahltIn        from Bonuskarte {=where} {=constraints}}
{Union @Angespart  a.GeschaeftsVorfaelle from Bonuskarte {=where} {=constraints}}
  </template>
```

```
  <transform-script>
    bo.Badge = tag
  </transform-script>
  [...]
</Query>
```

Folgende XML-Attribute können für das `transform-script`-Element angegeben werden:

language

Optional - Welche Programmiersprache für das Skript genutzt werden soll; Standard und bisher eigentlich einzig unterstützt wird `groovy`.

onTag

Optional - Für Resultate mit welchem Tag (s.o.) das Skript ausgeführt werden soll; Standard ist `*` was das Skript für alle Resultate, unabhängig von einem evtl. gesetzten Tag ausführt.

Sonstiges der Lesezeichen-XML-Definition

Das Query-Element

Die meisten möglichen Angaben zur Konfiguration wurden bereits weiter oben beschrieben; hier noch kurze Erklärungen zu den möglichen aber noch nicht erwähnten Optionen.

Folgende XML-Attribute können im `Query`-Element angegeben werden:

entity

FIXME ???

fieldWidth

Optional - Die bevorzugte Breite des Text-Suchfeldes in Zeichen.

minSearchLength

FIXME ???

projection

FIXME ???

showDeleted

Optional - Hier kann mit einem Boolean-Wert "true" oder "false" (der Standardwert) angegeben werden, ob auch als gelöscht markierte Objekte im Lesezeichen angezeigt werden sollen oder nicht.

showFtsPopup

Optional, obsolet - Definiert, dass im Suchfeld Suchvorschläge für die veraltete "Compass"-Volltextsuche gezeigt werden sollen. Wird nicht mehr weiter unterstützt und fällt irgendwann komplett weg.

type

Unterstützt werden zwei Formen von Queries:

- `Text`: Die Standardform FIXME Alternativ erreichbar indem, statt des `Query`-Elements das `TextQuery`-Element genutzt wird.
- `Free` oder `Raw`: Hier kann bzw. muss direkt ein vollständiger OQL-Query eingegeben werden. Alternativ erreichbar indem, statt des `Query`-Elements das `FreeQuery`-Element genutzt wird.

Folgende Kindelemente kann das `Query`-Element haben:

transform-script

Optional, kann mehrfach verwendet werden - Siehe Abschnitt über "Transform Scripts"

addProperty

Siehe Abschnitt über zusätzliche Felder für die Suche

filter

Siehe Abschnitt über Filter

template

Siehe Abschnitt über eigene Query-Schablone

separator

Siehe Abschnitt über Trenner

Formulare

Formulare sind Eingabemasken, mit deren Hilfe BOs erstellt oder bearbeitet werden können. Sie definieren welche (Eingabefelder für welche) Attribute angezeigt werden.

Eingabemöglichkeiten nach Datentypen

(***FIXME Diese Sektion passt eigentlich nicht wirklich hier hin; sollte man später mal alles sauber anordnen ...***)

Timespan (Zeitspanne)

FIXME Standardmässig wird für die Eingabe im Solstice jetzt der SimpleTimespanChooser verwendet, der eine einfachere Eingabe als hier angegeben erlaubt.

Zeitspannen werden intern als Anzahl von Sekunden abgespeichert. Eingegeben werden können jedoch intuitivere Werte wie z.B. eine Anzahl von Minuten, Stunden, Tage etc. Es gibt dafür grob drei Gruppen von Formaten:

Altes Standardformat

Dieses Format wird verwendet wenn kein spezielles displayFormat angegeben ist.

Beispiele:

- `30s` = Dreissig Sekunden
- `10m` = Zehn Minuten
- `1d 2h` = Ein Tag und zwei Stunden
- `3w` = Drei Wochen
- `5y 3M` = Fünf Jahre und drei Monate

Folgende Bezeichner können dabei verwendet werden:

Table 1. Eingabe Timespan

Bezeichner	Name	Entspricht
y	Jahr (year)	365d
M	Monat (month)	30d
w	Woche (week)	7d
d	Tag (day)	24h = 1440m = 86400s
h	Stunde (hour)	60m = 3600s
m	Minute (minute)	60s
s	Sekunde (second)	1s

Achten Sie darauf, dass sie bei Benutzung mehrere Bezeichner immer mit den grössten anfangen.

Beispiele:

- Richtig: `1m 30s`
- Falsch: `30s 1m`
- Richtig: `1d 5h 20m`
- Falsch: `1d 20m 5h`

Achten Sie auch darauf, dass zwischen Zahl und Bezeichner keine Leerzeichen stehen dürfen und dass der Bezeichner immer *nach* der Zahl kommen muss.

Beispiele:

- Richtig: `1m 30s`
- Falsch: `1 m 30 s`
- Falsch: `m1 s30`
- Falsch: `1 30s`
- Falsch: `1x 30s`
- Falsch: `a1 30s`
- Falsch: `m 1 30s`

Alle eingegebenen Zeitspannen werden automatisch in ein kanonisches, d.h. festgelegtes, eineindeutiges Format umgewandelt.

Beispiele:

- `55s` bleibt `55s`
- `73s` wird zu `1m 13s`
- `30h` wird zu `1d 6h`
- `10d` wird zu `1w 3d`
- `200w` wird zu `3y 10M 5d`
- `70m 340s` wird zu `1h 15m 40s`
- `70M 340s` wird zu `5y 9M 5d 5m 40s`
- `13y 6M 45d` wird zu `13y 7M 2w 1d`

"Doppelpunkt"-Format(e)

Dieses Format wird verwendet wenn als displayFormat "HH:mm:ss" bzw. "HH:mm" angegeben ist. Die Stundenanzahl hat dabei immer mindestens zwei Ziffern, bei Bedarf können aber auch mehr dargestellt/verwendet werden.

Beispiele für "HH:mm:ss":

- `00:00:30` = Dreissig Sekunden
- `00:10:00` = Zehn Minuten

- `26:00:00` = Ein Tag und zwei Stunden
- `504:00:00` = Drei Wochen
- `45960:00:00` = Fünf Jahre und drei Monate

"Marker"-Format(e)

Bei diesen Formaten wird die Zeitspanne als nur eine Zahl dargestellt. Ein Marker-Buchstabe im displayFormat gibt dabei an, in welche Einheit die Zeitspanne umgerechnet bzw. angezeigt wird.

Beispiele für Darstellung bzw. akzeptierte Eingaben für eine Zeitspanne von 455984 Sekunden mit verschiedenen displayFormat-Alternativen:

- ###,##0.00s = `455,984.00`
- #####0s = `455984`
- ###,##0.00m = `7,599.73`
- #####0m = `7600`
- ###,##0.00h = `126.66`
- #####0h = `127`
- ###,##0.00d = `5.28`
- #####0d = `5`

Als Marker erlaubt sind, wie im Beispiel zu sehen, 's' für Sekunden, 'm' für Minuten, 'h' für Stunden, 'd' für Tage, 'w' für Wochen, 'M' für Monate (= 30 Tage) und 'y' für Jahre (= 365 Tage). Bei Aus- oder Eingabe werden diese Marker-Buchstaben nicht angezeigt bzw. eingegeben.

Als Besonderheit gibt es noch den Marker '*'. Bei Verwendung dieses Markers wird (bei der Ausgabe) automatisch die "beste" Einheit gewählt, d.h. diejenige, bei der eine Zahl >= 1.0 herauskommt. Als Spezialfall wird bei diesem Format der passende Marker-Buchstabe mit ausgegeben, bzw. muss bei der Eingabe ebenfalls an die Zahl angehägt werden, damit die korrekte Einheit gewählt werden kann.

Die Zeichen vor dem Marker-Buchstaben sind ein Pattern für java.text.DecimalFormat, welches für die Formatierung der Zahl verwendet wird.

Diverses

- Messagebox erzeugen: `ctx.showMessageDialog("bla")`
- Sperren von Formularfeldern: dem jeweiligen Feld mit name="ich" einen Namen geben und im Formular-Code dann: `ich.setEditable(false);`
- Der Parameter `lazy` wird in der Formular-Definition im Tab-Tag verwendet (Bsp.: `<Tab lazy="false"`) und gibt an, ob die Daten die im Formular hinter diesem Tab (Reiter) stecken, direkt beim Öffnen des Formulars geladen werden sollen (`lazy="false"`) oder erst wenn man den Tab anklickt (`lazy="true"` - das ist die Standard-Einstellung).
- Farbliches Aussehen der Reiter wird im jeweiligen Benutzer (Formular, Parameter, ganz unten)

eingestellt. Diese "Defaults" kommen aus Projekt/gui/Client.nrx (nach "xpath" suchen)

Schablonen

Wie in "Grundlagen" bereits beschrieben, dienen Schablonen dazu, neue BOs anzulegen. Eine Schablone definiert, von welcher Klasse ein neues Objekt erzeugt werden soll und welches Formular zur Darstellung und Bearbeitung benutzt werden soll. Möglicherweise werden auch bereits bestimmte Werte in das neu zu erzeugende BO geschrieben.

Die meisten Attribute des Formulars sind aus den anderen Strukturelementen bekannt und/oder selbsterklärend. Wichtige spezielle Attribute:

BOTyp

Von welcher Klasse soll ein Objekt erzeugt werden?

Formular

Welches Formular (passend zum BOTyp bitte) soll für die Bearbeitung des neuen Objekts/Eintragen der Werte benutzt werden?

Parameter

Hier kann (per XML) ein Script definiert werden, über das z.B. Werte im neuen Objekt bereits vorbelegt werden. Weitere Konfigurationsmöglichkeiten bzw. Angaben sind hier z.Zt. nicht möglich.

Erzeugen des neuen Objektes

vgl. de/ipcon/gui/solstice/Client.openNew()

Im Normalfall wird ein Objekt der angegebenen Klasse (BOTyp, s.o.) einfach durch Aufruf des entsprechenden No-Argument-Konstruktors erzeugt. Will man aber selber z.B. direkt Werte des neuen Objektes setzen, kann man die Objekterzeugung mittels Script selbst in die Hand nehmen. Dazu gibt man als Parameter für das Formular ein entsprechendes BeanShell-Script an, welches die gewünschten Aktionen durchführt. Das Script muss ein neu erstelltes Objekt der gewünschten Klasse zurückliefern.

Das Beispiel zeigt den Inhalts des Parameter-Feldes einer Schablone für MyTISMBenachrichtigungsAuftrag; wie man sieht können so auch andere Objekte direkt mit erzeugt und konfiguriert werden:

```
<Schablone>
  <newInstance>
    ba = tx.include(new MyTISMBenachrichtigungsAuftrag());
    ba.setAbsender(ctx.getCurrentUser());
    bv = tx.include(new MyTISMBenachrichtigungsVorlage());
    bv.setIstEinweg(true);
    ba.setVorlage(bv);
    return ba;
  </newInstance>
</Schablone>
```

Folgende Variablen sind im Script immer verfügbar (vgl. s.o. und de/ipcon/gui/BasicClient.initScript()); ggf. können aber auch noch weitere Variablen übergeben worden sein:

ctx

Der verwendete ClientContextI. ***FIXME gibt es den wirklich immer?***

ftx

Der verwendete FormContext.

tx

Die Transaction, die für die Erstellung des Objekts verwendet wird.

Die alte Methode der Definition von Default-Werten im Schema wird aus Kompatibilitätsgründen zwar noch unterstützt, sollte aber nicht mehr verwendet werden.

Reports

Grundlagen

Mittels `Reports` können Sie aus MyTISM heraus Listen oder Dokumente in verschiedenen Formaten (z.B. PDF) erzeugen. Reports nutzen die Daten von Objekten aus der MyTISM-Datenbank und stellen diese gemäß dem definierten Vorlage-Layout dar.

Was ist ein Report überhaupt?

"Reporting" ist ein Begriff für das Erzeugen von strukturierten Dokumenten oder Listen aus Daten einer Datenbank. Reports unterscheiden sich von einfachem Textfluss durch "Schaltpunkte" wie Seitengrenzen, Spaltenenden oder Gruppenwechsel, die den Aufbau steuern.

Traditionelle Reports:

Früher basierten Reports auf einer Matrix aus Spalten (Felder) und Zeilen (Datensätze). Durch Sortierung entstehen Gruppen.

- **Beispiel:** Eine Liste von Personen mit Spalten für Anrede, Familienname, Rufname etc.
- **Gruppierung:** Sortierung nach Anrede erzeugt Gruppen "Frau" und "Herr". Weitere Sortierung nach Familienname innerhalb der Anrede erzeugt Untergruppen (z.B. alle Herren mit Nachnamen "Müller").

Bänder und Gruppenwechsel:

Jede Gruppe kann Kopf- und Fußbänder erhalten, die um das Detailband (enthält die eigentlichen Daten) herum angeordnet werden. Gruppenwechsel lösen das Drucken der entsprechenden Fuß- und Kopfbänder aus.

- **Beispiel:** Ein Report mit Gruppierung nach Anrede. Jede Seite fasst maximal 3 Personen.

```
Report-Titel
   Seiten-Kopf "Seite 1"
      Gruppen-Kopf "Frau"        // Kopfband der Gruppe "Frau"
         Details <1>             // Detailband mit Daten der ersten Person
         Details ②
         Details ③
   Seiten-Fuß "Seite 1"
   Seiten-Kopf "Seite 2"
         Details ④
         Details ⑤
      Gruppen-Fuß "Frau"         // Fußband der Gruppe "Frau"
      Gruppen-Kopf "Mann"        // Kopfband der Gruppe "Mann"
         Details ⑥
   Seiten-Fuß "Seite 2"
   ...
```

Variationen:

- **Gruppenwechsel mit Seitenwechsel verbinden:** Jede neue Gruppe beginnt auf einer neuen Seite.
- **Kopfband auf jeder Seite drucken:** Der Gruppenkopf wird bei jedem Seitenwechsel wiederholt.

Komplexeres Beispiel:

- **Daten:** Schrauben mit Eigenschaften Material (M), Kopfart (K) und Durchmesser (D), sortiert nach M, K, D.
- **Gruppen:** M und K
- **Ausgabe:** Der Report zeigt, wie Kopf- und Fußbänder bei Gruppenwechseln reagieren.

```
Report-Titel
   Seiten-Kopf "Seite 1"
      Gruppen-Kopf "Blech"
         Gruppen-Kopf "Flach"
            Details ①
            Details ②
         Gruppen-Fuß "Flach"
         Gruppen-Kopf "Rund"
            Details ③
   Seiten-Fuß "Seite 1"
   Seiten-Kopf "Seite 2"
            Details ④
         Gruppen-Fuß "Rund"
         ...
```

MyTISM-Ansatz:

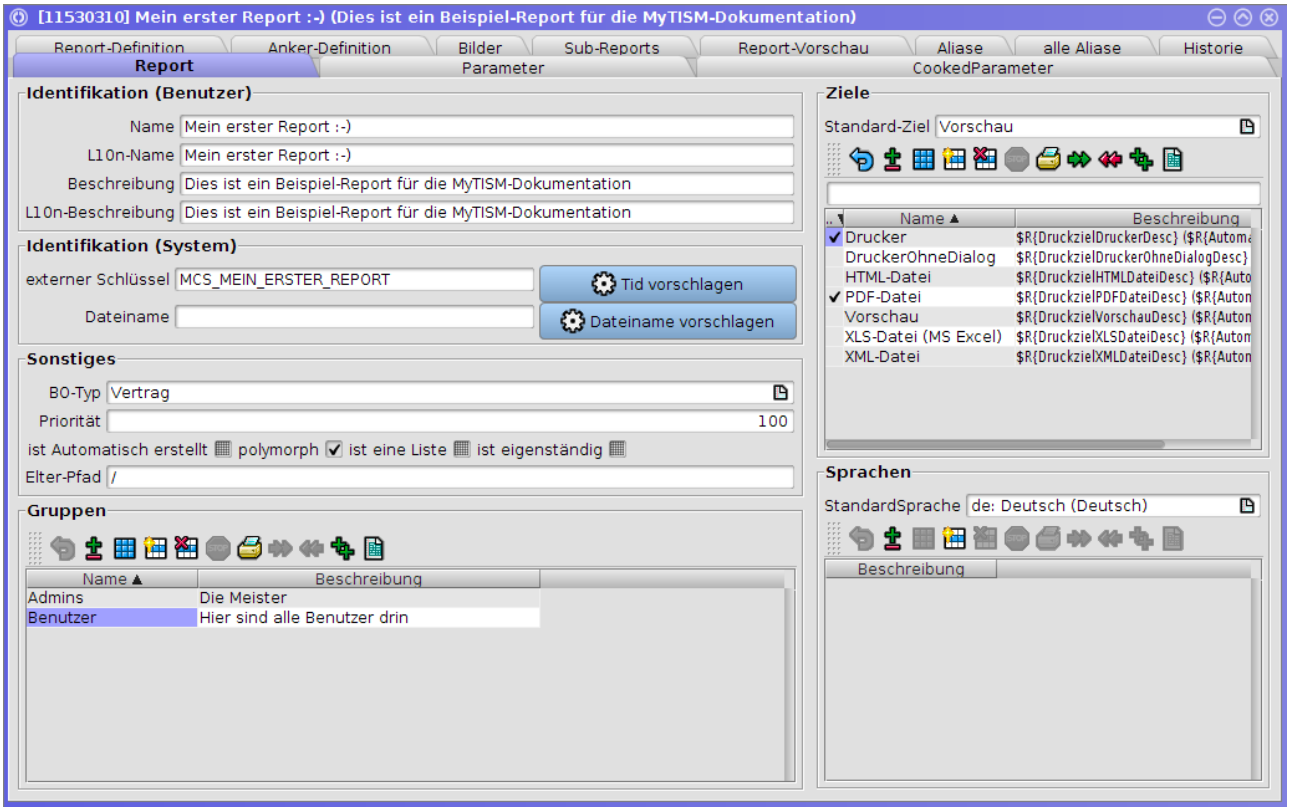
MyTISM verfolgt einen eigenen Ansatz für Reports, der auf der objektorientierten Struktur basiert und den direkten Zugriff auf Datenbankfelder und virtuelle Eigenschaften ermöglicht. Die Anker-Definition legt die Struktur fest, komplexe SQL-Queries entfallen.

Reports für MyTISM werden in XML geschrieben und können mit jedem Texteditor bearbeitet werden. Für eine bessere Übersicht und Vorschau empfiehlt sich jedoch ein grafischer Editor wie iReport.

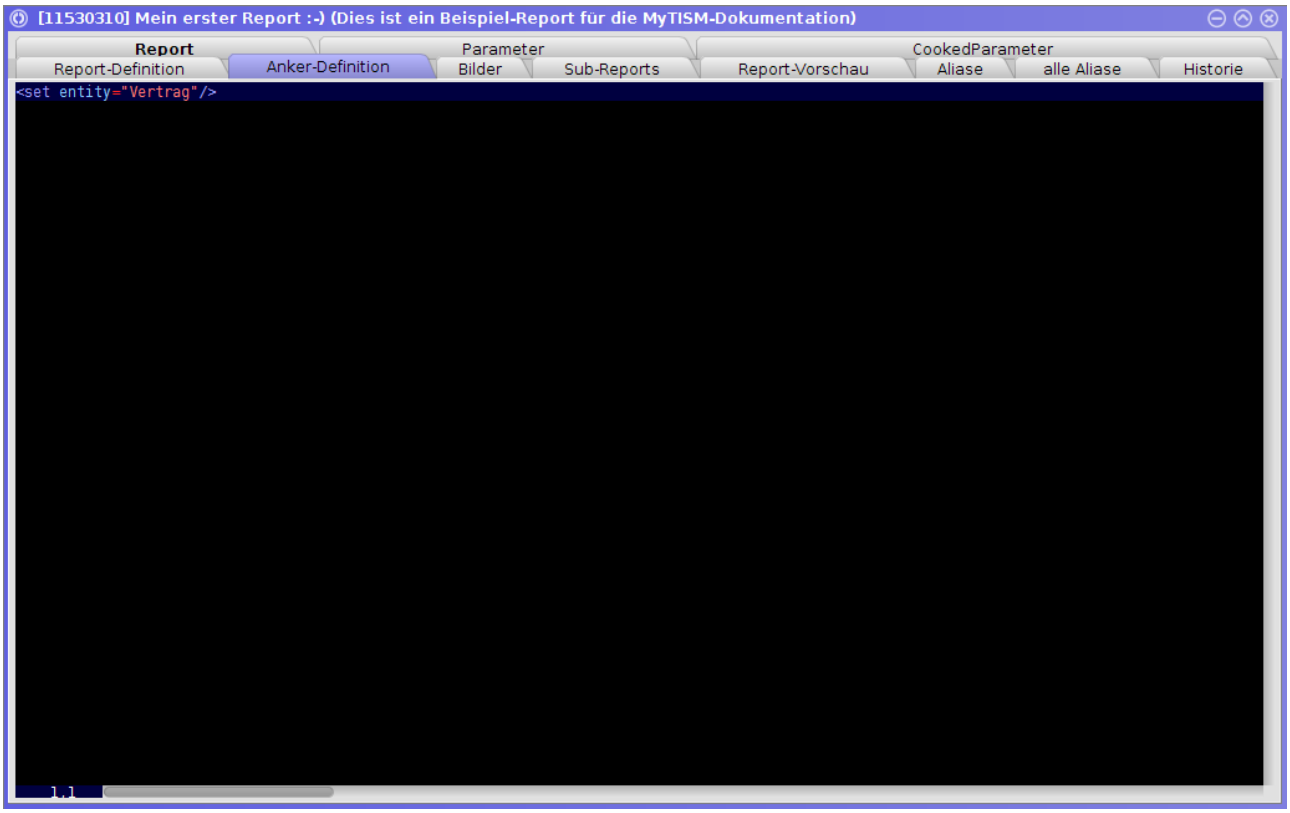
Erstellung eines neuen Reports

1. Erzeugen Sie ein neues Report-Objekt mittels der Schablone `/Admins/MyTISM (Vorgebaut)/Grundelemente/Report (Vorgebaut)`.
2. Vergeben Sie einen aussagekräftigen `Namen` und eine kurze `Beschreibung`.
3. Verwenden Sie den Knopf `Tid vorschlagen`, um einen Kurznamen/externen Schlüssel automatisch zu generieren, oder vergeben Sie einen manuell.

4. Wählen Sie den BO-Typ. Dieser definiert, welche Objekte als Datengrundlage für den Report dienen.
   Beispiel: Für einen Report zur Erzeugung von Vertragsdokumenten wählen Sie Vertrag.
5. Geben Sie die Priorität an. Diese bestimmt die Position des Reports in Listen (z.B. Kontextmenüs). Ein sinnvoller Wert hängt von anderen Reports für denselben BO-Typ ab (Vorschlag: 100).
6. Auch für Unterklassen des BO-Typs nutzbar: Ermöglicht die Verwendung des Reports auch für Objekte von Unterklassen des angegebenen BO-Typs.
   Beispiel: Ein Report für Vertrag ist mit dieser Option auch für Mietvertrag verfügbar.
7. Ist eine Liste: Gibt an, ob der Report mehrere Objekte auflistet (z.B. Übersicht aller Verträge) oder nur ein einzelnes Objekt darstellt (z.B. ein Vertragsdokument).
8. Ist eigenständig: Ermöglicht dem Report, die Datengrundlage selbstständig anhand einer definierten Abfrage zu ermitteln. Andernfalls kann der Report nur auf eine Objektauswahl (z.B. in einem Lesezeichen) angewendet werden.
9. Weisen Sie den Report bestimmten Gruppen zu. Nur Mitglieder dieser Gruppen können den Report verwenden (Vorschlag: Admins und Benutzer).
10. Wählen Sie mögliche (Druck-)Ziele und das voreingestellte Standard-Druckziel. Dies legt das Ausgabeformat des Reports fest (Vorschlag für Standard-Ziel: Vorschau).
11. Sprachen: Bestimmt, in welchen Sprachen der Report gerendert werden kann. Im Druckdialog wird die Auswahl auf diese Sprachen beschränkt. Ohne Auswahl wird die Standardsprache des Reports verwendet.
12. Speichern und schließen Sie das Report-Objekt.
13. Verschieben Sie den Report im Navigationsbaum in den Zielordner.
14. Öffnen Sie den Report erneut zur Bearbeitung.
15. Verwenden Sie den Knopf Dateiname vorschlagen, um einen Dateinamen zu generieren.



16. Wechseln Sie zum Reiter `Anker-Definition`.
17. Beispiel für die einfachste Version: `<set entity="Vertrag"/>` (ersetzen Sie "Vertrag" durch den gewünschten internen Namen des BO-Typs). Genauere Informationen finden Sie im Abschnitt zur Anker-Definition.



18. Speichern und schließen Sie den Report.
19. Exportieren Sie den Report mittels Struktursynchronisation in ein Verzeichnis.
20. Erstellen Sie die eigentliche Report-Layoutvorlage. Nutzen Sie dafür das externe Programm

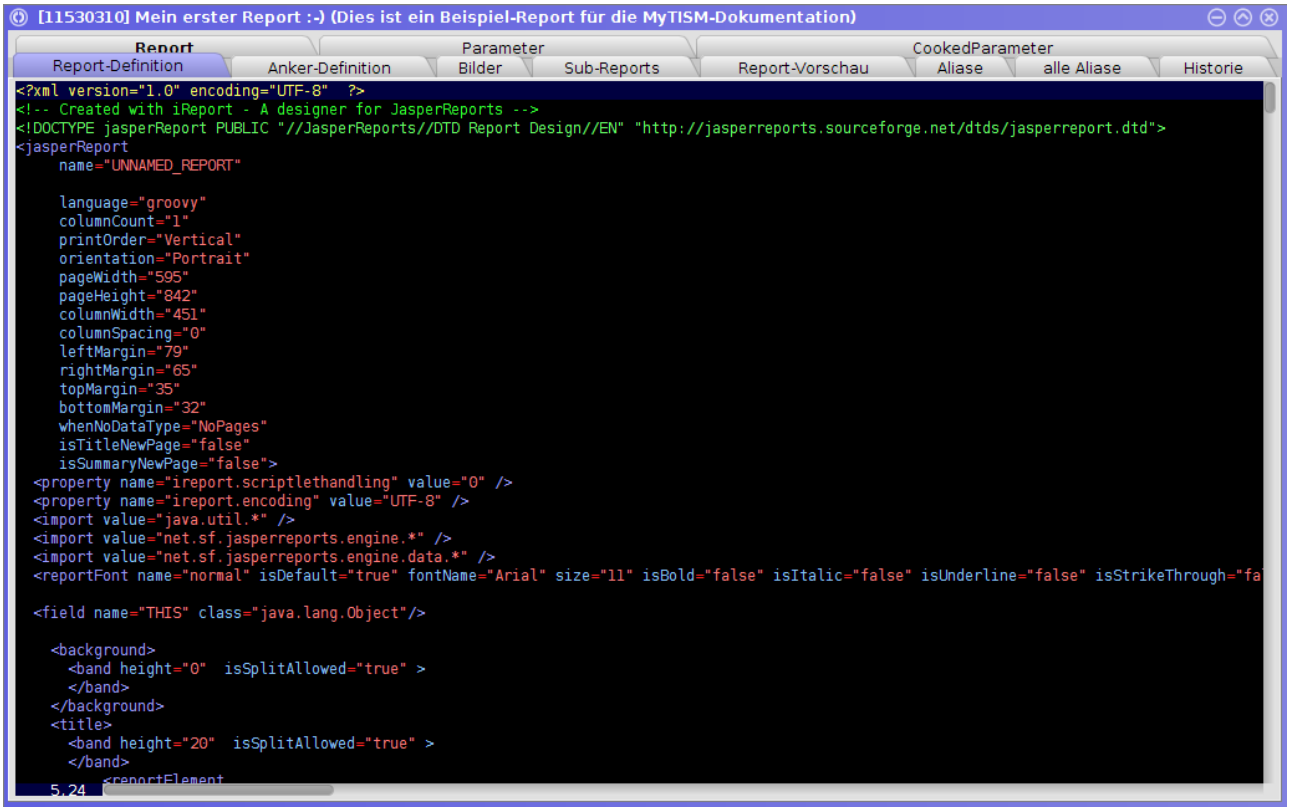
iReport. Falls noch nicht vorhanden, laden Sie es herunter und installieren Sie es.

Verwenden Sie zum Bearbeiten der Reports ausschließlich iReport 2.0.5 (die letzte Version, die eine Änderung der Kompatibilitätseinstellung erlaubt) und wählen Sie unter `Options Compatibility` "JasperReports 2.0.0 - 2.0.1". MyTISM nutzt eine angepasste Version der JasperReports-Bibliotheken, die neuere Report-Formate noch nicht unterstützt.

21. Starten Sie iReport und öffnen Sie die durch die Struktursynchronisation erstellte Vorlage (`REPORTNAME.xml`).

Die Struktursynchronisation erzeugt zwei Dateien: `REPORTNAME.rpt.xml` und `REPORTNAME.xml`. Die zweite Datei ist die in iReport ladbare Report-Layout-Definition.

22. Erstellen/bearbeiten Sie das Vorlage-Layout wie gewünscht.
    Ein Report hat sinnvolle Voreinstellungen, die angepasst werden können:
    - Seitenformat (vordefiniert oder benutzerdefiniert)
    - Seitenausrichtung (Hoch- oder Querformat)
    - Seitenränder
23. Speichern Sie die Report-Definition in iReport.
24. Importieren Sie den Report zurück in MyTISM mittels Struktursynchronisation und testen Sie ihn.
    Beispiel: Wählen Sie ein passendes Objekt (z.B. einen Vertrag) aus einem Lesezeichen und wählen Sie im Kontextmenü `drucken mit "Mein erster Report :-)"`. Lassen Sie das Fenster für die Struktursynchronisation geöffnet, um bei Änderungen der Layout-Vorlage den Import einfach erneut durchführen zu können.



(Eingabe-)Parameter für Reports

Parameter ermöglichen die Übergabe benutzerdefinierter, manuell eingegebener Werte an Reports. Diese müssen in der Reportdefinition definiert und konfiguriert werden. Beim Ausführen des Reports im Solstice-Client werden sie dann in einem Dialog abgefragt.

Beispiel:

```
[...]
<reportFont name="heading" isDefault="false" fontName="Arial" size="10" [...]/>
<parameter name="GruppierenNach" isForPrompting="true" class="java.lang.String">
  <property name="choiceScript"
            value="model.addEntry('$R{Produkt}');
                   model.addEntry('$R{Saison}', 'Ich will einen anderen Titel in der
Box :-)');
                   model.addEntry('$R{Kurzbezeichnung}');
                   model.addEntry('$R{Lieferant}');"/>
  <property name="chooseOnly" value="true"/>
</parameter>
<parameter name="Stichtag" isForPrompting="true" class="java.util.Date">
  <property name="format" value="MEDIUM_"/>
</parameter>
<field name="THIS" class="java.lang.Object"/>
[...]
```

- `format`: Wird von fast allen Parametern unterstützt und enthält ein CBOFormat zur Verarbeitung der Eingabewerte.
- `choiceScript`: Bei Angabe dieser Property wird eine Combobox mit den Werten aus dem Skript

angezeigt.
Weitere Konfiguration:

  - `chooseOnly` (true/false): Nutzer kann nur vorgegebene Werte auswählen (true) oder eigene eingeben (false).
  - `nullable` (true/false): Kein Wert muss ausgewählt werden (true).
  - Siehe JavaDoc der Klasse `de.ipcon.form.FComboBox` für weitere Informationen

- `rawInputDefinition`: *Fortgeschrittene Benutzer* können *in Spezialfällen* die Definition des Eingabeelements direkt angeben. Dies erfordert Kenntnisse der MyTISM-Formular-XML-Sprache und sorgfältiges Escapen von Sonderzeichen. Der Name der `property` muss dem Typ der Report-Parameter-Klasse entsprechen (z.B. `VString` für `java.lang.String`, `VBO` für Subtypen von `BO`).

Beispiel für ein Auswahl-Popup für einen Benutzer:

```
  <parameter name="EinBenutzer" isForPrompting="true"
class="de.ipcon.db.core.Benutzer">
    <property name="rawInputDefinition" value="&lt;Popup
property=&quot;VBO&quot;&gt;&lt;Table&gt;&lt;Query type=&quot;Text&quot;
entity=&quot;Benutzer&quot;&gt;&lt;filter&gt;NOT AnmeldungVerweigern OR
AnmeldungVerweigern = null&lt;/filter&gt;&lt;/Query&gt;&lt;Columns&gt;Name,
ASC|Beschreibung&lt;/Columns&gt;&lt;/Table&gt;&lt;/Popup&gt;"/>
  </parameter>
```

Die Anker-Definition oder: Wie komme ich an die Daten?

MyTISM verwendet einen eigenen Ansatz für den Reportgenerator, um unabhängig von Datenbankänderungen zu sein und virtuelle Eigenschaften nutzen zu können. Das Kernstück ist die Anker-Definition, die festlegt, an welchen Objekttypen ein Report verankert ist:

- Welche Entität haben die übergebenen Objekte?
- Welche Relationen werden aufgefaltet (ähnlich zu Joins in SQL)?
- Wie wird sortiert?

Die Anker-Definition ist ein XML-Schnipsel und definiert die Entität (z.B. `Rechnung`) sowie das Auffalten von Relationen mittels `many`-Tags:

```
<set entity="Rechnung">
  <many property="Posten" alias="P">
    <sort ascending="true" byProperty="Position">
  </many>
</set>
```

Dadurch werden automatisch Gruppen gebildet und die Daten sortiert. Alle n-1 Relationen und deren Attribute sind direkt zugreifbar, ohne weitere Definition. Alias-Namen können zur

Vereinfachung verwendet werden:

```
Nummer
Adressat.StandardKontakt.Anschrift.Strasse
P.Artikel.Listenpreis
P.Einzelpreis
P.Gesamtpreis
P.Position
```

Komplexe Szenarien können mit eingebetteten OQL-Queries, Script-Schnipseln oder virtuellen Properties abgebildet werden.

Um Eigenschaften in Ausdrücken innerhalb eines Reports zu verwenden, nutzen Sie Feld-Klammern `$F{}`:

```
$F{P}.Position
$F{Nummer}
```

Weitere Klammern sind `$P{}` für Parameter und `$V{}` für Variablen. Ausdrücke können kombiniert werden, z.B.:

```
"RG-Nr ${$F{Nummer}}"
"Seite ${$V{PAGE_NUMBER}}"
"${$F{Familienname}}, ${$F{Rufname}}"
L10n.formatDate(new Date(), "yyyy-MM-dd")
```

Für komplexe Formatierungen bietet sich das CBOFormat an.

virtualProperties in Reports

Definieren Sie virtuelle Eigenschaften wie folgt:

```
<set entity="StueckListe">
  <virtualProperty name="VorhandeneZusatzstoffeAlsString" entity="StueckListe">
    <get>de.ipcon.tools.TextTools.join(getVorhandeneZusatzstoffe().values())</get>
  </virtualProperty>
</set>
```

Achtung: Rufen Sie virtuelle Eigenschaften in `textFieldExpressions` **ohne** "get" auf:

Richtig:

```
<textFieldExpression
class="java.lang.String"><$F{THIS}.getVorhandeneZusatzstoffe().isEmpty() ? "-" :
$F{THIS}.VorhandeneZusatzstoffeAlsString</textFieldExpression>
```

Falsch:

```
<textFieldExpression
class="java.lang.String"><$F{THIS}.getVorhandeneZusatzstoffe().isEmpty() ? "-" :
$F{THIS}.getVorhandeneZusatzstoffeAlsString()</textFieldExpression>
```

Sonst erhalten Sie eine Fehlermeldung, dass die Methode nicht existiert.

Das CBOFormat und seine Verwendung im Report

Das CBOFormat ermöglicht eine elegante Verwendung von Objekteigenschaften innerhalb eines Reports. Es muss in Form eines Feldes verpackt werden:

```
$F{Objektname}.describe("Eigenschaft1")
$F{Objektname}.describe("Eigenschaft1(', 'Eigenschaft2)")
$F{Objektname}.describe("Eigenschaft1' 'Eigenschaft2")
```

Für den Zugriff auf das "Haupt-BO" eines Reports wird automatisch ein Feld namens "THIS" angelegt. Darüber können Sie per CBO-Format, GStrings oder Groovy-Auswertung auf Inhalte zugreifen.

Beispiel für Zugriff über CBO-Format

```
<field name="THIS" class="java.lang.Object"/>
[...]
<textFieldExpression
class="java.lang.String">$F{THIS}.describe("Familienname")</textFieldExpression>
[...]
<textFieldExpression
class="java.lang.String">$F{THIS}.describe("Rufname")</textFieldExpression>
[...]
<textFieldExpression
class="java.lang.String">$F{THIS}.describe("Titel")</textFieldExpression>
[...]
<textFieldExpression
class="java.lang.String">$F{THIS}.describe("Geburtstag")</textFieldExpression>
[...]
```

Der Zugriff via Groovy-Notation ermöglicht den Aufruf von Gettern oder anderen Methoden, die auch andere Werte als Strings zurückgeben können.

Beispiel für Zugriff via Groovy-Notation

```
<field name="THIS" class="java.lang.Object"/>
[...]
<textFieldExpression
class="java.lang.String">$F{THIS}.familienname</textFieldExpression>
[...]
<textFieldExpression class="java.lang.String">$F{THIS}.rufname</textFieldExpression>
[...]
<textFieldExpression
class="java.lang.String">$F{THIS}.titel?.name</textFieldExpression>
[...]
<textFieldExpression class="java.lang.String">$F{THIS}.getAlter(new
Date())</textFieldExpression>
[...]
```

Troubleshooting

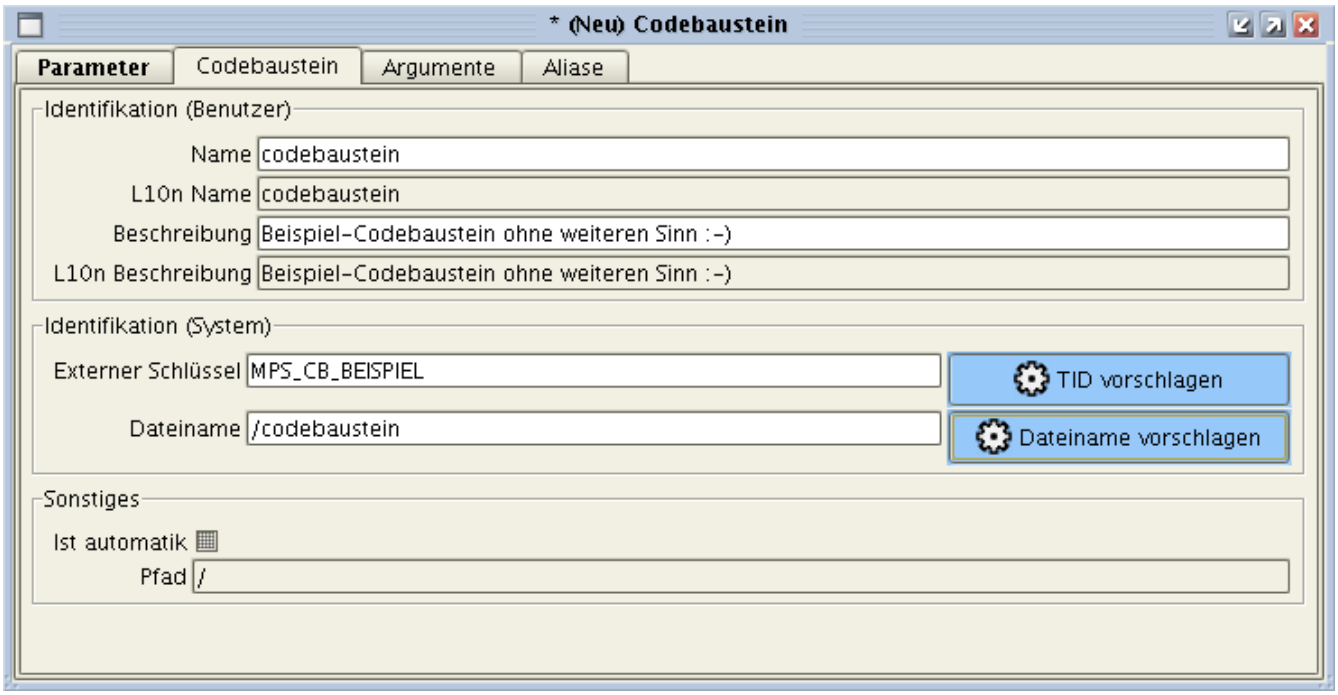
Seitenwechsel / Überlappende Felder / "wachsende" Felder bei dynamischem Text

- `isStretchWithOverflow="true"`: Passt die Größe von Textfeldern an den Inhalt an.
- `positionType="float"`: Verschiebt nachfolgende Felder automatisch nach unten.
- `isSplitAllowed="true"`: Ermöglicht den Umbruch von Bändern bei wachsendem Inhalt.
- `minHeightToStartNewPage`: Beeinflusst den Band-Umbruch.

Codebausteine

Codebausteine dienen dazu, Teile des XML-Quelltextes zu verwalten, die von verschiedenen Strukturelementen gemeinsam verwendet werden. Diese Codeteile können dann auf einfache Weise in den Quelltext von Strukturelementen eingebunden werden, ohne dass der Code immer wieder kopiert werden muss.

Einbinden von Codebausteinen



Codebausteine können einfach durch Einfügen eines Elementes `<Includename="codebausteinName/pfad"/>` im Quelltext (Attribut "Parameter" bzw. zusätzlich Attribute "AnkerDefinition" und "ReportDefinition" bei Reports) eines Strukturelementes eingebunden werden. Dabei gibt das Attribut `name` den Namen (ggf. mit Pfad) an, unter dem der Codebaustein im Navigationsbaum abgelegt ist.

Bitte beachten: Damit der Codebaustein richtig gefunden wird, müssen Sie sowohl für den Codebaustein als auch für die Ordner im ggf. angegebenen Pfad den Wert aus dem "Name"-Attribut des Codebausteins bzw. Ordners verwenden! Der im Baum angezeigte Name ist der sog. "L10nName", der automatisch (soweit verfügbar) in der für den Client angezeigten Sprache gehalten ist. Dieser "L10nName" wird sich in vielen Fällen vom eigentlichen Namen des Elements in "Name" unterscheiden!

Beispiel eines Codebausteins und seiner Einbindung in Formularen.

Codebaustein "Allgemein.elem", abgelegt im Ordner "/Admins/$R{MyTISM}/$R{Alarme}/$R{X}":

```
<Element>
  <Border etched="true" title="Allgemein">
    <View>
      <Element label="$R{Name}">
        <Text displayProperty="Name" columns="25"/>
      </Element>
      <!-- ...noch mehr Quelltext... -->
    </View>
  </Border>
</Element>
```

Formular "$R{_BOBasierterTermin} (Vorgebaut)":

```
  <Tab title="Allgemein" scrollable="true">
    <View>
      <!-- Einbindung von Codebausteinen: -->
      <Include name="/Admins/$R{MyTISM}/$R{Alarme}/$R{X}/Allgemein.elem"/>
      <Include name="/Admins/$R{MyTISM}/$R{Alarme}/$R{X}/Maske.elem"
parentClass="de.ipcon.db.core.BOBasierterTermin"/>
      <Element>
        <Border etched="true" title="Auslösung">
          <View>
            <Element label="$R{Attribut}">
              <Text displayProperty="Attribut" columns="25"/>
            </Element>
  ...
```

Formular "$R{_Hinweis} (Vorgebaut)":

```
  ...
    </View>
  </Tab>
  <Tab title="Allgemein" scrollable="true">
    <View>
      <!-- Einbindung von Codebausteinen: -->
      <Include name="/Admins/$R{MyTISM}/$R{Alarme}/$R{X}/Allgemein.elem"/>
      <Include name="/Admins/$R{MyTISM}/$R{Alarme}/$R{X}/Maske.elem"
parentClass="de.ipcon.db.core.Hinweis"/>
      <Include name="/Admins/$R{MyTISM}/$R{Alarme}/$R{X}/Sonstiges.elem"/>
    </View>
  </Tab>
  <Tab title="Auslösekriterien" scrollable="true">
    <View>
      <Element>
  ...
```

Reiter "CookedParameter" und "Codebausteine"

Die vorgebauten Formulare für Lesezeichen, Formulare, Schablonen und Reports beinhalten zwei Reiter namens "CookedParameter" und "Codebausteine". Unter "CookedParameter" kann man sich ansehen, wie der Quellcode des Strukturelements (aus dem Attribut "Parameter") letztendlich aussieht, nachdem der Inhalt aller Codebausteine eingefügt und alle L10n-Einträgen durch den entsprechenden sprachspezifischen Text ersetzt wurden. Bei Reports existiert außerdem noch "CookedReportDefinition" die dasselbe für den Inhalt des Attributs "ReportDefinition" anzeigt. Unter "Codebausteine" kann man sehen, welche Codebausteine vom aktuellen Strukturelement verwendet werden und diese direkt öffnen. Hier ist zu beachten, dass diese Liste erst gefüllt wird (technische Gründe...), wenn der Reiter "CookedParameter" des Strukturelements mindestens einmal angesehen wurde.

Pfadangaben für Codebausteine

Absolute Pfade mit "/" am Anfang (wie in obigen Beispielen) werden immer von der Wurzel des Navigationsbaumes (genauer eigentlich: Der Struktur-Hierarchie) aus aufgelöst.

Relative Pfade mit ".", ".." oder direkt einem Namen am Anfang werden vom "Standort" des aufrufenden Strukturelements aus aufgelöst. Dabei bezeichnet "." den aktuellen Standort (wird wohl eher selten benötigt), ".." den Elter des aktuellen bzw. des vorher im Pfad genannten Strukturelements. Normalerweise greift bei relativen Pfaden automatisch ein Fallback-Mechanismus; dieser funktioniert indem der Codebaustein (mit dem gegebenen relativen Pfad)erst vom Standort des aufrufenden Strukturelements, wenn er dort nicht gefundenwird von dessen Elter aus, dann ggf. von dessen Elter, etc. gesucht wird. Durch Angabe von `useFallback="false"` beim Include-Aufruf wird der Fallback-Mechanismus deaktiviert; in diesem Fall wird nur einmal, ausgehend vom aufrufenden Strukturelement aus, gesucht.

Benamsung von Codebausteinen

Der Name des Codebausteins sollte einen Hinweis darauf geben, um was es sich bei dem Inhalt handelt. Hierzu wird er mit einer Endung versehen. Oft verwendete Endungen sind: "button":: Inhalt besteht aus einem "button"-Element, ggf. mit zugehöriger Action. "elements":: Inhalt besteht aus mehreren, beliebigen XML-Elementen. "filter":: Inhalt definiert einen Filter für ein Lesezeichen. "script":: Inhalt ist ein Skript. "tab":: Inhalt ist ein ganzer Reiter ("Tab") eines Formulars. "table":: Inhalt ist eine Tabellendefinition. "view":: Inhalt ist ein "view"-Element für ein Formular.

Die Verwendung dieser (und ggf. weiterer Endungen) ist allerdings nur eine Konvention und zur Nutzung von Codebausteinen nicht unbedingt notwendig.

Example 1. Namensbeispiele:

```
EinstellungenNavigationsbaum.tab
Benutzer.table
```

Inhalt von Codebausteinen

Codebausteine können einen beliebigen Inhalt haben, angefangen von einem kurzen oder längeren normalem Text bis hin zu großen XML-Stücken.

Beispiele für mögliche Codebausteine:

```
text
```

```
text text text viel text
und noch mehr text
und noch weiterer text

dann ausserdem noch text
```

```
<element>text</element>
```

```
<element>
  <kindelement>text</kindelement>
</element>
```

```
<element attribut1="wert">
  <kindelement>text1</kindelement>
  <kindelement attribut="wert">text2</kindelement>
</element>
```

Zu beachten ist allerdings, dass es sich - aus technischen Gründen - bei dem Inhalt eines Codebausteins (mehr oder weniger) um ein wohlgeformtes XML-Dokument handeln muss. Dies bedeutet insb. dass es genau ein "Root"- bzw. "Wurzel"-Element geben muss; will man mehrere, in der gleichen "Hierarchie-Stufe" befindliche (XML-)Elemente in einem Codebaustein abspeichern, muss man in diesem Fall ein "künstliches"Wurzel-Element einfügen. Dieses trägt den Namen `Include` und wird beim Einfügenin den Quellcode anderer Struktruelemente einfach entfernt (d.h. es wird nur der Inhalt dieses Elements eingefügt.

Beispiel:

```
<element attribut1="wert"/>
<element>text</element>
<andereselement/>
```

muss geschrieben werden als:

```
<Include>
  <element attribut1="wert"/>
  <element>text</element>
  <andereselement/>
</Include>
```

Dieses künstliche "Include"-Element *kann* immer - also auch wenn sowieso eigentlich schon nur ein Wurzel-Element existiert - angegeben werden.

Beispiel:

```
<element attribut1="wert">text</element>
```

kann auch geschrieben werden als:

```
<Include>
  <element attribut1="wert">text</element>
</Include>
```

`hideComment` beim Einbinden eines Codebausteines

Beim Einbinden von Codebausteinen werden vor dem Code des eigentlichen Codebausteines standardmässig Kommentare eingesetzt, die Anfang und Ende des Codebausteines im Quelltext kennzeichnen. Der Mechanismus, der diese Kommentare erzeugt, fügt zwischen den Kommentartags und dem eigentlichen Inhalt des Codebausteines eine Anzahl von Leerzeichen ein. Dieses Verhalten ist offensichtlich eine Eigenart der genutzten XML-Bibliothek. Für Programmquelltext ist dieses Verhalten nicht weiter störend.
Wenn der Codebaustein jedoch z.B. für eine mehrzeilige Kundenadresse verwendet wird, so kann es passieren, daß die Leerzeichen, die hinter dem Ende des Kommentares automatisch eingefügt wurden, eine Verschiebung in der ersten Zeile der Adresse verursachen. Die erste Zeile, die in dem Fall einen Namen enthielt und in einem Report verwendet wurde, war im generierten Report nach rechts verschoben. Um diesen Effekt zu vermeiden läßt sich das Einfügen von Kommentaren beim Einbinden des Codebausteines pro Verwendung individuell deaktivieren. Dafür existiert das vordefinierte Argument `hideComment`. Es wird, analog zu den bereits beschriebenen Argumenten, als Attribut im `Include`-Statement wie im folgenden Beispiel eingegeben.

```
<Include name="codebaustein" hideComment="true"/>
```

Einziger - bekannter - Nachteil dieses Argumentes: Die Referenzpunkte, die man in den Cooked-Parameters etc. hat, um diesen Codebaustein zu finden - die XML-Kommentarzeilen - existieren nicht mehr.

Argumente für Codebausteine

Teilweise kann es vorkommen, dass ein Stück Quellcode in verschiedenen Strukturelementen *fast* gleich vorkommt, sich aber in einem oder mehreren kleinen Punkten unterscheidet:

Quellcode 1:

```
<element>
   Ein bisschen Text.
   <-- Fast gleich: -->
   <element attribut="eins"/>
   <-- Ende -->
   <weiteresElement/>
</element>
```

Quellcode 2:

```
<element>
   <einElement>inhalt</einElement>
   <nochEinElement/>
   <-- Fast gleich: -->
   <element attribut="zwei"/>
   <-- Ende -->
   <wiederumEinElement attr="wert"/>
</element>
```

Für diese Argumente kann man auch Standardwerte definieren (siehe u.a. obiger Screenshot), die automatisch genommen werden, wenn beim "Aufruf" desCodebausteins kein Wert für das Argument mit übergeben wurde. Dies ist insb. dann sinnvoll, wenn der Codebaustein oft verwendet wird, der Wert aber in den meisten Fällen gleich ist und nur ein- oder wenige Male ein anderer Wertbenötigt wird:

Quelltext eines Codebausteins der außerdem (über das Codebaustein-Formular) noch ein CodebausteinArgument "attrWert" mit Standardwert "eins" definiert hat:

```
<element attribut="$IP{attrWert}"/>
```

Quellcode 1:

```
<element>
   Ein bisschen Text.
   <!-- War: <element attribut="eins"/> -->
   <Include name="codebaustein"/>  <!-- attrWert="eins" braucht nicht angegeben zu
werden. -->
   <weiteresElement/>
</element>
```

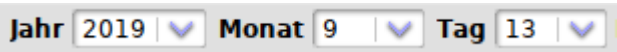
Quellcode 2:

```
<element>
   <einElement>inhalt</einElement>
   <nochEinElement/>
   <!-- War: <element attribut="zwei"/> -->
   <Include name="codebaustein" attrWert="zwei"/>
   <wiederumEinElement attr="wert"/>
</element>
```

Core-Codebausteine

jahrMonatTag.filter

Dieser Codebausten gibt die Möglichkeit, Einträge bestimmter Tabellen nach einem bestimmten Datumsattribut zu filtern.
Er erzeugt drei Drop-Downs: Eins für jeweils Jahr, Monat und Tag.



Der Filter richtet sich zunächst einmal nach den verfügbaren Einstellungen-Variablen, die gesetzt sind (entweder global, speziell für eine Gruppe, oder den Benutzer selbst), um die drei Filter mit Werten vorzubelegen.
Die Einstellungen-Variablen sind die folgenden:

- jahrMonatTagFilter.Jahr
- jahrMonatTagFilter.Monat
- jahrMonatTagFilter.Tag

Ist für ein Feld keine Einstellungen-Variable gesetzt, wird das Feld mit keinem konkreten Wert vorbelegt, steht also auf "alle".

Um das Jahr via einer Einstellungen-Variable zu besetzen(also Systemweit, gruppen- oder benutzerspezifisch), die aber nur für eine spezifische Tabelle gelten soll, kann man dem Codebaustein den Parameter "parmPostfix" mitgeben.
Ist dann eine Einstellungen-Variable `jahrMonatTagFilter.Jahr.<parmPostfix>` gesetzt, hat diese nur für Tabellen aus Strukturelementen, die dem Codebaustein den selben Postfix mitgeben, auswirkungen.

Weitere optionale Parameter des Codebaustens:

- attrDatum 1.)
- parmJahrVon 2.)
- parmJahrBis 3.)
- JahrDefaultIsAll 4.)

1.) Der Name des Attributes, nach dessen Datum gefiltert werden soll. Default ist "Crea".
2.) + 3.) Das Start- und Endjahr, von denen ausgewählt werden darf.
Beispiele:

- ParmJahrVon="2000", ParmJahrBis="2019" → logischerweise alle Jahre von 2000 bis 2019
- ParmJahrVon="-10", ParmJahrBis="+10" → die letzten und nächsten 10 Jahre (im Jahr 2019: 2009-2029)

Die Default-Werte liegen hier bei: ParmJahrVon="2000" und ParmJahrBis="+0", also von 2000 bis zu dem aktuellen Jahr.
4.) JahrDefaultIsAll: Ein Parameter, mit dem, unabhängig von Einstellungen-Variablen, Strukturelemente dem Codebaustein vorgeben können, dass der "Jahr"-Filter mit "alle" vorbelegt werden soll. Default ist "false".

Problembehebung

IllegalArgumentException: Invalid parameter "xyz" given...

Diese Fehlermeldung bedeutet, dass beim "Aufruf" eines Codebausteins ein Argument angegeben wurde, das für diesen Codebaustein nicht definiert wurde. Wenn nicht wirklich einfach vergessen wurde, das Codebaustein-Argument am Codebaustein zu definieren (s.o.) kann das auch passieren, falls der Benutzer *keine ausreichenden Rechte* hat, Codebaustein-Argumente zu lesen. In diesem Fall wird dann zwar der Codebaustein (für den Leserechte gesetzt sind) geladen, aber die eigentlich dafür definierten Codebaustein-Argumente können nicht geladen werden (was aufgrund des Designs des Rechtesystems aber nicht zu einer Fehlermeldung führen kann und soll) und deswegen sieht es so aus, als wären für den Codebaustein keine Argumente vorhanden, was wiederum diesen Fehler zur Folge hat.

Benachrichtigungen

Dokumentation zum Benachrichtigungssystem befindet sich im Admin-Handbuch.

Alarme

Grundlagen

Es ist möglich, in MyTISM sog. Alarme zu definieren, bei deren Auslösung die für den jeweiligen Alarm eingetragenen Empfänger benachrichtigt oder andere Aktionen ausgeführt werden. Es gibt vier Varianten von Alarmen, die für jeweils unterschiedliche Zwecke gedacht sind.

Einfacher Termin

Dies ist die einfachste Alarm-Variante; der Alarm wird einfach zu einem vorher eingetragenen, festen Zeitpunkt ausgelöst.
Alternativ gibt es auch die Möglichkeit, den Alarm mit einer konfigurierbaren Frequenz wiederholt auslösen zu lassen.

> *Beispiel:* Am 22. Juli 2011 um 14:00 Uhr ist eine Projektbesprechung angesetzt. Alle Projektteilnehmer sollen eine Viertelstunde vorher eine Benachrichtigung erhalten.

BO-basierter Termin

Diese Alarm-Variante ähnelt der Variante "Einfacher Termin" insofern, als dass die Alarme ebenfalls zu einem festgelegten Zeitpunkt ausgelöst werden. Allerdings "überwacht" ein BO-basierter Termin eine Menge von Objekten ("BOs") und legt für jedes dieser Objekte einen eigenen Auslösezeitpunkt fest.

> *Beispiel:* Für alle Mitarbeiter ist der jeweilige Geburtstag eingetragen. Die Mitarbeiter sollen jedes Jahr eine automatische Gratulation erhalten (ob das wirklich so eine tolle Idee ist, sei mal dahingestellt ...).

Hinweise

Diese Alarm-Variante dient dazu, Alarme auszulösen, wenn bestimmte Ereignisse in der MyTISM-Anwendung auftreten bzw. bestimmte Änderungen an Objekten erfolgen.

> *Beispiel:* Der Chef der Buchhaltung möchte benachrichtigt werden, sobald der Bestand eines Kontos unter 100,- EUR sinkt.

Wiedervorlagen

Diese Alarm-Variante dient dazu, Alarme auszulösen, wenn bestimmte Ereignisse in der MyTISM-Anwendung *nicht* innerhalb einer festgelegten Zeit aufgetreten sind bzw. bestimmte Änderungen an Objekten innerhalb einer festgelegten Zeit *nicht* erfolgt sind.

> *Beispiel:* Der Projektleiter möchte benachrichtigt werden, wenn sich der Status eines Projekts zwei Tage lang nicht geändert hat.

Gegenbenenfalls kann es in Ihrer MyTISM-Anwendung auch noch eigene Untervarianten dieser Alarm-Typen geben, die für spezielle Zwecke gedacht sind. Diese besitzen ggf. zusätzlich zu den

normalen Eigenschaften der Alarme noch zusätzliche Eigenschaften und Funktionen. Ob solche Untervarianten existieren, wofür sie benutzt werden und weitere Informationen hierzu kann Ihnen Ihr MyTISM-Administrator geben.

Vorbereitung und Konfiguration

Alarmsystem-Lizenz einspielen

Das Alarmsystem ist eine optionale Erweiterung des Standard-MyTISM-Systems. Um es aktivieren und nutzen zu können, müssen Sie zuerst eine gültige Alarmsystem-Lizenz erworben und auf dem Server eingespielt haben.

Alarmsystem aktivieren

Das Alarmsystem ist normalerweise deaktiviert, d.h. Sie können zwar beliebige Alarme anlegen, diese werden aber von MyTISM erst einmal in keiner Weise behandelt.

Um das Alarmsystem zu aktivieren müssen Sie in der Datei `mytism.ini` im Abschnitt `[Alarme]` die Einstellung `activateAlarme` auf `if_possible` oder `mandatory` setzen. Sowohl `if_possible` als auch `mandatory` starten das Alarmsystem; sie unterscheiden sich lediglich darin, dass bei `mandatory` eine auffälligere Fehlermeldung ausgegeben wird (ursprünglich sollte der Serverstart abgebrochen werden, was nach Diskussion dann aber deaktiviert wurde).

Sollte der entsprechende Abschnitt noch nicht existieren, fügen Sie ihn einfach ein.

```
[Alarme]
activateAlarme=if_possible
```

Wenn Ihre MyTISM-Installation mehrere synchronisierende Server umfasst, müssen - und dürfen - Sie das Alarmsystem aus technischen Gründen nur auf dem autoritativen Server aktivieren. Wenn Sie obigen Eintrag in der Datei `mytism.ini` eines nicht-autoritativen Servers eintragen, wird nur eine Warnmeldung im Log ausgegeben und das Alarmsystem dort *nicht* aktiviert.

Sync-Events behandeln

Sollten Sie in der Datei `mytism.ini` im Abschnitt `[Alarme]` noch einen Eintrag `handleSyncEvents=1` oder `handleSyncEvents=0` aufgeführt haben, können Sie diese Zeile löschen, da sie zu einer mittlerweile nicht mehr benötigten und nicht mehr unterstützten Konfigurationsmöglichkeit gehört. Falls die Zeile vorhanden ist wird sie ignoriert.

Benachrichtigungssystem aktivieren

Sollen Empfänger beim Auslösen eines Alarms benachrichtigt werden muss das Benachrichtigungssystem ebenfalls aktiviert und entsprechend konfiguriert sein. Wenn dies nicht der Fall ist, können keine Benachrichtigungen (per e-Mail o.Ä) versandt werden und die ausgelösten Alarme sind nur über das `AlarmAusloesungen`-Lesezeichen bzw. den entsprechenden Reiter z.B. im Benutzerformular ersichtlich.

Anlegen und Verwalten von Alarmen

Alarme sind ganz normale Objekte und können mit den entsprechenden Schablonen, Lesezeichen und Formularen angelegt und verwaltet werden. Die automatisch generierten Schablonen, Lesezeichen und Formulare befinden sich im Ordner `Admins MyTISM Alarme`. Diese Strukturelemente sind normalerweise nur für MyTISM-Administratoren verfügbar.

Evtl. existieren auf Ihrer speziellen MyTISM-Installation auch noch weitere, angepasste Formulare und Lesezeichen oder Formulare und Lesezeichen für eigene Alarm-Untervarianten. Diese befinden sich dann möglicherweise in anderen Ordnern; weitere Informationen hierzu kann Ihnen Ihr MyTISM-Administrator geben.

Gruppe "Admins Alarmsystem"

Es gibt in MyTISM-Applikationen eine automatisch angelegte Gruppe "Admins Alarmsystem". Benutzer, die dieser Gruppe zugewiesen wurden, haben automatisch alle Rechte um Alarme und damit zusammenhängende Objekte zu erstellen und zu verwalten.

Außerdem steht ihnen im Gruppen-Ordner ein Lesezeichen "Alarme" zur Verfügung, mittels derer sie die im System vorhandenen Alarme auflisten, öffnen, editieren und über das Kontext-Menü neue Alarme anlegen können.

Alarme aktivieren und deaktivieren

Alle Alarme besitzen ein "Aktiv"-Flag. Ist dieses gesetzt, so ist der Alarm aktiviert und kann ausgelöst werden. Ist das Flag nicht gesetzt - der Standard bei neuen Alarmen - so ist der Alarm deaktiviert und löst *nicht* aus.

Sie können hiermit einen Alarm quasi "vorbereiten", in dem Sie alle benötigten Daten des Alarms eintragen, aber das "Aktiv"-Flag noch nicht setzen. Der Alarm ist dann bereits im System bekannt, wird aber noch nicht behandelt. Sie können in diesem Fall das "Aktiv"-Flag zu einem späteren Zeitpunkt setzen und den Alarm speichern; der Alarm wird dann ab diesem Zeitpunkt behandelt.

Für BO-basierte Termine und Wiedervorlagen erfolgt die Initialisierung der WiedervorlageStatus bzw. BOBasierterTerminStatus in jedem Fall beim ersten Speichern des Alarms, da diese Informationen zum Funktionieren dieser Alarme essentiell sind und immer benötigt werden.

Die Aktivierung von neu angelegten (und auf "Aktiv" gesetzten) Hinweisen und EinfachenTerminen geschieht sehr schnell, die Aktivierung von BO-basierten Terminen und Wiedervorlagen kann aus technischen Gründen, je nach der Anzahl der zu "überwachenden" Objekte, etwas Zeit in Anspruch nehmen (siehe Wiedervorlagestatus).

Testmodus für Alarme

Der Testmodus für Alarme ist noch in Arbeit; ggf. ändert sich das Verhalten in diesem Bereich in Zukunft noch.

Alle Alarme können in einem Testmodus betrieben werden; hierzu muss das entsprechende "Testmodus"-Flag gesetzt werden. Ist dieses gesetzt, so löst der Alarm keine Benachrichtigungen aus und erzeugt auch keine AlarmAuslösungen-Objekte.

Es werden lediglich entsprechende Info-Meldungen im Log ausgegeben, mittels derer verfolgt werden kann, was bei der Auslösung passiert wäre.

Sonstige bei der Auslösung normalerweise erfolgende Dinge passieren jedoch weiterhin: So werden z.B. einfache Termine auch gelöscht, wenn sie im Testmodus "ausgelöst" wurden, etc. Für BO-basierte Termine und Wiedervorlagen erfolgt die Aktualisierung der WiedervorlageStatus bzw. BO-basierter TerminStatus (siehe alarme_alarmAusloesungen) in jedem Fall weiterhin, da diese Informationen zum Funktionieren dieser Alarme essentiell sind und immer benötigt werden.

Gemeinsame Eigenschaften aller Alarme

Alle Alarme haben bestimmte Eigenschaften gemeinsam:

Erster Reiter

Name

Pflichtfeld - Der Name oder Titel eines Alarms sollte den Alarm kurz und prägnant benennen. Der Name kann frei gewählt werden und kann z.B. bei der Anzeige der Alarme im zugehörigen Lesezeichen oder bei den Benachrichtigungen bei der Alarm-Auslösung benutzt werden. Es ist sehr sinnvoll, jedoch keineswegs zwingend, dass unterschiedliche Alarme unterschiedliche Namen haben :-)

Beschreibung

Optional - Die Beschreibung kann einen längeren Kommentar bzw. eine längere Beschreibung des Alarms beinhalten. Dieser Text kann z.B. bei den Benachrichtigungen benutzt werden.

Empfänger "Sende Benachrichtigungen an ... diese(n) Empfänger (CC) ... (und) diese(n) Empfänger (BCC)"

Mindestens eines von "Empfänger (CC)", "Empfänger (BCC)" oder "Benachrichtigungsskript" muss gegeben sein - Wie bereits erwähnt können Empfänger definiert werden, die bei der Auslösung eines Alarms benachrichtigt werden sollen.
Alle "Empfänger (CC)" sind bei der Benachrichtigung für alle anderen Empfänger einsehbar; Benachrichtigungen für "Empfänger (BCC)" werden dagegen einzeln versendet, so dass kein Empfänger über die anderen Bescheid weiß.
Ein Empfänger, z.B. ein Benutzer, erhält für eine Alarm-Auslösung immer nur *eine* Benachrichtigung, auch wenn z.B. für einen Alarm mehrere Gruppen als Empfänger eingetragen wurden und der Benutzer Mitglied in mehreren dieser Gruppen ist oder der Benutzer selbst ebenfalls für den Alarm eingetragen wurde.

Benachrichtigungsvorlage "Erstelle die Benachrichtigung mit ..."

Pflichtfeld - Die BenachrichtigungsVorlage wird beim Versand von Benachrichtigungen verwendet und gibt die Texte für Betreff und Nachrichtentext vor sowie ermöglicht die Definition von Inline-Bildern, die in HTML-formatierten Email via `cid:` mit der angegebenen Content-Id in URLs referenziert werden können. Es handelt sich hierbei zwar um ein eigenständiges Objekt, dessen Daten können jedoch direkt im Alarm-Formular bearbeitet werden.

"Alte Alarme nur auslösen wenn nicht älter als", Reiter "Erweitert"

Optional - Es kann passieren, dass Alarme zu einem bestimmten Zeitpunkt hätten ausgelöst werden sollen, dies jedoch nicht passiert ist, z.B. weil zu dieser Zeit das Alarmsystem deaktiviert war. Die entsprechenden Auslösungen werden dann normalerweise später (also z.B. sobald das Alarmsystem wieder aktiviert wird) "nachgeholt".
Wenn Sie möchten, dass dabei nur Alarme ausgelöst werden, bei denen der eigentliche Auslösezeitpunkt nicht *zu* weit in der Vergangenheit liegt, können Sie hier angeben, wie weit die eigentliche Auslösung maximal zurück liegen darf.

Benachrichtigungsskript "Sende Benachrichtigungen mittels dieses Skripts", Reiter "Erweitert"

Mindestens eines von Benutzer, Gruppe oder Benachrichtigungsskript muss gegeben sein - Diese Eigenschaft dient dazu, bei der Auslösung von Alarmen eigene Aktionen ausführen zu können und wird nur benötigt, wenn die Standardmöglichkeiten zur Benachrichtigung von Benutzern bzw. Gruppen einmal nicht ausreichen. Ausführlichere Informationen hierzu finden Sie im Abschnitt Benachrichtigungsskript.

Reiter "Erweitert"

"Verantwortlicher"

Optional - Hier kann ein Benutzer angegeben werden, der in irgendeiner Weise "verantwortlich" für diesen Alarm ist. Wird der Alarm aufgrund von zu vielen Fehlern deaktiviert, wird diesem Benutzer eine Benachrichtigung als Information geschickt; ebenso wird z.B. bei Fehlern im Auslöseskript eine Benachrichtigung an den Verantwortlichen geschickt. Außerdem wird der Wert dazu verwendet bei Benachrichtigungen bei Alarmauslösung den "Absender" dieser Benachrichtigung zu setzen, falls nicht explizit ein abweichender *Absender* am Alarm gesetzt wurde.
Wenn hier nichts angegeben ist und auch kein *Absender* gesetzt ist, wird als Absender der interne Benutzer des Alarmsystems benutzt. Bei Benachrichtigungen, die per e-Mail verschickt werden, wird die erste e-Mail-Adresse dieses Benutzers als Absender der Mails gesetzt (wenn mehrere Adressen für den Benutzer verfügbar sind, ist nicht definiert, welche davon "die erste" ist).

"Bei Ausfall benachrichtigen"

Optional - Hier kann eine Gruppe angegeben werden, die bei Fehlern am Alarm, zusätzlich zum Verantwortlichen, benachrichtigt wird.

"Überwachung starten ab"

Optional - Normalerweise werden Alarme sofort aktiv, sobald sie erstellt und als "Alarm ist aktiv" definiert wurden. Falls Sie hier ein Datum und ggf. eine Zeit eintragen wird der Alarm erst zu diesem Zeitpunkt aktiv und wird nicht vor diesem Zeitpunkt ausgelöst.

"Will verschlüsselte Benachrichtigungen" und "Will signierte Benachrichtigungen"

Optional - Alarme können die Standardeinstellungen des Systems für verschlüsselte und/oder signierte Benachrichtigungen gezielt überschreiben. Eingestellte Werte hier übersteuern die Standardeinstellungen aber werden wiederum selbst von ggf. vorhandenen Einstellungen der Benutzer übersteuert.

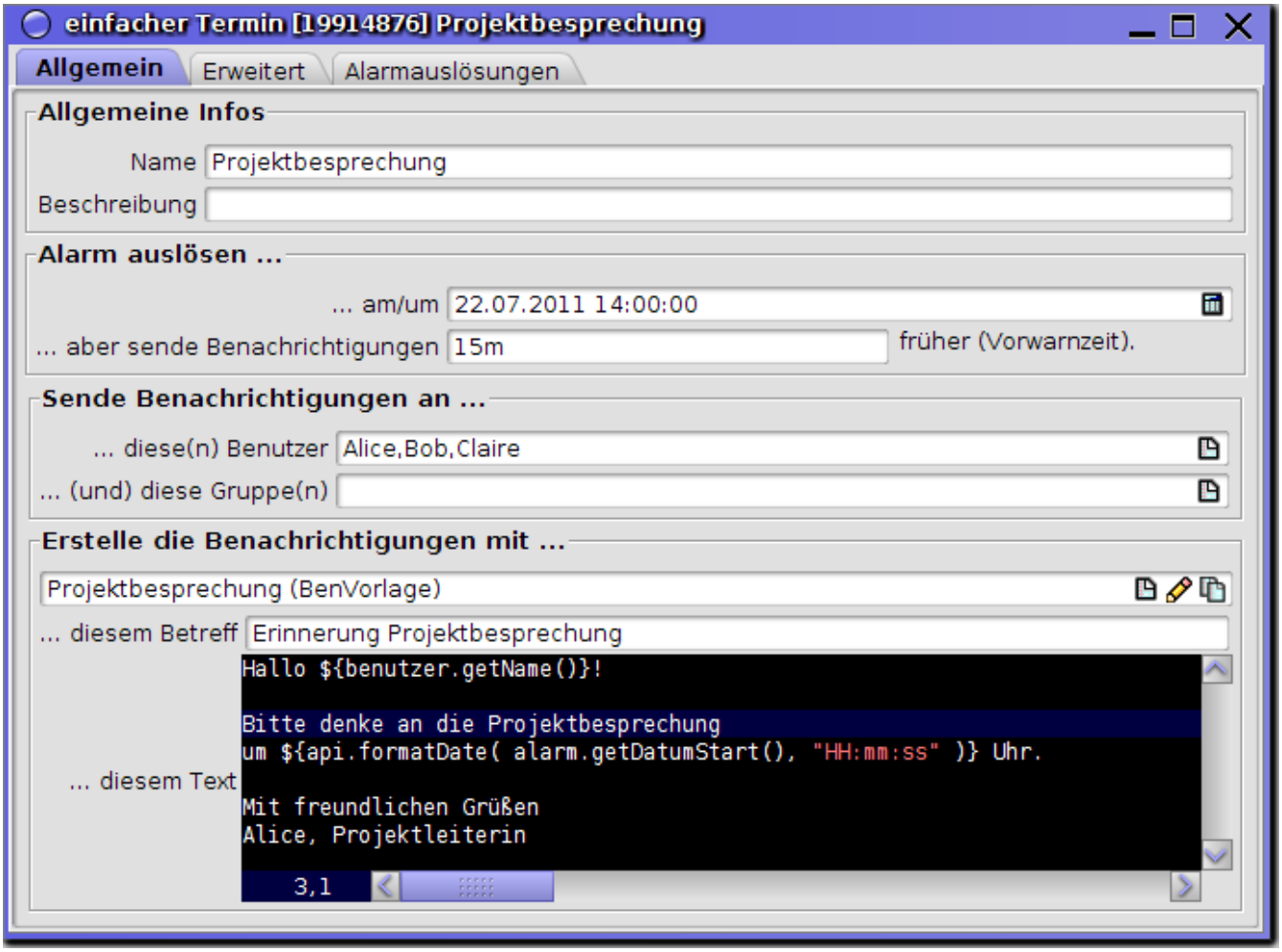
"Hänge statt dem auslösenden Objekt an die Benachrichtigungen an ..."

Wird unter Anhängen von (weiteren) Objekten genauer erklärt.

Einfacher Termin

Wie oben bereits erwähnt, handelt es sich bei einfachen Terminen um Alarme, die, ohne dass weitere Bedingungen erfüllt sein müssen, einfach zu einem festgelegten Zeitpunkt ausgelöst werden.

> *Beispiel:* Am 22. Juli 2011 um 14:00 Uhr ist eine Projektbesprechung angesetzt. Alle Projektteilnehmer sollen eine Viertelstunde vorher eine Benachrichtigung erhalten.



Allgemeine Eigenschaften festlegen

Geben Sie dem einfachen Termin einen kurzen aber aussagekräftigen Namen, und ggf. wenn sinnvoll eine längere Beschreibung.

"Alte Alarme nur auslösen wenn nicht älter als" und "Verantwortlicher" können Sie, bei Bedarf, auf dem Reiter "Erweitert" angeben.

Wann soll der einfache Termin stattfinden?

Einfache Termine können entweder einmalig, zu einem fest eingetragenen Zeitpunkt, oder wiederholt, mit einer konfigurierbaren Frequenz, ausgelöst weden.

An einem festen Zeitpunkt

Geben Sie bei "Alarm auslösen" → "... am/um" das Datum und die Zeit an, wann der einfache Termin stattfindet bzw. beginnt.
Der Alarm löst zu diesem Zeitpunkt einmal aus und wird danach automatisch *gelöscht*.

Wiederholt

Geben Sie bei "Alarm auslösen" → "... (oder stattdessen) wiederholen nach Muster" die Definition an, die festlegt, mit welcher Frequenz der Termin auslösen soll.
Der Alarm löst immer wieder aus, zu Zeitpunkten die anhand der angegebenen Definition bestimmt werden. Hilfe zur Definition (im sog. "Cron-Format") finden Sie z.B. unter http://www.nncron.ru/help/EN/working/cron-format.htm

Vorwarnzeit

Normalerweise wird der einfache Termin erst zum angegebenen bzw. ermittelten Zeitpunkt ausgelöst und eventuelle Benachrichtigungen werden also auch erst dann versendet.

Wenn die Auslösung bereits statt finden soll, bevor der einfache Termin *eigentlich* "startet", können Sie unter "... aber sende Benachrichtigungen ... früher (Vorwarnzeit)" optional eine Zeitspanne angeben, um wieviel früher dem angegebenen bzw. ermittelten Zeitpunkt dies erfolgen soll.

Wer soll Benachrichtigungen erhalten und wie sollen diese aussehen?

Geben Sie bei "Sende Benachrichtigungen an ... diese(n) Benutzer ... (und) diese Gruppe(n)" ein oder mehrere Benutzer und/oder Gruppen an, der oder die bei der Auslösung des einfachen Termins benachrichtigt werden soll(en).

Ein Benutzer erhält für eine Alarm-Auslösung immer nur eine Benachrichtigung, auch wenn z.B. für einen Alarm mehrere Gruppen eingetragen wurden und der Benutzer Mitglied in mehreren dieser Gruppen ist oder der Benutzer selbst ebenfalls für den Alarm eingetragen wurde.

Bei "Erstelle die Benachrichtigung mit ..." wählen Sie eine Textvorlage aus, mittels derer der Betreff und der Text der zu versendenden Benachrichtigungen festgelegt werden. Alternativ können Sie mittels des Schreibstift-Icons auch eine neue, eigene Vorlage direkt erstellen.

Wenn der Text für die Benachrichtigung (von Leerzeichen abgesehen) mit `<html>` beginnt, werden die daraus generierten e-Mails als HTML-Mails verschickt. Wenn der Text nicht auf diese Weise beginnt werden die e-Mails als ganz normale Textmails verschickt.

Alice, die Projektleiterin, möchte Bob und Claire, die beiden anderen Projektmitarbeiter, am 22. Juli um 14:00 Uhr zu einer Besprechung einladen.

Sie erstellt also einen neuen EinfachenTermin mit Namen "Projektbesprechung". Unter "Alarm auslösen ... am/um" trägt sie "22.07.2011 14:00" ein.

Damit jeder auch noch Zeit hat, seine Sachen zusammenzusuchen und von seinem Büro in den Besprechungsraum am anderen Ende des Gebäudes zu gelangen, setzt sie eine "... aber sende Benachrichtigungen ... früher (Vorwarnzeit)" von 15 Minuten (d.h. sie trägt "15m" ein), so dass die Benachrichtigungen um 13:45 Uhr bei den Teilnehmern ankommen.

Zu benachrichtigende Benutzer sind natürlich Bob und Claire und sie trägt sich selbst ebenfalls nochmal für eine Erinnerung ein (da sie leider notorisch vergesslich ist :-).

Als Benachrichtigungsvorlage wählt sie die bereits in der Datenbank vorhandenen Vorlage "Projektbesprechung".

Nachdem sie den EinfachenTermin gespeichert hat, wird dieser von der MyTISM-Anwendung in eine interne Liste eingetragen. Am 22. Juli um 13:45 Uhr werden dann automatisch entsprechende Benachrichtigungen an Alice, Bob und Claire verschickt und der einfache Termin wird automatisch gelöscht.

BO-basierter Termin

BO-basierte Termine werden einer Menge von Objekten zugeordnet, von denen entweder jedes einen eigenen, festen Auslösungszeitpunkt bereits selbst definiert oder für welche der BO-basierte Termin jeweils einen eigenen, festen Auslösezeitpunkt berechnet.

> *Beispiel:* Für alle Mitarbeiter ist der jeweilige Geburtstag eingetragen. Die Mitarbeiter sollen jedes Jahr eine automatische Gratulation erhalten (ob das wirklich so eine tolle Idee ist, sei mal dahingestellt ...).

Allgemeine Eigenschaften festlegen

Geben Sie dem BO-basierten Termin einen kurzen aber aussagekräftigen Namen, und ggf. wenn sinnvoll eine längere Beschreibung.

"Alte Alarme nur auslösen wenn nicht älter als" und "Verantwortlicher" können Sie, bei Bedarf, auf dem Reiter "Erweitert" angeben.

Welche Objekte sollen "überwacht" werden?

Legen Sie fest, für welche Objekte der BO-basierte Termin auslösen soll. Dazu muss dem BO-basierten Termin unter "Überwache die Objekte ..." eine sog. `BOMaske` zugewiesen werden, die die Menge der zu "beobachtenden" Objekte definiert.

Sie können hier entweder eine bereits vorhandene `BOMaske` auswählen oder alternativ mittels des Schreibstift-Icons auch eine neue, eigene Maske direkt erstellen. Für die meisten Anwendungszwecke ist es ausreichend, unter "... vom Typ" einfach einen Objekt-Typ (Entität) auszuwählen, womit dann alle Objekte dieses gewählten Typs vom BO-basierten Termin "beobachtet" werden.

Die Erstellung von `BOMasken` wird im Abschnitt "BOMasken" im Kapitel "Rechteverwaltung" im Administrator-Handbuch ausführlich erklärt, für weitere Informationen sehen Sie bitte dort nach. Anzumerken ist hier noch, dass die Eigenschaft `Attribut` von `BOMaske` nur für die Rechteverwaltung notwendig ist und für das Alarmsystem nicht benutzt wird; evtl. hier eingetragene Werte werden vom Alarmsystem einfach ignoriert.

Exkurs: Vor- und Nachteile der verschiedenen BOMasken-Typen

tl;dr: In den meisten Fällen sind `OQLBOMasken` für BO-basierte Termine und Wiedervorlagen die richtige Wahl; für Hinweise ist der benutzte BOMasken-Typ nicht so relevant.
Falls ein `Skript` benutzt werden muss, sollten die Informationen im Abschnitt `Skript` immer berücksichtigt werden.

Insb. für BO-basierte Termine und Wiedervorlagen ist es sinnvoll, den richtigen Typ von BOMaske einzusetzen, da das die Leistung des Systems deutlich beeinflussen kann.

Insb. die Dauer der Berechnung der WiedervorlageStatus bzw. BOBasierterTerminStatus kann hiermit teils sehr verringert werden.
Bei der Initialisierung bzw. Neuberechnung dieser Statuswerte müssen im Normalfall alle möglicherweise passenden Objekte aus der Datenbank geladen und dann mit der definierten Maske überprüft werden. Je nach Menge der Objekte und der Komplexität der Prüfung kann das teils sehr lange dauern.

Mit Benutzung des richtigen BOMaske-Typs können aber sowohl das Abfragen und Laden aus der Datenbank als auch die nachfolgende Überprüfung der Objekte zum Teil deutlich optimiert werden.

Skript

Generell gilt: Wenn eine BOMaske ein `Skript` nutzt, ist das auf jeden Fall eher aufwändig, selbst wenn es sich um ein sehr einfaches Skript handelt z.B. nur ein `!Ldel`. Aufwändigere Skripte, in denen z.B. Many-Relationen des BOs für die Prüfung heran gezogen werden, können die Performance dann noch mal sehr deutlich verschlechtern und sollten möglichst vermieden werden.

Falls Skripte mehrere Bedingungen prüfen, sollten die schnell und mit wenig Aufwand abzuprüfenden Bedingungen auf jeden Fall am Anfang geprüft werden und falls eine davon bereits *nicht* zutrifft, sollte das Skript bereits verlassen werden. Erst danach sollten weitere Prüfungen, von den "günstigsten" hin zu den "teuersten", erfolgen.

Beispiel, wie die Prüfungen durchgeführt werden sollten

```
// Ldel ist nur ein Boolean-Flag, sehr günstig zu prüfen:
if (bo.Ldel) {
   return false
}
// String-Vergleich, schon etwas "teurer" aber noch nicht allzu aufwändig:
if (bo.Name == null || bo.Name != 'Gewünschter Name') {
   return false
}
// Many-Relationen abzufragen ist sehr teuer, da die Objekte jedesmal erst geladen
werden müssen:
def hatBevorzugtesMitglied = bo.Mitglieder.find{ it.istBevorzugt }
if (!hatBevorzugtesMitglied) {
   return false
}
return true
```

Die wenigsten Alarme wollen aber einfach *alle* Objekte eines bestimmten Typs überwachen. Daher ist es normalerweise notwendig, zusätzliche Kriterien, denen die Objekte genügen müssen, zu definieren.

Bei normalen BOMaske ist das aber nur mit Benutzung eines Skripts möglich. Aus diesem Grund ist

es normalerweise vorteilhaft, wenigstens für Alarme - insb. BO-basierte Termine und Wiedervorlagen - einen der anderen verfügbaren BOMasken-Typen zu benutzen.

Insbesondere bei der Initialisierung oder Neuberechnung der WiedervorlageStatus bzw. BOBasierterTerminStatus müssen oft sehr viele Objekte mit den Kriterien der Maske geprüft werden. Die speziellen BOMasken-Typen `GrooqlBOMaske` und insb. `OQLBOMaske` erlauben, diese Prüfungen effizienter durchzuführen und insbesondere auch bereits bei der Abfrage der zu prüfenden Objekte aus der Datenbank diese zu filtern und die Menge damit möglichst klein zu halten.

Grooql-BOMasken

Grooql-BOMasken erlauben, neben dem oben genannten `Skript` noch ein `GrooqlScript` anzugeben.

Die Bedingungen, die dieses `GrooqlScript` definiert, können *bereits bei der Abfrage* der Objekte aus der Datenbank berücksichtigt werden. Damit wird die Menge der "nachträglich" mit der Maske zu prüfenden Objekte bereits im Vorfeld möglichst klein gehalten. Viele Objekte, die nicht zur Maske passen, werden dann erst gar nicht zur Prüfung geladen.

Die Objekte, die dennoch aus der Datenbank geladen werden, werden dann in einem ersten Schritt noch einmal mit dem definierten `GrooqlScript` geprüft.
Falls diese Prüfung positiv ausfällt (das `GrooqlScript` liefert `true` zurück) wird das Objekt danach - wie bei einer normalen BOMaske auch - dann nochmal mit einem evtl. definierten `Skript` geprüft. Dieses muss auch `true` zurückgeben, damit das Objekt dann vom Alarm berücksichtigt wird.

Weitere Informationen zu Grooql finden sich im entsprechenden Handbuch-Kapitel.

OQL-BOMasken

OQL-BOMasken erlauben, neben dem oben genannten `Skript` noch ein oder mehrere `WhereClauses` (OQL-WHERE-Klauseln) anzugeben.

Diese OQL-Klauseln werden direkt bei der Abfrage der Objekte aus der Datenbank berücksichtigt. Damit wird die Menge der "nachträglich" mit der Maske zu prüfenden Objekte bereits im Vorfeld möglichst klein gehalten. Viele Objekte, die nicht zur Maske passen, werden dann erst gar nicht zur Prüfung geladen.

Nur die Objekte, die dennoch aus der Datenbank geladen werden, werden dann mit einem evtl. definierten `Skript` geprüft.
Oft mag es aber möglich sein, bereits alle gewünschten Kriterien als `WhereClauses` zu definieren, so dass gar kein `Skript` angegeben werden muss und die nachträgliche Prüfung mit `Skript` ganz weggelassen werden kann.

Wann soll der BO-basierte Termin (für ein Objekt) ausgelöst werden?

Es gibt zwei Möglichkeiten, zu bestimmen, wann der BO-basierte Termin für ein bestimmtes Objekt

ausgelöst wird: Durch Angabe eines Attributes, das einfach ausgelesen werden soll, oder durch ein Skript, welches ausgewertet wird und für jedes Objekt das Auslösedatum berechnet.

Auslösedatum aus Objekt-Attribut auslesen

Unter "... das Datum aus Attribut" können Sie den Namen eines Attributes der Objekte auswählen, aus dem der Auslösezeitpunkt gelesen werden soll.

Es werden nur Attribute angezeigt, die einen Datumswert beinhalten, also im Schema mit Typ `Datetime` (oder einem davon abgeleiteten Typ) definiert sind.

Wenn möglich sollten Sie diese Variante der Variante mit Skript (s.u.) vorziehen, da sie

1. weniger Schreibarbeit und keine Kenntnisse in Skriptprogrammierung erfordert
2. die Anforderungen an das System geringer sind und
3. direkt bei der Definition des Alarms überprüft werden kann, ob alle Angaben korrekt sind - bei einem Skript kann das normalerweise erst festgestellt werden, wenn zur Laufzeit bei der Auswertung des Skripts ein Fehler auftritt.

Das Auslösedatum wird nur zu bestimmten Zeitpunkten (Erstellung des Alarms, Start der Überwachung für ein Objekt, Änderung des Auslösedatum-Attributs oder -Skripts) berechnet. Falls ein nicht-persistentes Attribut zur Ermittlung des Auslösedatums für überwachte Objekte benutzt wird, kann es - je nachdem wie das virtuelle Attribut seinen Wert bestimmt - sein, dass sich der Wert anderweitig, z.B. zeitabhängig, ändert. In solchen Fällen kann diese Änderung vom Alarm nicht registriert werden und es wird weiter das bestehende Auslösedatum für das Objekt benutzt!

Auslösedatum mit Skript berechnen

Falls das einfache Auslesen eines Attributwertes für Ihre Zwecke nicht ausreicht, können Sie alternativ unter "... das Datum, das dieses Skript liefert, erreicht ist" ein Skript angeben, welches das Datum für die Auslösung berechnet. Damit das entsprechende Eingabefeld angezeigt wird, darf in der Auswahlbox kein Attribut angewählt sein (Eintrag "(kein Attribut, benutze Skript)" muss ausgewählt sein).

Es existiert noch ein Bug, der sporadisch auftritt, so dass "... das Datum, das dieses Skript liefert, erreicht ist" nicht angezeigt wird und die Eingabe eines Scripts nicht möglich ist.

Das so definierte Skript führt der BO-basierte Termin dann für jedes seiner zu überwachenden Objekte aus. Dieses Skript muss dann einen Wert vom Typ `java.util.Date` zurückliefern, welcher angibt, wann der Alarm für das entsprechende Objekt ausgelöst werden soll.

Wie das Skript diesen Zeitpunkt bestimmt, ist im Prinzip vollkommen egal; es könnte z.B. theoretisch ebenfalls einfach nur den Wert eines Attributes des Objektes auslesen und diesen zurückgeben (wobei dann die Benutzung eines Skripts natürlich nicht wirklich Sinn macht) oder aber auch beliebig komplizierte Berechnungen ausführen, um das Auslösedatum für das aktuelle

Objekt zu errechnen.

Skript zur Berechnung des nächsten Geburtstages:

```
kal = Calendar.getInstance()
kal.setTime(bo.getGeburtstag()) // Auf Geburtstag initialisieren.
kal.set(Calendar.HOUR_OF_DAY, 10) // Auslösen um 10 Uhr morgens.
now = new Date()
while (kal.getTime().before(now)) { // Nächsten Termin finden.
  kal.roll(Calendar.YEAR, true) }
return kal.getTime() // Als Date() zurückgeben.
```

Zu beachten ist, dass das Skript möglichst schnell ein Ergebnis zurückliefern sollte, um das Alarmsystem nicht unnötig zu verlangsamen.

Außerdem *muss* das Skript in jedem Fall ein Objekt vom Typ `java.util.Date` zurückliefern - also nicht etwa gar keinen Wert oder einen Wert von einem anderen Typ! Sollte das passieren, oder sollte irgendein Fehler im Skript auftreten, wird für das entsprechende Objekt kein Alarm ausgelöst.

Das Auslösedatum wird nur zu bestimmten Zeitpunkten (Erstellung des Alarms, Start der Überwachung für ein Objekt, Änderung des Auslösedatum-Attributs oder -Skripts) berechnet. Falls ein Skript zur Ermittlung des Auslösedatums für überwachte Objekte benutzt wird, kann es - je nachdem wie das Skript seinen Wert bestimmt - sein, dass sich der Wert anderweitig, z.B. zeitabhängig, ändert. In solchen Fällen kann diese Änderung vom Alarm nicht registriert werden und es wird weiter das bestehende Auslösedatum für das Objekt benutzt!

Im Skript stehen folgende vordefinierte Variablen zur Verfügung:

bo

Das Objekt, für welches der Auslösezeitpunkt bestimmt werden soll.

bbt

Der BO-basierte Termin, zu dem das Skript gehört.

log

Ein Logger-Objekt (Name "de.ipcon.db.core.BOBasierterTermin") mit dem Debug- und andere Meldungen ins Server-Log ausgegeben werden können.

Wie auch bei EinfachenTerminen kann auch bei BO-basierten Terminen Außerdem noch unter "... aber sende Benachrichtigungen ... früher (Vorwarnzeit)" eine Vorwarnzeit angegeben werden, die die entsprechende Alarmauslösung dann noch früher stattfinden lässt.

Wer soll Benachrichtigungen erhalten und wie sollen diese aussehen?

Die Konfiguration für die Benachrichtigungen funktioniert hier genauso wie bereits für einfache

Termine beschrieben.

Automatische Neuterminierung nach Auslösung

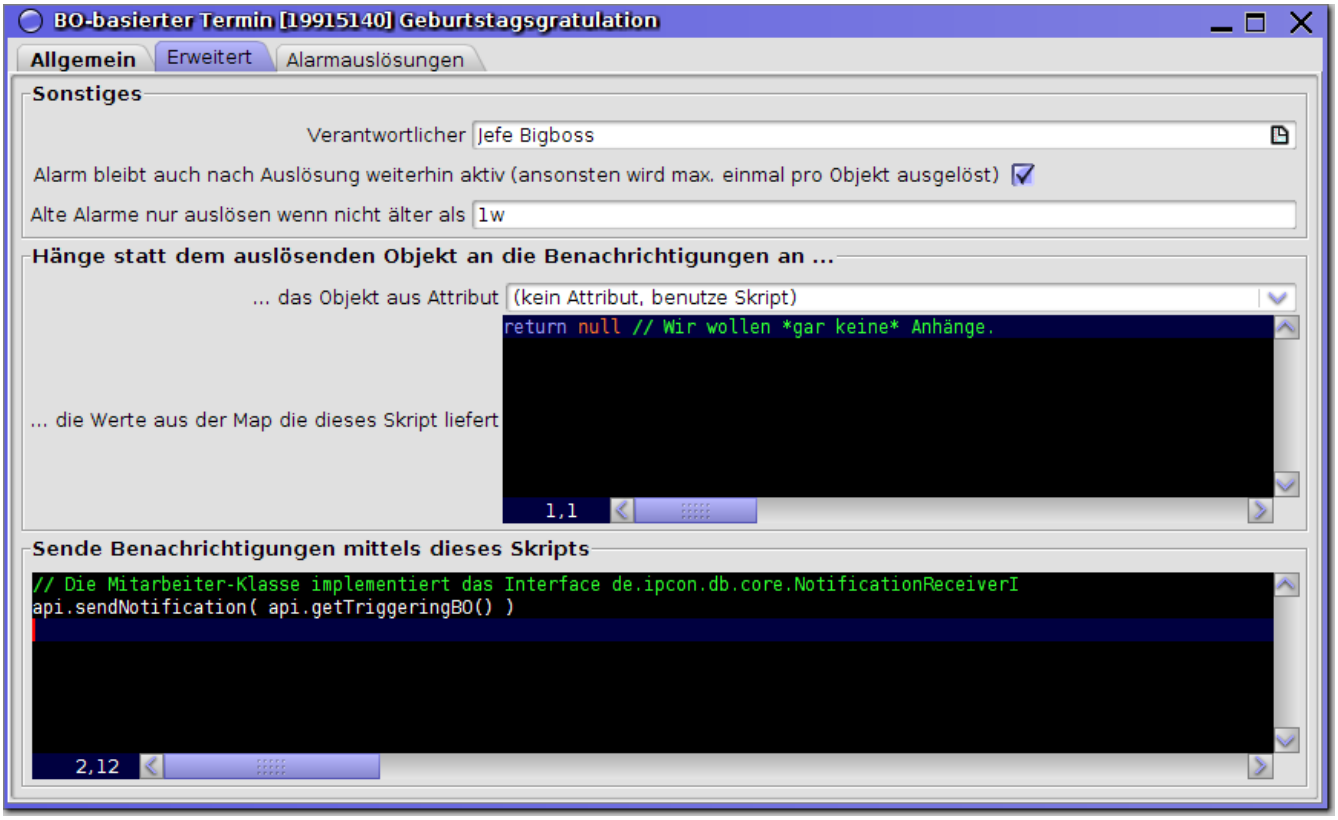
Normalerweise wird, analog zu den EinfachenTerminen, auch bei den BO-basierten Terminen für jedes überwachte Objekt nur ein einziges Mal ein Alarm ausgelöst. In gewissen Fällen kann es aber sinnvoll bzw. möglich sein, dass für ein Objekt der Alarm mehrfach zu verschiedenen Zeitpunkten ausgelöst werden kann und soll.

Durch Setzen von "Alarm bleibt auch nach Auslösung weiterhin aktiv" (auf dem Reiter "Erweitert") kann bestimmt werden, dass nach der Auslösung des Alarms für ein Objekt das Skript erneut aufgerufen bzw. das angegebene Attribut des Objekts erneut ausgelesen wird um sofort einen neuen Auslösezeitpunkt festzulegen, an dem dann der Alarm für dieses Objekt erneut ausgelöst werden soll.

Hierbei ist allerdings zu beachten, dass der Alarm für dieses Objekt nur dann wieder neu eingeplant wird, wenn hierbei dann ein Datum zurückliefert wird, welches *in der Zukunft* d.h. *nach* dem aktuellen Auslösezeitpunkt liegt. Ansonsten könnte es zu Problemen kommen, da der Alarm dann ohne Unterbrechung direkt hintereinander immer wieder ausgelöst würde.

Sollte das neue Datum ungültig sein (in der Vergangenheit liegen), so wird der Alarm für das aktuelle Objekt nicht mehr neu terminiert und in Zukunft nicht mehr ausgelöst.

Wenn Sie die Variante mit Attribut verwenden, macht diese Funktion normalerweise keinen Sinn, da ja immer nur *ein* Datum (welches dann nach der Auslösung garantiert in der Vergangenheit liegt) zurückgeliefert wird. Eine Ausnahme wäre, falls es sich um ein virtuelles Attribut handelt, da diese ja normalerweise ebenfalls berechnete Werte zurückliefern. Dies ist jedoch eher ein Thema für Fortgeschrittene und wird deshalb hier nicht weiter behandelt.



Anhängen von (weiteren) Objekten

Normalerweise wird an die Benachrichtigungen von Alarmen das Objekt, aufgrund welcher der Alarm ausgelöst wurde, angehängt. Es ist jedoch auch möglich, nicht das Objekt selbst, sondern ein von diesem Objekt referenziertes anderes Objekt oder gar beliebige Objekte stattdessen anzuhängen.

Im Feld "... das Objekt aus Attribut" können Sie ein Attribut des eigentlichen Objekts wählen, dessen Wert statt dem auslösenden Objekt angehängt werden soll. Es werden nur Relationen-Attribute angezeigt, d.h. keine Attribute die nur einfache Zahlen oder Zeichenketten, etc. als Werte beinhalten.

Wenn Sie mehrere oder andere Objekte anhängen möchten, wählen Sie hier "(kein Attribut, benutze Skript)" und können dann im darunter angezeigten Feld ein Skript eingeben, welches die anzuhängenden Objekte zusammenstellt und als eine Sammlung vom Java-Typ `Map` zurückliefert.

Skript das mehrere Werte an Benachrichtigungen anhängt:

```
// FIXME Vermutlich kürzere Alternative:
// ['Objekt selbst':bo, 'Mitarbeiter':bo.getMitarbeiter(), 'Personeneintrag des
Mitarbeiters':bo.getMitarbeiter().getPerson()] as Map
map = new HashMap()
map.put("Objekt selbst", bo)
map.put("Mitarbeiter", bo.getMitarbeiter())
map.put("Personeneintrag des Mitarbeiters", bo.getMitarbeiter().getPerson())
return map
```

Wenn Sie ein Attribut auswählen, oder ein Skript angeben, wird das eigentliche auslösende Objekt nicht mehr angehängt.

Wenn Sie an eine Benachrichtigung *gar keine* Objekte anhängen wollen, dann benutzen Sie ein Skript, welches `null` oder eine leere `Map` zurückgibt.

Gar nichts anhängen:

```
return null
```

Es stehen folgende vordefinierte Variablen zur Verfügung:

alarm

Der Alarm, der ausgelöst wurde.

dateNow

Das Datum und die Zeit (als `java.util.Date`-Objekt) wann der Alarm ausgelöst wurde.

log

Ein Logger-Objekt (Name "de.ipcon.db.alarm.AlarmNotificationManager") mit dem Debug- und andere Meldungen ins Server-Log ausgegeben werden können.

Für BO-basierte Termine, Hinweise und Wiedervorlagen, die sich ja immer auf BOs beziehen, stehen noch zwei zusätzliche Variablen zur Verfügung:

bo

Das Objekt (`BO`), welches erstellt/geändert/gelöscht wurde (kann null sein).

bot

Der BOT des BOs, für welches der Alarm ausgelöst wurde (kann evtl. null sein).

Für `Hinweise`, die ja immer durch ein Ereignis ausgelöst werden, ist schlussendlich noch eine Variable definiert:

bt

Die `BT`, welche den Alarm ausgelöst hat.

BOBasierterTermin-Status

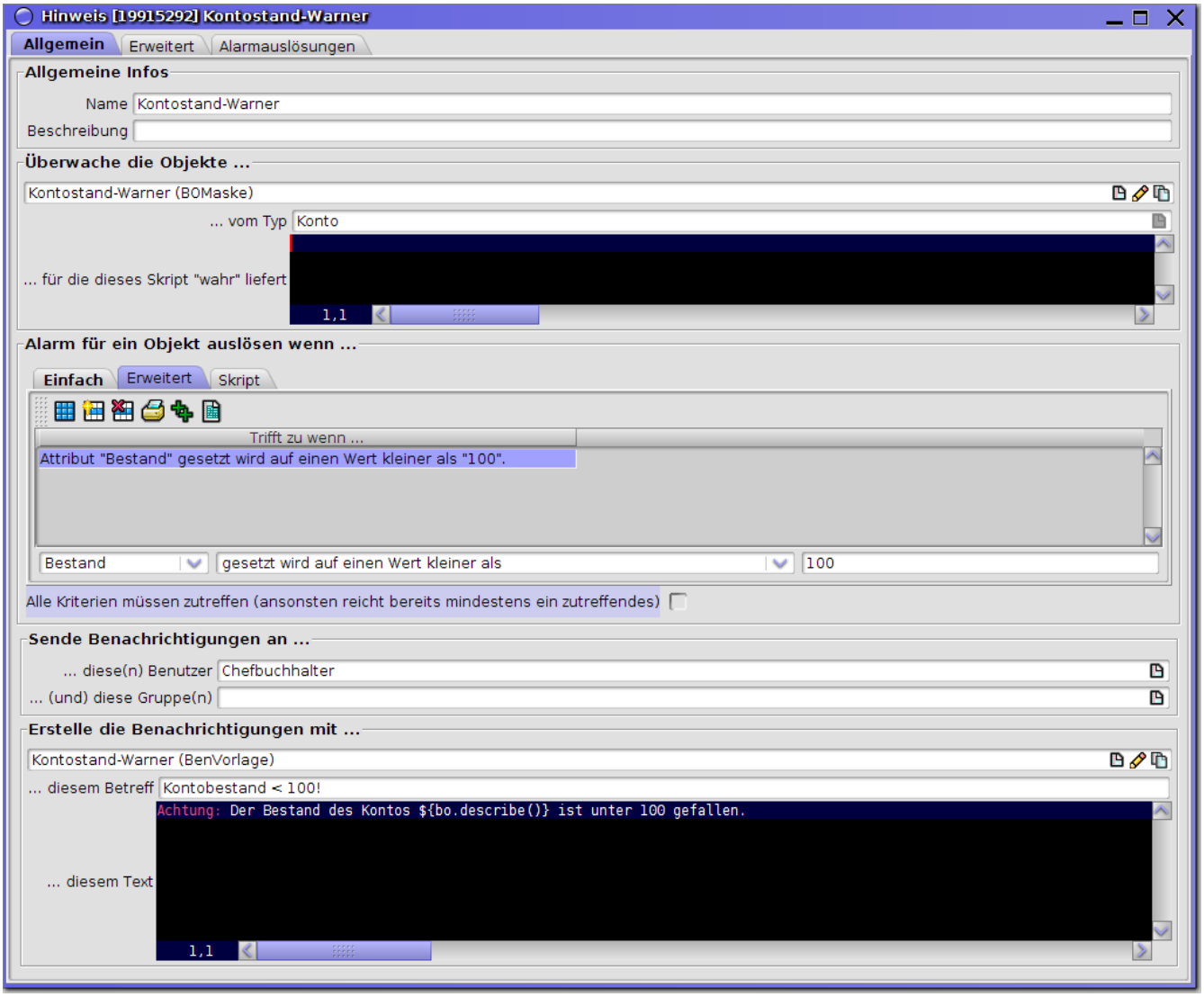
Wenn ein neuer BOBasierterTermin angelegt wird, legt MyTISM automatisch für alle überwachten Objekte sogenannte `BOBasierterTermin-Status` an, welche Daten beinhalten, die für die korrekte Überwachung und Auslösung des Alarms für die Objekte benötigt werden.

Je nach Anzahl der zu überwachenden Objekte, kann das Anlegen der Status einige Zeit in Anspruch nehmen. Der BOBasierteTermin wird erst dann aktiv, wenn alle benötigten Status angelegt wurden.

Hinweise

Hinweise sind dazu gedacht, eine Menge von Objekten zu überwachen und Benachrichtigungen zu versenden, wenn an einem oder mehreren dieser Objekte bestimmte Änderungen durchgeführt, bzw. solche Objekte erzeugt oder gelöscht wurden.

> *Beispiel:* Der Chef der Buchhaltung möchte benachrichtigt werden, sobald der Bestand eines Kontos unter 100,- EUR sinkt.



Allgemeine Eigenschaften festlegen

Geben Sie dem Hinweis einen kurzen aber aussagekräftigen Namen, und ggf. wenn sinnvoll eine längere Beschreibung.

"Alte Alarme nur auslösen wenn nicht älter als" und "Verantwortlicher" können Sie, bei Bedarf, auf dem Reiter "Erweitert" angeben.

Welche Objekte sollen "überwacht" werden?

Die Menge der zu "überwachenden" Objekte wird hier genau so wie für BO-basierte Termine definiert.

Ein Objekt muss den hier definierten Kriterien entsprechen, *nachdem* die gewünschte Änderung eingetreten ist. FIXME! Aktuell kann man keinen Hinweis definieren, der intuitiv so funktioniert, wie in obigem Beispiel angegeben. Man kann zwar definieren, dass der Bestand auf einen Wert < 100 gesetzt wird, das springt aber dann **immer** an, da man nicht definieren kann "aber nur für Konten, bei denen der Bestand (vorher) >= 100 war" :-/ Siehe auch Ticket 103771604.

Wann soll der Hinweis ausgelöst werden?

Nachdem definiert wurde, welche Objekte überwacht werden sollen, muss festgelegt werden, welche Ereignisse, z.B. Änderungen an diesen Objekten, den Hinweis auslösen sollen.

Hierzu gibt es verschiedene Möglichkeiten, die im Folgenden beschrieben werden.

Aus technischen Gründen kann nicht garantiert werden, wie schnell auf die gewünschte Änderung reagiert wird. Die Auslösung erfolgt in der Regel zwar wenige Sekunden, nachdem das Ereignis eingetreten ist, aber die genaue Reaktionszeit ist unbestimmt. Insbesondere eine sofortige Reaktion in Echtzeit ist praktisch nicht möglich; wird die schnellstmögliche Reaktion auf Ereignisse benötigt, sollte die Behandlung in einer `verifyOnServer()`-Methode implementiert werden (fortgeschrittenes Thema, Zugang zum Quellcode der Applikation ist dafür erforderlich).

Ignorierte BTs/Änderungen

BedingteAlarme (Hinweise, aber auch BOBasierteTermine, Wiedervorlagen) ignorieren einige BTs/Änderungen, die vom Alarmsystem selber durchgeführt wurden, da es ansonsten zu Endlosschleifen kommen könnte. Dies sind

- Das Speichern von neu angelegten AlarmAusloesung-Einträgen
- Das Versenden der Standardbenachrichtigungen durch Alarme
- Die Deaktivierung von Alarmen aufgrund von Fehlern
- Änderungen aufgrund der Ausführung des "Speichern-Fehler-Skripts"
- Das Speichern der auslösenden Transaktion and von aufgetretenen Fehlern am AlarmAusloesung-Objekt

Die beteiligten Entitäten sind `Alarm` (inkl. aller Subklassen) sowie `AlarmAusloesung` (inkl. aller Subklassen) sowie `MyTISMBenachrichtigung` und `MyTISMBenachrichtigungsauftrag` (inkl. aller Subklassen). Für diese Entitäten können also nicht alle Änderungen überwacht werden (wobei die Anwendungsfälle dafür im Normalfall auch sehr beschränkt sein sollten).

BTs/Änderungen die anderweitig aufgrund der Auslösung eines Alarms erzeugt wurden - insb. durch das beim Auslösen ggf. ausgeführte Skript - können dagegen von anderen Alarmen "erkannt" und darauf reagiert werden.

Auslösung bei beliebiger Änderung, Erstellen oder Löschen von Objekten (Unter-Reiter "Einfach")

Diese oft benutzten, einfachen Fälle können einfach durch Aktivieren der entsprechenden Checkbox definiert werden:

- *"... ein überwachtes Objekt erzeugt wurde"* löst den Hinweis aus, sobald eines oder mehrer neue Objekte, auf die die für den Hinweis definierte Maske (s.o.) passt, erstellt wurden.
- *"... ein überwachtes Objekt geändert wurde"* löst den Hinweis aus, sobald an einem oder mehreren der überwachten Objekte *irgendeine* Änderung durchgeführt wurde; sei es z.B., dass der Wert eines Attributes gesetzt, gelöscht oder geändert wurde oder ein Objekt in einer Relation hinzugefügt oder gelöscht wurde.
- *"... ein überwachtes Objekt gelöscht wurde"* löst den Hinweis aus, sobald eines oder mehrere der überwachten Objekte gelöscht wurden.
- *"... ein überwachtes Objekt erschienen ist"* löst den Hinweis aus, sobald ein *existierendes* Objekt so geändert wurde, dass es jetzt in die Menge der vom Alarm überwachten Objekte passt.
- *"... ein überwachtes Objekt verschwunden ist"* löst den Hinweis aus, sobald ein *existierendes und vom Alarm überwachtes* Objekt so geändert wurde, dass es jetzt *nicht* mehr in die Menge der vom Alarm überwachten Objekte passt.

Auslösung mittels Auslösekriterien (Unter-Reiter "Erweitert")

Mit den sog. Auslösekriterien gibt es eine recht einfache aber sehr flexible Möglichkeit, festzulegen, welche Änderungen erfolgt sein müssen, damit der Hinweis ausgelöst wird.

Mittels der Auslösekriterien geben Sie an, für welche Attribute ("Felder" oder "Eigenschaften" der Objekte) welche Änderungen oder Ereignisse eingetreten sein müssen, damit der Hinweis für das Objekt ausgelöst wird. Sie können für jeden Hinweis beliebig viele Auslösekriterien festlegen.
Jedes dieser Auslösekriterien hat drei wichtige Eigenschaften:

Attribut (erstes Feld)

Hiermit definieren Sie, *an welchem Attribut* der überwachten Objekte eine Änderung erfolgt sein muss. Die Einträge in der Auswahlbox geben alle Attribute an, welche für die vom Alarm überwachten Objekte verfügbar sind - mit Ausnahme von virtuellen und nicht-persistenten (weil diese aus technischen Gründen hier nicht geprüft werden können) und System-Attributen.

Änderungstyp (zweites Feld)

Hiermit definieren Sie, *wie* sich das oben angegebene Attribut verändert haben muss, damit der Hinweis ausgelöst wird. Je nach Typ des ausgewählten Attributs werden hier nur passende Änderungstypen aufgeführt.

Wert (drittes Feld, ist ausgeblendet wenn nicht anwendbar)

Manche Änderungstypen, wie z.B. "wird gesetzt auf den Wert", erfordern einen Vergleichswert;

diesen können sie hier angeben. Wenn ein Änderungstyp keinen Vergleichswert erfordert, wird dieses Feld automatisch ausgeblendet.
Die Werte können so eingegeben werden, wie Sie sie auch normalerweise in anderen MyTISM-Formularen angeben. Für Datums- und Wahrheitswerte wird ebenfalls ein passendes Eingabefeld angezeigt. Für Relationen (also Attribute die Verweise/Links auf ein oder mehrere andere Objekte abbilden) können Sie das gewünschte Vergleichsobjekt mittels Popup auswählen. Allerdings ist die Unterstützung für Relationen hier noch lückenhaft, so kann z.Zt. z.B. noch nicht geprüft werden, ob ein Objekt zu einer Mehrfach-Relation hinzugefügt oder entfernt wurde; dies ist z.Zt. nur mit einem Skript (s.u.) möglich.

Wenn Sie informiert werden wollen, wenn der *Kontostand* Ihres Kontos *unter 100*,- EUR gesunken ist, setzen Sie *Attribut* (Feld 1) auf *Kontostand*, *Änderungstyp* (Feld 2) auf *"wird gesetzt auf Wert kleiner als"* und *Wert* (Feld 3) auf *100*.

Wenn Sie über *jede* Änderung Ihres Kontostandes informiert werden wollen, setzen sie *Änderungstyp* (Feld 2) auf *"wird in irgendeiner Weise geändert"*; in diesem Fall brauchen Sie keinen Vergleichswert anzugeben und das *Wert*-Feld wird automatisch ausgeblendet.

Wenn Ihnen die vordefinierten Möglichkeiten, z.B. die verfügbaren Vergleichsmöglichkeiten der Änderungstypen, nicht ausreichen, steht Ihnen noch die Möglichkeit zur Verfügung, mittels eines eigenen Skript praktisch jeden beliebigen Vergleich zu realisieren, auch wenn hierzu ein paar Kenntnisse in Skript-Programmierung und etwas Wissen über die internen Abläufe in einer MyTISM-Anwendung nötig sind.

Das Skript wird für jeden Transaktionsschritt (BP) der aktuellen Transaktion (BT), in der das angegebene Attribut gesetzt, gelöscht oder geändert wurde, einmal ausgeführt. Wenn das Skript für *mindestens eine* der BPs "true" zurückliefert, gilt das Auslösekriterium als erfüllt; wenn es für *alle* BPs nur "false" liefert als "nicht erfüllt".

Im Skript stehen folgende vordefinierte Variablen zur Verfügung:

bp

Das BP-Objekt, welches gerade überprüft wird.

bo

Das Objekt (BO), welches erstellt/geändert/gelöscht wurde (kann null sein).

valueNew

Der neue/gesetzte Wert, aus dem BP-Objekt (als Java-Objekt! Kann null sein).

valueOld

Der alte/vorher gesetzte Wert, aus dem BP-Objekt (als Java-Objekt! Kann null sein).

valueCompare

Der von Ihnen eingegebene (Vergleichs)Wert (bereits umgewandelt in Java-Objekt! Kann null sein wenn Sie keinen Wert eingegeben haben bzw. der gewählte Änderungstyp keinen

Vergleichswert erfordert und das Feld ausgeblendet war).

schema

Das SchemaI für die aktuelle MyTISM-Installation.

attribute

Das beim Auslösekriterium angegebene `AttributeI` (nicht der Name, sondern das Java-Objekt! Kann null sein).

type

Der `CBOType` des beim Auslösekriterium angegebenen Attributes (kann null sein).

kriterium

Das `AusloeseKriterium`-Objekt (wird eher selten benötigt).

log

Ein Logger-Objekt (Name "de.ipcon.db.core.AusloeseKriterium") mit dem Debug- und andere Meldungen ins Server-Log ausgegeben werden können.

Bitte verwechseln Sie dieses Skript nicht mit der unten erwähnten Möglichkeit eines "globalen" Auslöseskript für den gesamten Hinweis. Das oben beschriebene Skript stellt nur eine Option dar, weitere Vergleichsmöglichkeiten für Auslösekriterien zu realisieren. Es ist nur ein Teil dieses einzelnen Auslösekriteriums und bezieht sich immer nur auf Änderungen an einem einzelnen Attribut.

Auslösung mittels Auslöseskript (Unter-Reiter "Skript")

Ein Auslöseskript gibt Ihnen vollkommene Freiheit, um die Auslösung eines Hinweises zu bestimmen; um dieses Feature benutzen zu können, müssen sie allerdings über gewisse Kenntnisse in Skript-Programmierung und etwas Wissen über die interne Struktur und Abläufe in MyTISM-Anwendungen verfügen.

Mittels eines Auslöseskripts können Sie in jeder von Ihnen gewünschten Art und Weise überprüfen, ob der Hinweis ausgelöst werden soll, oder nicht. Wenn das Skript "true" zurückliefert, wird der Hinweis ausgelöst; bei "false" nicht.

```
if (bo.PreisInCentNN.intValue() > bo.MaxPreisInCentNN.intValue())
   if (!bo.PreisueberschreitungErlaubtNN.booleanValue())
      return true
return false
```

Im Skript stehen folgende vordefinierte Variablen zur Verfügung:

bo

Das Objekt (`BO`) welches erstellt/geändert/gelöscht wurde.

schema

Das SchemaI für die aktuelle MyTISM-Installation.

bp

Das `BP`-Objekt, welches gerade überprüft wird.

kriterium

Das `AusloeseKriterium`-Objekt (eher uninteressant).

log

Ein Logger-Objekt (Name "de.ipcon.db.core.AusloeseKriterium") mit dem Debug- und andere Meldungen ins Server-Log ausgegeben werden können.

Das Skript wird für jeden Transaktionsschritt (`BP`) einmal ausgeführt, d.h. beim Speichern eines Formulars im Normalfall mehrmals, wenn sich mehrere Werte geändert haben. Es reicht in diesem Fall, wenn das Skript mindestens einmal "true" zurückliefert, um die Auslösung des Hinweises zu veranlassen.

Mindestens eines oder alle gleichzeitig?

Wenn Sie mehrere Kriterien für die Auslösung des Hinweises angeben - also "... ein überwachtes Objekt erzeugt wurde", "... ein überwachtes Objekt geändert wurde", "... ein überwachtes Objekt gelöscht wurde", ggf. Auslösekriterien, ggf. ein Auslöseskript - so wird der Hinweis normalerweise bereits ausgelöst, wenn *mindestens eines* dieser Kriterien zutrifft (die Kriterien sind mit "oder" verknüpft).

Möchten Sie, dass *alle* Kriterien gleichzeitig zutreffen müssen, damit die Auslösung erfolgt, so aktivieren sie die Checkbox "Alle Kriterien müssen zutreffen".

Sie haben zwei AuslöseKriterien definiert: * Kontostand, "wird gesetzt auf einen Wert kleiner als", 100 und * Kontostand, "wird gesetzt auf einen Wert größer als", 50 .

Im Normalfall würde der Hinweis *immer* auslösen, wenn sich der Kontostand ändert, da jede Zahl entweder kleiner als 100 oder größer als 50 ist. Wenn Sie aber "Alle Kriterien müssen zutreffen" setzen, müssen *beide* Kriterien zutreffen und der Hinweis wird nur ausgelöst, wenn der Kontostand auf einen Wert größer als 50 *und* kleiner als 100 - also z.B. auf 80 - gesetzt wird.

Wer soll Benachrichtigungen erhalten und wie sollen diese aussehen?

Die Konfiguration für die Benachrichtigungen funktioniert hier genauso wie bereits für einfache Termine beschrieben.

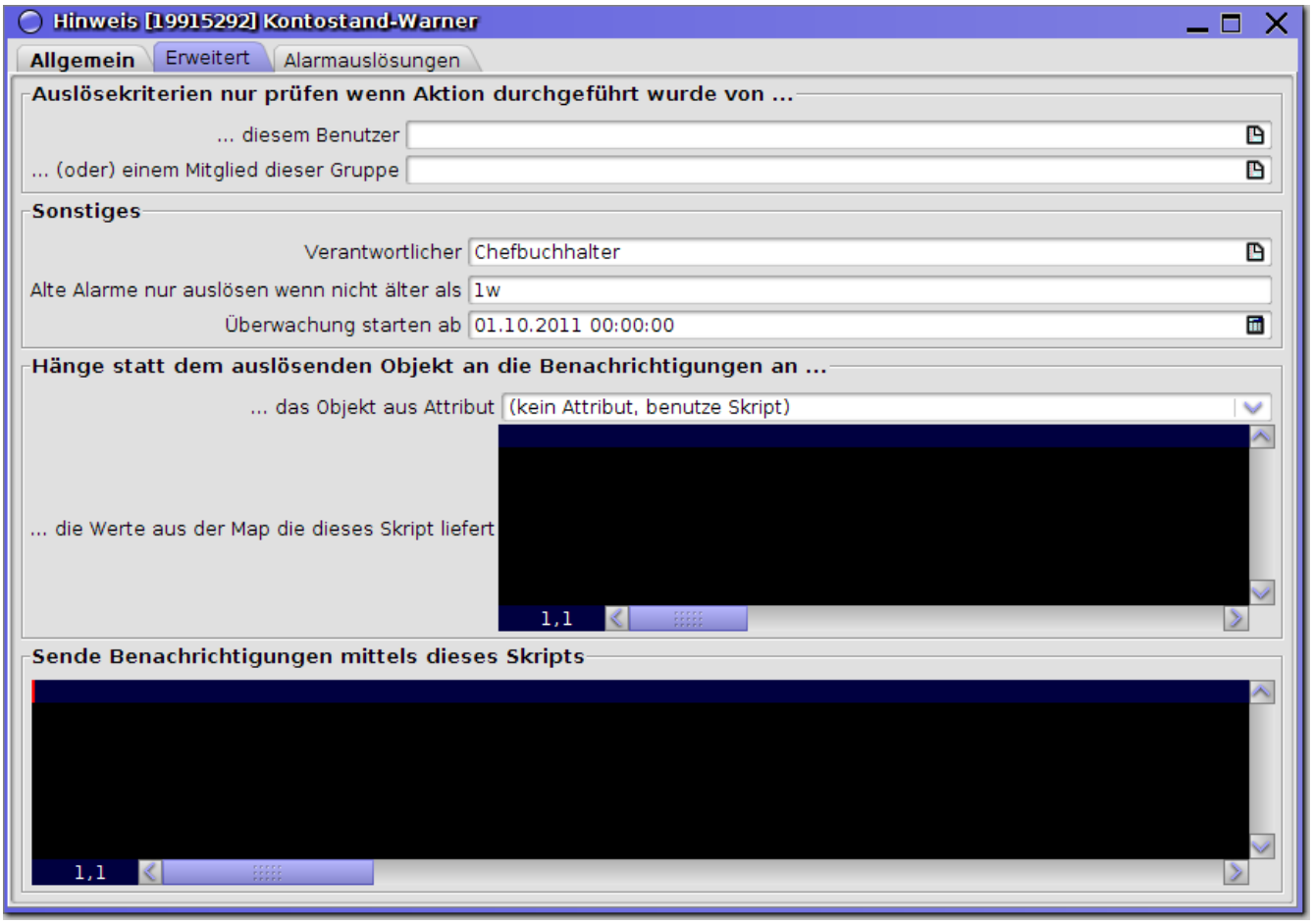
Von wem muss die Änderung stammen?

Wenn der Hinweis nur ausgelöst werden soll, wenn eine Änderung von einem bestimmten Benutzer bzw. einem Mitglied einer bestimmten Gruppe durchgeführt wurde, können Sie dies mit "... diesem Benutzer" und "... (oder) einem Mitglied dieser Gruppe" angeben.

Wenn Sie für beides einen Wert eintragen, reicht es aus, wenn die Änderung von dem Benutzer

oder einem Mitglied der Gruppe gemacht wurde; es ist also *nicht* erforderlich, dass der angegebene Benutzer auch noch Mitglied der angegebenen Gruppe ist.

Wenn Sie für eines der beiden Kriterien keine Angabe machen, wird nur für das andere Kriterium überprüft, ob die Änderung von diesem Benutzer bzw. dieser Gruppe gemacht wurde (was dann zutreffen muss, damit die Änderung "gewertet" wird). Wenn Sie für beide Eigenschaften keinen Wert angegeben haben, ist vollkommen egal, von welchem Benutzer oder welcher Gruppe die Änderung stammt und die sonstigen Auslösekriterien werden immer überprüft.



Ab wann ist der Hinweis aktiv?

Wie bereits früher erwähnt werden Alarme normalerweise sofort aktiv, sobald sie das erste Mal gespeichert werden. Hinweise beginnen also direkt nachdem sie erzeugt wurden, die ihnen zugewiesene Menge von Objekten zu überwachen und bei Eintreten der durch ihre AenderungsKriterien definierten Änderungen Alarme auszulösen.

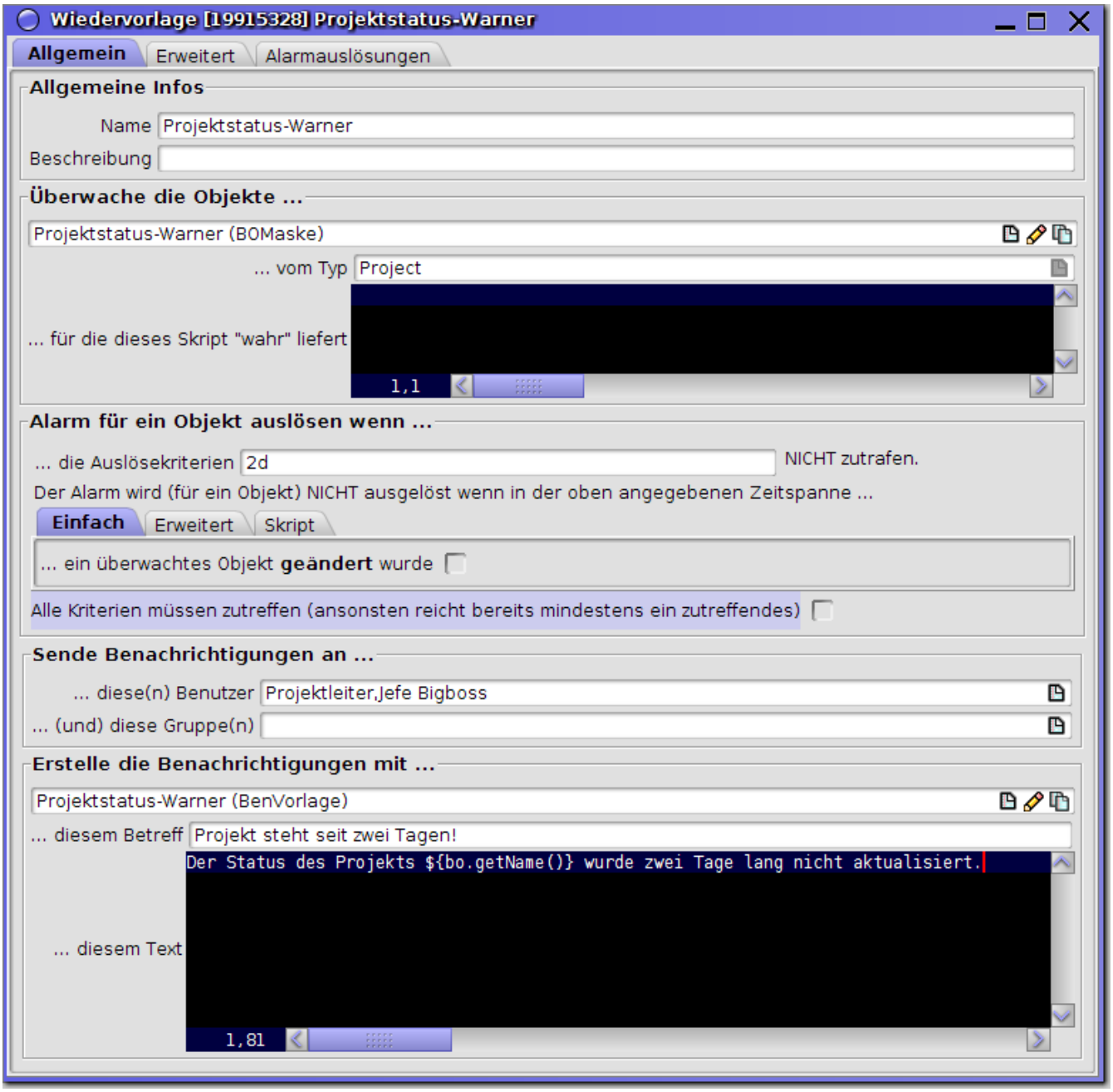
Es ist jedoch auch möglich anzugeben, dass der Hinweis erst zu einem späteren Zeitpunkt aktiv wird. Bei "Überwachung starten ab" (Reiter "Erweitert") können Sie ein Datum angeben, ab welchem der Hinweis aktiv werden soll. Liegt dieses Datum in der Zukunft, werden erst die ab diesem Datum erfolgenden Änderungen überprüft und für die Auslösung berücksichtigt. Wenn Sie hier keinen Wert angeben oder das angegebene Datum in der Vergangenheit liegt, wird der Hinweis ganz normal sofort aktiv.

Wiedervorlagen

Evtl. sind Wiedervorlagen nicht genau das, was Sie benötigen bzw. was Sie sich darunter vorstellen. Wenn Sie z.B. in Ihrer Anwendung `Dokumente` mit einem Attribut `WiedervorlageAm` haben, und Sie möchten, dass jeweils an den dort eingetragenen Daten ein Alarm ausgelöst bzw. eine Benachrichtigung verschickt wird, dann können Sie das mit einem BoBasierterTermin realisieren.

Wiedervorlagen sind sozusagen das Gegenteil der Hinweise. Alarme werden hier ausgelöst, wenn innerhalb einer bestimmten, festgelegten Zeitspanne an einer Menge von überwachten Objekten bestimmte Änderungen *nicht* durchgeführt wurden.

Beispiel: Der Projektleiter möchte benachrichtigt werden, wenn sich der Status eines Projekts zwei Tage lang *nicht* geändert hat.



Allgemeine Eigenschaften festlegen

Geben Sie der Wiedervorlage einen kurzen aber aussagekräftigen Namen, und ggf. wenn sinnvoll eine längere Beschreibung.

"Alte Alarme nur auslösen wenn nicht älter als" und "Verantwortlicher" können Sie, bei Bedarf, auf dem Reiter "Erweitert" angeben.

Welche Objekte sollen "überwacht" werden?

Die Menge der zu "überwachenden" Objekte wird hier genau so wie für BO-basierte Termine definiert.

Wann soll die Wiedervorlage ausgelöst werden?

Nachdem definiert wurde, welche Objekte überwacht werden sollen, muss festgelegt werden, welche Ereignisse, z.B. Änderungen an diesen Objekten, *verhindern sollen*, dass die Wiedervorlage ausgelöst wird.

Die Definition der Kriterien, die geprüft werden, erfolgt hier genau so wie für die Hinweise.

Nachdem die Wiedervorlage angelegt wurde, wartet sie für jedes der von ihr "überwachten" Objekte eine festgelegte Zeit.

- Tritt innerhalb dieser Zeitspanne das gewünschte (durch die Kriterien definierte) Ereignis ein oder wird die gewünschte (durch die Kriterien definierte) Änderung an einem Objekt durchgeführt, wurde damit die Auslösung der Wiedervorlage für das betreffende Objekt verhindert und die Überwachung für dieses Objekt wird beendet (außer wenn *"Alarm bleibt auch nach Kriterienerfüllung weiterhin aktiv"* gesetzt ist).
- Bleibt jedoch das gewünschte Ereignis innerhalb dieser Zeitspanne *aus* oder tritt die gewünschte Änderung für ein Objekt *nicht* ein, so löst die Wiedervorlage am Ende der Zeitspanne für das betreffende Objekt Alarm aus. Dann beendet sie die Überwachung für das betreffende Objekt (außer wenn *"Alarm bleibt auch nach Auslösung weiterhin aktiv"* gesetzt ist).

Wie oben erwähnt, wird das genaue Verhalten der Wiedervorlage mittels dreier Einstellungen definiert:

Inaktivitätszeit "... die Auslösekriterien ... NICHT zutrafen."

Dies gibt die Zeitspanne an, welche die Wiedervorlage auf das Eintreten der Ereignisse bzw. Änderungen warten soll.

Beispiel: Wollen Sie benachrichtigt werden, wenn sich an einem Projekt zwei Tage nichts getan hat, so geben Sie hier "2d" an.

Neuterminierung nach Aufschub "Alarm bleibt auch nach Kriterienerfüllung weiterhin aktiv", Reiter "Erweitert"

Normalerweise wird die Überwachung eines Objektes beendet, nachdem die definierte Änderung für dieses Objekt innerhalb der Inaktivitätszeit eingetreten ist und die Wiedervorlage für dieses Objekt damit verhindert wurde.
Wollen Sie jedoch, dass die Überwachung auch weiter fortgeführt wird, obwohl das gewünschte Ereignis oder die gewünschte Änderung einmal eingetreten ist, so setzen sie dieses Flag. Wenn das Flag gesetzt ist, heißt das im Endeffekt, dass das gewünschte Ereignis oder die gewünschte Änderung regelmäßig immer wieder eintreten muss, um zu verhindern, dass die Wiedervorlage letztendlich ausgelöst wird.

Beispiel: Der Fertigstellungsstand eines Projektes muss mindestens einmal jeden Tag aktualisiert werden.

Neuterminierung nach Auslösung "Alarm bleibt auch nach Auslösung weiterhin aktiv", Reiter "Erweitert"

Normalerweise wird die Überwachung eines Objektes ebenfalls beendet, nachdem die definierte Änderung für dieses Objekt innerhalb der Inaktivitätszeit nicht eingetreten ist und die Wiedervorlage für dieses Objekt ausgelöst wurde, d.h. für jedes überwachte Objekt löst die Wiedervorlage nur ein einziges Mal einen Alarm aus (es gibt gewisse Ausnahmen, s.u.).
Wollen Sie jedoch, dass die Überwachung auch danach weiter fortgeführt wird, so setzen sie dieses Flag. Wenn das Flag gesetzt ist, heißt das im Endeffekt, dass regelmäßig wieder nach erneutem Ablauf der Inaktivitätszeit ein Alarm für das betreffende Objekt ausgelöst wird.

Beispiel: Wenn eine Rechnung nicht innerhalb eines Tages bezahlt wurde, soll an jedem folgenden Tag eine Benachrichtigung darüber versandt werden, nicht nur einmal.

Wer soll Benachrichtigungen erhalten und wie sollen diese aussehen?

Die Konfiguration für die Benachrichtigungen funktioniert hier genauso wie bereits für einfache Termine beschrieben.

Wiedervorlage-Status

Wenn eine neue Wiedervorlage angelegt wird, legt MyTISM automatisch für alle überwachten Objekte sogenannte `Wiedervorlage-Status` an, welche Daten beinhalten, die für die korrekte Überwachung und Auslösung des Alarms für die Objekte benötigt werden.

Je nach Anzahl der zu überwachenden Objekte, kann das Anlegen der Status einige Zeit in Anspruch nehmen. Die Wiedervorlage wird erst dann aktiv, wenn alle benötigten Status angelegt wurden.

Benachrichtigung bei Alarm-Auslösung

Wenn ein Alarm ausgelöst wird, werden nacheinander drei verschiedene Mechanismen in Gang gesetzt:

1. Die hartkodierte trigger()-Methode der Klasse, zu der der Alarm gehört, wird aufgerufen.
2. Ein evtl. für den Alarm eingetragenes Benachrichtigungsskript wird ausgeführt.
3. Die Standard-Benachrichtigungen werden ausgeführt.

Hartkodierte trigger()-Methode

Jede Subklasse der Alarm-Basisklasse `Alarm` erbt deren trigger()-Methode. Bei der Auslösung eines Alarms wird diese Methode automatisch vom Alarmsystem aufgerufen und kann beliebige Aktionen ausführen.

Wenn die Methode "true" zurückliefert, wird angenommen, dass alle bei der Auslösung erforderlichen bzw. gewünschten Aktionen vollständig durchgeführt wurden; in diesem Fall werden weder das Benachrichtigungsskript noch die Standard-Benachrichtigungen ausgeführt.

Bei allen standardmässig in MyTISM implementierten Alarm-Klassen (also den in dieser Dokumentation erwähnten) tut diese Methode nichts und liefert "false" zurück, so dass mit der Bearbeitung fortgefahren wird; sie brauchen sich in diesem Fall also keine weiteren Gedanken hierzu zu machen. Es kann allerdings sein, dass für Ihre MyTISM-Installation spezielle Subklassen von `Alarm` existieren, die eine "richtige" trigger()-Methode besitzen; wenn dies der Fall ist, kann Ihnen Ihr MyTISM-Administrator weitere Informationen hierzu geben.

Benachrichtigungsskript "Sende Benachrichtigungen mittels dieses Skripts", Reiter "Erweitert"

Wenn Ihnen die Standard-Möglichkeiten (s.u.) für Benachrichtigungen nicht ausreichen, können Sie mit Hilfe der Benachrichtigungsskript-Eigenschaft der Alarme weitere Tätigkeiten ausführen lassen.

Sie können hier ein Stück Groovy-Code angeben, das in der von Ihnen gewünschten Art und Weise Benachrichtigungen auslöst oder auch andere Aktionen ausführt.

Über die `api` Variable können transaktionelle Änderungen aufgezeichnet werden, die bei der Auslösung des Alarms automatisch gespeichert werden. Via `api.getTransaction()` bekommt man eine Transaction, um Objekte zu laden und diese zu includen, um Änderungen an diesen aufzuzeichnen. Ein Aufruf von `api.getBO(Long, Class)` initialisiert ebenfalls bereits eine neue Transaction, um das Objekt für die übergebene Id zu laden.

Ein erneuter Aufruf von `api.getTransaction` gibt immer wieder die gleiche Transaction zurück, da es nur eine im Kontext des Benachrichtigungsskripts gibt.

Bei Systemen, in denen Alarme auch auf Änderungen durch andere Alarme getriggert werden können, muss man sehr gut aufpassen, dass dabei keine unerwünschten Schleifen entstehen.

Wenn das Groovy-Skript "true" zurückliefert, wird angenommen, dass alle erforderlichen Aktionen durchgeführt wurden und die Standard-Benachrichtigungen werden *nicht mehr* ausgelöst. Wenn das Skript "false" zurückliefert, werden die Standard-Benachrichtigungen zusätzlich zu allen evtl. bereits vom Skript gemachten Aktionen auch noch ganz normal ausgelöst.

Sollte im Skript ein Fehler auftreten (d.h. eine Exception geworfen werden) wird die Bearbeitung ebenfalls abgebrochen, d.h. auch in diesem Fall werden die Standard-Benachrichtigungen *nicht mehr* ausgelöst.

Beispiel, Mitarbeiter implementiert NotificationReceiverI:

```
api.getLogger().info("Alarm " + alarm + " wurde um " + dateNow + " ausgeloest!")
api.sendNotification(api.getBOById(idBO, de.beispielprojekt.bo.Mitarbeiter.class))
```

Beispiel, Empfänger hängt per Attribut "Benutzer" an auslösendem Objekt:

```
api.sendNotification(getTriggeringBO().getBenutzer())
```

Beispiel, Mail an beliebige e-Mail-Adresse senden:

```
api.sendNotificationByEmail("nobody@example.com")
```

Beispiel, Mail senden und Mailversanddatum am auslösenden BO setzen:

```
import com.oashi.m.bo.Rechnung

def r = api.getTriggeringBO() as Rechnung
def tx = api.getTransaction("Mailversanddatum via Alarm setzen.")
r = tx.include(r)
r.Mailversanddatum = new Date()

return false // do send default notifications
```

Es stehen die folgenden vordefinierten Variablen zur Verfügung:

alarm

Der Alarm, der ausgelöst wurde.

dateNow

Das Datum und die Zeit (als `java.util.Date`-Objekt) wann der Alarm ausgelöst wurde.

api

Ein Objekt vom Typ `BedingterAlarmBenachrichtigungsScriptAPI` (für Hinweise und

Wiedervorlagen), BOBasierterTerminBenachrichtigungsScriptAPI (für BO-basierte Termine) oder EinfacherTerminBenachrichtigungsScriptAPI (für einfache Termine) welches nützliche Methoden zur Verfügung stellt.

log

Ein Logger-Objekt (Name "de.ipcon.db.alarm.BenachrichtigungsScriptAPI", das gleiche Objekt was auch api.getLogger() liefert) mit dem Debug- und andere Meldungen ins Server-Log ausgegeben werden können.

Für BO-basierte Termine, Hinweise und Wiedervorlagen, die sich ja immer auf BOs beziehen, stehen noch zwei zusätzliche Variablen zur Verfügung:

idBO

Der ID des BOs, für welches der Alarm ausgelöst wurde.

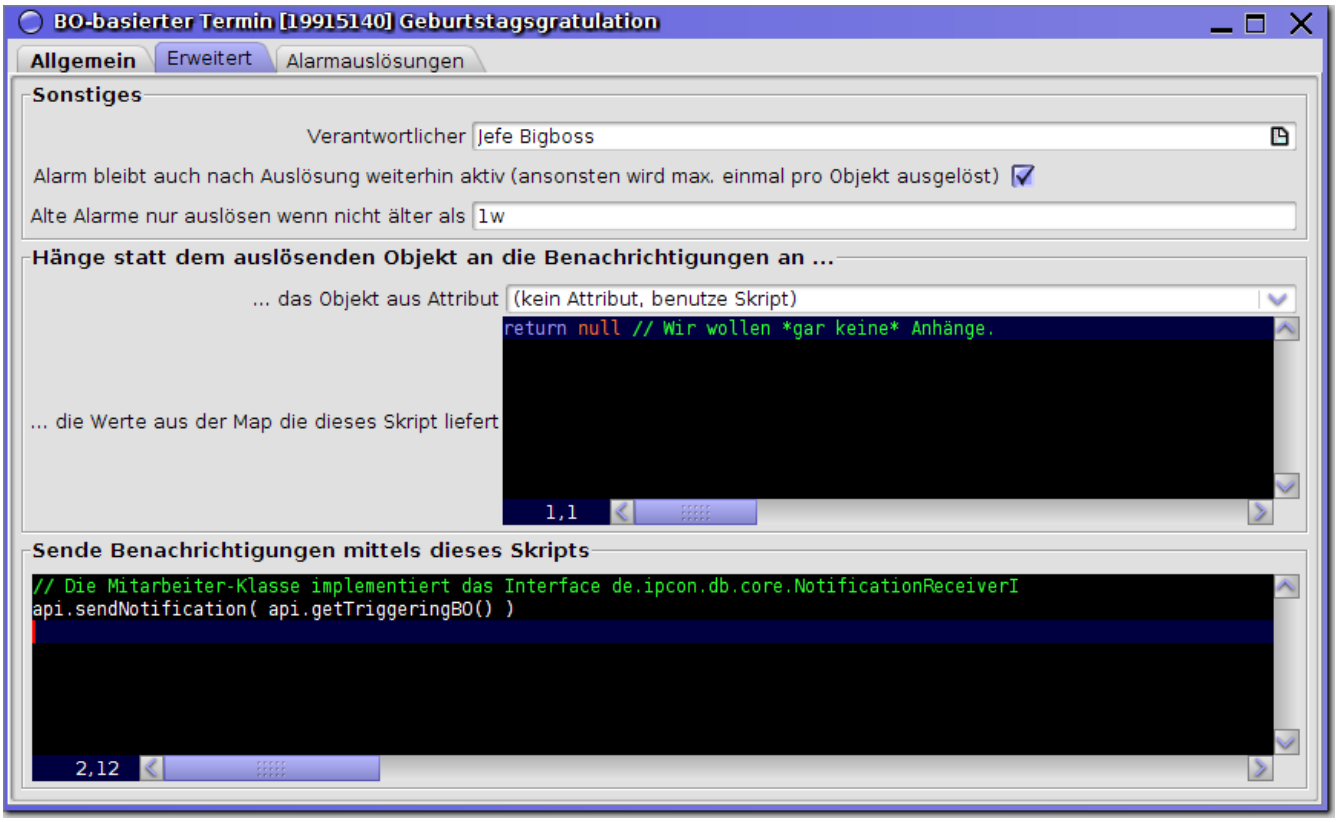
bot

Der BOT des BOs, für welches der Alarm ausgelöst wurde (kann evtl. null sein).

Für Hinweise und Wiedervorlagen, die ja immer durch ein Ereignis ausgelöst werden, ist schlussendlich noch eine Variable definiert:

bt

Die BT, welche den Alarm ausgelöst hat.



Standard-Mechanismus

Wenn ein Alarm ausgelöst wird, werden alle dafür eingetragenen Benutzer benachrichtigt (sofern das Benachrichtigungssystem aktiviert und richtig konfiguriert ist).

Ein Benutzer erhält für eine Alarm-Auslösung immer nur eine Benachrichtigung, auch wenn z.B. für einen Alarm mehrere Gruppen eingetragen wurden und der Benutzer Mitglied in mehreren dieser Gruppen ist oder der Benutzer selbst ebenfalls für den Alarm eingetragen wurde.

Der Betreff und der Text der entsprechenden Benachrichtigungen können für jeden Alarm eigens definiert werden. Die Definition der Textvorlagen erfolgt im Format GSP (Groovy Server Pages); die eigentlichen Texte werden bei der Auslösung dynamisch aus diesen Vorlagen generiert.

Eine genaue Beschreibung von GSP würde hier zu weit führen; für Informationen dazu siehe http://groovy.codehaus.org/Groovy+Templates. Hier nur ein kleines

Beispiel:

```
Dies ist eine Benachrichtigung fuer ${benutzer.getName()}!
Der Alarm ${alarm.getName()} wurde am ${api.formatDate(dateNow, "dd.MM.yyyy")} um
${api.formatDate(dateNow, "HH:mm:ss")} Uhr ausgeloest.
```

Folgende vordefinierte Variablen stehen zur Verfügung:

benutzer

Der Benutzer, für den die Benachrichtigung gedacht ist; kann null sein, wenn die Benachrichtigung mittels des Benachrichtigungsskripts erzeugt wurde und nicht an einen Benutzer, sondern ein anderes Objekt verschickt wurde.

empfaenger

Das Empfaenger-Objekt, für das die Benachrichtigung gedacht ist; ist immer gesetzt. Wenn die Benachrichtigung an einen Benutzer ging, ist dieser Wert gleich dem Wert der Variable "benutzer".

alarm

Der Alarm, der ausgelöst wurde.

dateNow

Das Datum und die Zeit (als `java.util.Date`-Objekt) wann der Alarm ausgelöst wurde (oder genauer: Wann das MyTISMBenachrichtigungs-Objekt erstellt wurde; diese Zeiten können sich um einige Sekunden unterscheiden).

api

Ein Objekt vom Typ `TemplateScriptAPI`, welches nützliche Methoden zur Verfügung stellt.

Für BO-basierte Termine, Hinweise und Wiedervorlagen, die sich ja immer auf BOs beziehen, steht noch eine zusätzliche Variable zur Verfügung:

bo

Das Objekt (`BO`) für welches der Alarm ausgelöst wurde.

Für Hinweise und Wiedervorlagen, die ja immer durch ein Ereignis ausgelöst werden, ist schlussendlich noch eine Variable definiert:

bt

Die BT, welche den Alarm ausgelöst hat.

Schliesslich können spezielle Unterklassen von Alarmen evtl. auch noch weitere Variablen zur Verfügung stellen.

Wenn mehrere von einem Hinweis oder einer Wiedervorlage überwachte BOs erstellt, geändert oder gelöscht wurden, wird für jedes dieser BOs überprüft, ob eine Auslösung erfolgt; wenn ja wird für jedes entsprechende Objekt (BO) eine Benachrichtigung versendet.

Logging/Historie und `AlarmAusloesungen` -Objekte

Jede Auslösung eines Alarms wird automatisch "mitgeloggt". Für jede Auslösung wird ein Objekt vom Typ `AlarmAusloesung` angelegt, mit den Informationen, welcher Alarm wann ausgelöst wurde; für `BO-basierte Termine`, `Hinweise` und `Wiedervorlagen` Außerdem noch, welches Objekt die Auslösung verursacht hat.

Die `AlarmAusloesungen` können z.B. über das Lesezeichen `"Alarme AlarmAusloesungen"` eingesehen werden.

Sonstige Infos

"Verpasste" bzw. "Verspätete" Auslösung

Es kann passieren, das ein Alarm eigentlich zu einem bestimmten Zeitpunkt hätte ausgelöst werden sollen, dies jedoch nicht passiert ist, weil zu diesem Zeitpunkt das Alarmsystem deaktiviert war.

In solchen Fällen wird der Alarm dann normalerweise sofort ausgelöst, sobald das Alarmsystem wieder aktiviert wird.

Dieser Fall kann z.B. auch dann eintreten, wenn bei einer MyTISM-Installation mit synchronisierenden Instanzen eine Änderung, die z.B. einen Hinweis auslöst, auf einer der synchronisierenden Instanzen (ohne Alarmsystem) passiert ist. Wenn nun diese Änderung z.B. durch Netzwerkprobleme oder falsch konfigurierte Synchronisationseinstellungen erst nach einer längeren Zeit auf die autoritative Instanz (mit aktiviertem Alarmsystem) übertragen wird, wird auch hier der Hinweis erst mit dieser Verspätung ausgelöst.

Durch die Angabe von "Alte Alarme nur auslösen wenn nicht älter als" (siehe Abschnitt #alarme_eigenschaften) können Sie festlegen, ob bzw. mit wie viel Verspätung solche Alarme trotzdem noch ausgelöst werden.

Neuinitialisierung der Objekt-Status für BO-basierten Terminen und Wiedervorlagen

Bestimmte Änderungen an bereits bestehenden BO-basierten Terminen oder Wiedervorlagen können dazu führen, dass die für die interne Verarbeitung gespeicherten Informationen zur Auslösung des Alarms für die überwachten Objekte neu initialisiert werden (müssen).

Dies geschieht z.B. bei einem Wechseln der `BOMaske` ("Überwache die Objekte ...") oder auch wenn nur "innerhalb" der Maske die `Entitaet`-Eigenschaft geändert wurde.

Bei solchen Änderungen werden die zum Alarm zugehörigen `*AlarmStatus` neu initialisiert, genauso, als wenn der Alarm neu angelegt worden wäre.

Folgende Änderungen führen zur Neuinitialisierung:

- Für `BO-basierter Termin`: Jede Änderung an `Attribut`, `Script`, `Maske` sowie das *Aktivieren* von `NeuterminierungNachAusloesung`.
- Für `Wiedervorlagen`: Jede Änderung an `AusloeseKriterien`, `AchtetAufBOAendern`, `AchtetAufBOErstellen`, `AchtetAufBOLoeschen`, `Script`, `AenderungVonBenutzer`, `AenderungVonGruppe`, `AKsMitUndVerknuepfen`, `UeberwachungStartenAb`, `Inaktivitaetszeit` und `Maske` sowie das *Aktivieren* von `NeuterminierungNachAusloesung`.

CBOFormat

Diese relativ kleine Klasse hat mittlerweile einen derart hohen Stellenwert im Umgang mit MyTISM erlangt, dass ich ihm hiermit ein eigenes Kapitel widme - ohne ein fundiertes Verständnis der Leistungen dieses Mechanismus macht man sonst viele Sachen um Magnituden komplizierter als nötig - ob es Felder im Report oder einfach "schöne" Lesezeichen sind. Außerdem kann das CBOFormat im Export-Fall sehr nützlich sein.

Was ist CBOFormat?

Ursprünglich wurde CBOFormat entwickelt, um Variablen in Texten auszutauschen und dabei jeglichen "echten" Programmcode zu vermeiden. Dabei ging es um die Abbildung von Regeln wie "Wenn der Vorname leer ist, darf das Komma nach dem Nachnamen nicht gedruckt werden", "drucke die Emailadresse nur bis zum @ und den Rest in die nächste Zeile", oder "wenn das Feld nicht leer ist, dann kommt da noch folgender Text hin".

Alles Geschichten, für die man normalerweise ein Stückchen Programmcode braucht, aber wer schon einmal versucht hat, geschweifte Klammern und diverse andere Sonderzeichen vor dem Rest des Textes zu maskieren - man denke einmal nur an nötige Zeilenumbrüche innerhalb eines etwas komplizierteren Scripts, von Einrückungen ganz zu schweigen - wird ein Lied davon singen können, wie lesbar dann der Programmtext noch ist, ganz zu schweigen von der leicht "zerscripteten" Umgebung.

Es musste also eine Art Pseudocode her, der mit wenig Ballast diese Aufgaben bewerkstelligen kann und trotzdem den Funktionsumfang möglichst komplett abdeckt. Hier ist er:

Zunächst sei erwähnt, daß CBOFormat immer einen Satz Variablen und ein sogenanntes Root-Objekt zur Auswertung übergeben bekommt, und außerdem kompletten Zugriff auf das MyTISM-Schema hat und somit alle Entitäten deren Attribute kennt.

Nun zum ersten Beispiel:

Ein Ansprechpartner mit Familienname und Rufname soll konsistent formatiert werden. Gehen wir mal von einem Ansprechpartner-Objekt mit Familienname, Vorname, Titel, Geburtstag und AnzahlKinder aus. Der Ansprechpartner soll in der Form "Familienname, Rufname, Titel" gedruckt werden; falls aber der Rufname nicht angegeben ist, soll das Komma nicht mit angedruckt werden; ebenso soll beim Titel verfahren werden. Das sieht so aus:

```
Familienname(', 'Rufname)(', 'Titel)
```

Die runde Klammer bewirkt, dass wenn ein Feld darin leer ist, der ganze Konstrukt verschwindet. Mit ? ' ? Die ' um das Komma leiten statischen Text ein. Die Klammer bindet sozusagen einen Auswertungsversuch zusammen - geht er schief, dann verschwindet er komplett.

Das ganze kann man auch etwas weiter ausbauen:

```
(Familienname):('Kein Familienname angegeben!')(', 'Rufname)(', 'Titel)
```

Wie man sieht, kann man hinter einer Klammer einfach einen Doppelpunkt und eine weitere Klammer angeben, die dann benutzt wird, wenn die erste Klammer weg fällt. Das ist fast so wie die if-then-Makros in Word zum Beispiel, nur dass man in unserem Fall so viele Klammern mit Doppelpunkten verketten kann, wie man will (im Fall einer polymorphen Relation kann das sehr nützlich sein).

Man stelle sich jetzt vor, daß man ein Korrespondenz-Objekt an die Hand bekommt und nun diesen

Ansprechpartner in einer CBOFormat-Klausel formatieren soll: (wir nehmen an, daß der Ansprechpartner im Korrespondenz-Objekt über das Attribut "Adressat" definiert ist)

```
(Adressat.Familienname):('-')(', 'Adressat.Rufname)(', 'Adressat.Titel)
```

Sieht umständlich aus, weil das `Adressat` bei vermehrter Nutzung sehr oft angegeben werden muss. Dafür gibts einen einfacheren Weg: Vorklammern über `[`:

```
Adressat[(Familienname):('-')(', 'Rufname)(', 'Titel)]
```

Ein besonderes Verhalten zeigt sich bei der Adressierung von Entitäten. Nehmen wir das Beispiel von vorhin, und notieren einfach folgendes:

```
Adressat
```

Da nun das Ergebnis der Evaluierung ein BO ist, wird dessen Schema-Description zur Formatierung herangezogen. Wenn also im Schema ein

```
... description="(Familienname):('-')(', 'Rufname)(', 'Titel)"
```

steht, dann ist die Ausgabe identisch mit dem oberen Beispiel.

Den gleichen Effekt hat die Verwendung der Variable '.', welche für die Root-Variable steht. Das ist recht nützlich, um eine bestimmte Information vor oder nach der schon vorhandenen (und gegebenenfalls komplexen) Beschreibung hinzuzufügen:

```
Adressat['['Id'] '.]
```

Obiges Beispiel gibt zum Beispiel den Adressaten wie im Beispiel davor aus, allerdings mit seiner Id in eckigen Klammern.

Abweichendes Attribut aus der Attributkette als Label verwenden

Bei der Anzeige von Attributwerten kann im Attributpfad das Trennzeichen `-` statt des normalen `.` verwendet werden, um eine beschreibende Bezeichnung für den angezeigten Wert am Ende des Attributpfad zu erhalten. Durch das Setzen eines Strichs als Trennzeichen (quasi eine "Markierung" in der Attributkette) wird bewirkt, dass für das Label der Name des Attributs direkt **vor** dem Strich verwendet wird, anstatt wie sonst üblich der Attributname **am Ende** des Pfades. Dies hat den Vorteil, dass man sich eine separate Angabe des Labels über die Syntax $R spart und trotzdem eine spezifische beschreibende Bezeichnung für den Attributwert erhält.

Beispiele

```
MehrwertSteuer-Hoehe -> wirkt wie ein $R{MehrwertSteuer} als Label (statt $R{Hoehe})
Beleg.Adressat-Name1 -> wirkt wie ein $R{Adressat} als Label (statt $R{Name1})
Beleg-Adressat.Name1 -> wirkt wie ein $R{Beleg} als Label (statt $R{Name1})
```

Datum und Zeitwert-Formatierung

Eine weitere interessante Möglichkeit ist das Formatieren von Datum und Zeitwerten. Das geschieht einfach über das Anhängen einer geschweiften Klammer direkt an den Wert:

```
"Geburtstag{dd.MM.yyyy, HH:mm:ss 'Uhr'}"
```

Das würde den Geburtstag in der Form "24.04.1971, 15:35:00 Uhr" ausgeben.

Die verwendbaren Zeichen finden sich in der nachstehenden Tabelle:

Table 2. Die Bezeichner des SimpleDateFormat

Symbol	Datums- oder Zeitkomponente	Beispiel
G	Zeitalter	v.Chr, n.Chr
y	Jahr	2004, 04
Y	Wochenjahr	2004, 04
M	Monat im Jahr	Juli, Jul, 07
w	Woche im Jahr	27
W	Woche im Monat	2
D	Tag im Jahr	189
d	Tag im Monat	10
F	Wochentag im Monat	2 (also der 2. Dienstag im aktuellen Monat)
E	Wochentag textuell	Dienstag, Di
a	AM/PM	PM
H	Stunde im Tag (0-23)	0
k	Stunde im Tag (1-24)	24
K	Stunde in AM/PM (0-11)	0
h	Stunde in AM/PM (1-12)	12
m	Minute in der Stunde	30
s	Sekunde in der Minute	55
S	Millisekunden	978
z	Zeitzone Generisch	Pazifische Standardzeit; PST; GMT-08:00
Z	Zeitzone nach RFC822	-0800
'	Textbegrenzer	'Uhr'

Sollte in der Formatierung die Woche im Jahr benutzt werden (w), sollte für das Jahr das Wochenjahr benutzt werden (Y statt y). Sonst könnte es zu Verwirrungen bei Daten am Anfang des Jahres kommen, da die ersten Tage oft noch in die letzte Woche des Vorjahres fallen.

Zahlen-Formatierung

Zahlen lassen sich ebenso wie formatieren:

```
"AnzahlKinder{#,##0.000}"
```

würde zum Beispiel die Anzahl Kinder auf 3 Stellen nach dem Komma, die Tausender dreistellig gruppiert ausgeben.. :-)

Ein Zahlenformat beinhaltet optional ein negatives Format, abgetrennt durch ? ; ?, z.B. `#,##0.00+;#,\##0.00-`.

Die Definition der Symbole in nachstehender Tabelle:

Table 3. Die Bezeichner des DateFormat

Symbol	Ort	Landesabhängig	Bedeutung
0	Nummer	ja	Ziffer
#	Nummer	ja	Ziffer, 0 wird nicht gedruckt
.	Nummer	ja	Dezimaltrenner
-	Nummer	ja	Minus-Symbol
,	Nummer	ja	Gruppierungs-Symbol
E	Nummer	ja	Mantissen/Exponent-Separator.
;	Formattrenner	ja	Trennt positives von negativem Format.
%	Pre/Suffix	ja	Multipliziere mit 100 und zeige als Prozent.
\u2030	Pre/Suffix	ja	Multipliziere mit 1000 und zeige als Promille.

\u00A4	Pre/Suffix	ja	Platzhalter für das Währungssymbol, wird ersetzt durch das aktuelle Währungssymbol. Doppelt zeigt es das internationale Währungssymbol. Wenn es innerhalb eines Formates benutzt wird, tauscht es den Dezimaltrenner gegen den Währungsdezimaltrenner aus (ist in manchen Ländern üblich).
'	Pre/Suffix	nein	Textbegrenzer für spezielle Zeichen.

Funktionsaufrufe

Ein weiteres, allerdings selten benutztes Feature ist die Verwendung von Funktionen im CBOFormat. Das liegt nicht zuletzt daran, daß durch die Verwendung eines reinen Forward-Parsers eigentlich immer nur der aktuelle Wert zur Verfügung steht und somit nur reine String-Modifikationen möglich sind. Nichts desto trotz seien sie hier kurz vorgestellt. Eingeleitet werden die Funktionen mit | (Pipe), die Parameterübergabe erfolgt in Klammern. Die Klammern nach dem Funktionsnamen sind obligatorisch. Meist werden die Funktionen erst dann wirklich nützlich, wenn man sie zusammen mit der runden Klammer einsetzt.

equals(s)

Vergleicht den gerade aktiven String mit dem angegebenen String. Fällt der Vergleich positiv aus, bleibt der aktive String unverändert, wenn nicht, wird der aktive String geleert.

notEqual(s)

Vergleicht den gerade aktiven String mit dem angegebenen String. Fällt der Vergleich negativ aus, bleibt der String unverändert, wenn nicht, wird der aktive String geleert.

```
(Ansprechpartner.Familienname|notEqual('bla')):('blablabla')
```

```
Das Beispiel gibt im Falle eines Familiennamens "bla" statt dessen ein
"blablabla" aus
```

reverse()

Dreht den aktuellen String rückwärts.

cutLeftFrom(s)

Schneidet den aktuellen String an der Kante des übergebenen Strings links ab.

```
Ansprechpartner.Emailadresse|cutLeftFrom('@')
```

```
Ergibt im Falle von "foo@bar.com" ein "foo".
```

cutRightFrom(s)

Schneidet den aktuellen String an der Kante des übergebenen Strings rechts ab.

```
Ansprechpartner.Emailadresse|cutRightFrom('@')
```

```
Ergibt im Falle von "foo@bar.com" ein "bar.com".
```

ifTrue()

Falls der aktuelle String nicht leer ist, ist er es danach.

```
(Ansprechpartner.EmailAdresse|ifTrue()):('ja')
```

```
Gibt für den Fall, daß der Ansprechpartner eine Mailadresse hat, ein
"ja" zurück.
```

ifFalse()

Falls der aktuelle String leer ist, ist er danach nicht mehr leer.

```
(Ansprechpartner.EmailAdresse|ifFalse()):('nein')
```

```
Gibt für den Fall, daß der Ansprechpartner keine Mailadresse hat, ein
"nein" zurück.
```

stripLF()

Diese Operation entfernt alle Zeilenschaltungen aus dem aktuellen String.

left(count)

Gibt die ersten count Zeichen vom aktuellen String zurück.

right(count)

Gibt die letzten count Zeichen vom aktuellen String zurück.

strip()

Entfernt alle Leerzeichen um den aktuellen String herum.

Script-Verwendung

Nun gibt es immer noch Situationen, da geht's ohne Script einfach nicht. Dafür kann man auch ein Beanshell-Script in doppelter geschweiften Klammern angeben und dann den passenden String zusammenbasteln.

Um an die erforderlichen Daten heranzukommen wird die Root-Referenz als "bo" und alle im Variablenhash definierten Variablen unter ihrem dort hinterlegten Namen eingeblendet.

```
'vorgestern war'{{new
SimpleDateFormat("EEE").format(Calendar.getInstance().roll(Calendar.DAY_OF_YEAR,-
2).getTime())}}
```

Dabei kommt die Eigenschaft der Beanshell, das letzte Ergebnis als Rückgabewert zu liefern zu Hilfe, sonst wäre bei dem Beispiel noch ein return …; notwendig gewesen.

Wo kann man das CBOFormat nun überhaupt einsetzen?

Das CBOFormat findet seine Anwendung zunächst einmal in der Schema-Definition, und zwar in Form des "description" Attributs. Es soll helfen, die Entitäten mit einer Art textueller Beschreibung auszustatten (Programmierern als toString() Methode bekannt). Ich wollte aber aus verschiedenen Gründen nicht die toString() Methode überladen, weil eine für das Debugging wichtige Information, der hashCode, mit angegeben wird, der aber für den Benutzer völlig nichtssagend ist; zudem sind mehrere solcher descriptions denkbar (wenn auch (noch) nicht implementiert) oder können auch ad hoc angefordert werden.

Dafür hat jede vom System generierte Entität eine describe-Methode, die einen optionalen Stringparameter bekommt und auf diesem Weg einen String der gewünschten Form ausgibt. Das kann vom Reportgenerator über direkte Objektreferenzen direkt benutzt werden (zum Beispiel $F{THIS}.describe()), oder innerhalb einer BO-Methode für Debugging Ausgaben, Export-Formate…

In Solstice begegnet man dem CBOFormat ständig. Überall, wo ein "displayFormat" angegeben werden kann, ist das CBOFormat am Werk; in Lesezeichen die Spaltendefinition, in Formularen für Labels, TablePopups, in den Fenstertiteln etc.

MEX - Makros und erweiterte Query-Funktionen

Leider hat der bislang im MyTISM verwendete OQL-Parser nicht alle Funktionen, die man sich wünschen könnte. Die bislang fehlenden Funktionen sind unter anderem Subqueries, Subclass-Casting, explizite Joins, Unions und Fetch-Strategien. Leider können wir nicht alle Problem auf einmal lösen, aber dennoch gibt es zumindest für die Union und die dadurch substituierbaren Subclass-Castings eine Lösung: MEX.

Diese Geschichte soll aber kein Notnagel sein, um Probleme zu kaschieren, um später wieder ausgebaut zu werden. Insgesamt ist es eine gute Möglichkeit, in einigen Queries Roundtrips zu vermeiden oder einfach in der GUI komplexe Queries überhaupt handhabbar zu machen. Ein Präprozessor eben, der ohnehin verfügbare Möglichkeiten der API dem Benutzer zugänglich macht.

Definition MEX

Die Auswertung der einzelnen Konstrukte erfolgt an verschiedenen Stellen im Kernel, je nachdem wer sich dafür zuständig fühlt. MEX, so der Name der Sprache, besteht im wesentlichen aus verschachtelten geschweiften Klammer-Blocks. Die Funktionsweise ist die, daß ein bestehender Text mit den Tags versehen wird und auf der Serverseite ausgewertet wird, wobei in mehreren Schritten die Klammer-Blocks entfernt werden. Bleiben nach der Auswertung noch unbehandelte Klammer-Blocks übrig, kommt es zu einer Fehlermeldung, in der der unausgewertete Klammer-Block erwähnt wird, so daß man den Fehler hoffentlich direkt sehen kann.

Der Core, der MEXTransformer, beherrscht nur 3 Konstrukte:

Sichtbare Variablendefinition

Definiert eine Variable und ersetzt die Definition an dieser Stelle mit dem definierten Wert.

```
{hausnr=4711}
```

wird ersetzt durch

```
4711
```

und die Variable `hausnr` hat im Anschluß den Wert 4711.

Unsichtbare Variablendefinition

Definiert eine Variable und entfernt die Definition aus dem Quelltext.

```
{!hausnr=4711}
```

verschwindet komplett und die Variable `hausnr` hat im Anschluß den Wert 4711.

Variablenexpansion

Ersetzt die angegebene Variable durch ihren Wert.

```
{=hausnr}
```

wird ersetzt durch

```
4711
```

(vorausgesetzt, die Variable wurde mit diesem Wert definiert).

Wichtig zu wissen ist, daß die Variablendefinition und die Auswertung in verschiedenen Stufen passieren, die Variable hat also den Wert, den sie zuletzt im Text hatte im Anschluß in jeder

Variablen, wo auch immer sie erwähnt wurde. Ein Beispiel:

```
{a=1}{=a}+{=a}={a=2}{=a}
```

ergibt

```
2+2=2
```

Auf diese Weise kann man allerdings Werte schon benutzen bevor die Variablen ihren Wert erhalten haben. Wir können innerhalb der GUI später nicht explizit etwas früher oder später definieren wollen. Die Variablen werden genau einmal definiert und benutzt, ändern aber ihre Werte nicht während der Auswertung.

Das sind bereits alle Konstrukte, die MEX (im Moment) selbst beherrscht. Alle anderen kommen von anderen "Schichten" dazu. Mehr dazu im nächsten Kapitel.

Support auf Queryseite

MEX wird momentan auf Seiten des Backend ausschließlich für Queries benutzt. Der zuständige CastorPersistenceHandler, der die Queries im Backend an den ORM Castor abliefert, implementiert folgenden Konstrukt:

```
          select a from de.ipcon.db.core.Formular a where a.BOTyp.Name="Beleg"
{UnionAll select a from de.ipcon.db.core.Schablone a where a.BOTyp.Name="Beleg"}
```

(Man sollte erwähnen, daß BOTyp in beiden Fällen eine Eigenschaft ist, die nicht von der Oberklasse Struktur kommt und daher nicht über Struktur her fragbar ist). Dieses Beispiel gibt in einem einzigen Roundtrip zum Backend die Ergebnisse in einer einzigen Antwort zurück. Man kann diese Konstruktion auch beliebig oft wiederholen, ganz, wie die Situation es erfordert. Dabei sind auch Projektionen und andere Tricks kein Problem:

```
          select a.Benutzer from de.ipcon.db.core.Gruppe a where not Ldel and
a.Formulare.BOTyp.Name="Beleg"
{UnionAll select a from de.ipcon.db.core.Benutzer a where not Ldel and
a.Gruppen.Name="Admins"}
```

Im Moment sind Union und UnionAll funktionsgleich, weil die Dopplerfindung im entsprechenden QueryIterator noch nicht durchgebaut ist

Support in Solstice

Schließlich und endlich unterstützt die Solstice-Oberfläche das ganze dadurch, daß die CBOTextQuery, also diejenige Query, die durch <Query type="Text"/> gewählt wird, Teile der Query als Variablen bereits deklariert. So wird folgende typische Query nach Benutzern:

```
select a from de.ipcon.db.core.Benutzer where not Ldel
```

wie folgt umgeschrieben:

```
{!select=select a from}
{!where=a where}
{!constraints=not Ldel}
{=select} de.ipcon.db.core.Benutzer {=where} {=constraints}
```

Sieht jetzt sehr merkwürdig aus, ist aber absolut funktionsidentisch mit der ersten Version. Zwei Mechanismen sind jetzt interessant: Zum einen kann die letzte Zeile der Query mit dem Tagnamen <template> ausgetauscht werden. Man kann also so etwas schreiben:

```
<Query type="Text">
  <template>{=select} de.ipcon.db.core.Benutzer {=where} {=constraints} and Name like
= "A%"</template>
</Query>
```

Jetzt wirds interessant: Jeder Filter, jede Volltextsuche und weitere direkt eingegebene OQL-Schnipsel werden allesamt in der Variablen constraint mit eingearbeitet, so daß der einfache Query oben schnell an Komplexität gewinnt. Jetzt ist es natürlich leicht, den oben erwähnten Union-Konstrukt zu verwenden, um sämtliche Constraints auch schön durch alle Union-Teile mit durchzuschleifen:

```
<template>
            {=select} de.venice.bo.Rechnung {=where} {=constraints} and (Not Bezahlt
or is_undefined(Bezahlt))
  {UnionAll {=select} de.venice.bo.EingangsRechnung {=where} {=constraints} and
FreigabeFiBu}
</template>
```

Es wird noch netter. Es kann ja sein, daß man verschiedene Attribute nicht gleichlautend und/oder gar nicht in bestimmten Subklassen zu finden sind, aber dennoch über die netten grafischen Filter zu wählen sein sollen. Für diesen Zweck kann man die Filter nun gruppieren. Man gibt einem Filter nun einfach ein group="R" mit, und damit landen seine Query-Constraints in einer Variablen namens "constraintsR". Ein Beispiel:

```
<Query type="Text">
  <template>
              {=select} de.venice.bo.Rechnung {=where} {=constraints} and
{=constraintsAR} and (Not Bezahlt or is_undefined(Bezahlt))
    {UnionAll {=select} de.venice.bo.EingangsRechnung {=where} {=constraints} and
{=constraintsER} and FreigabeFiBu}
  </template>
  <filter type="bool" group="AR" title="Ausgangs-Rechnung hat Skonto">
    <ifTrue>is_defined(Skonto)</ifTrue>
  </filter>
  <filter type="string" group="AR" title="Kunden-Nr">
    <clause>Kunde.DebitorenNr like "%{}%"</clause>
  </filter>
  <filter type="string" group="ER" title="Lieferanten-Nr">
    <clause>Lieferant.KreditorenNr like "%{}%"</clause>
  </filter>
</Query>
```

Dieses Beispiel verdeutlicht ganz gut die möglichen Szenarien solcher Queries.

Weitere geplante Features sind die Verwendung von Skalaren und Listen als Parameter für und von Subqueries und Fetch-Strategien zur Verbesserung des Unlazy-Verhaltens der GUI (prefetching von Relationen z.B.).

Volltextsuche

Die Volltextsuche erlaubt die einfache und schnelle Suche nach gegebenen Suchbegriffen über alle in der MyTISM-Datenbank gespeicherten Objekte.

Vorbereitung und Konfiguration

Volltextsuche aktivieren

Die Volltextsuche ist normalerweise deaktiviert, d.h. Sie können in Abfragen keine auf der Volltextsuche basierenden Klauseln verwenden. Abfragen mit solchen Klauseln führen bei nicht aktiver Volltextsuche zu einer Fehlermeldung.

Um die Volltextsuche zu aktivieren müssen Sie in der Datei `mytism.ini` im Abschnitt "Fulltextsearch" das Flag "activateFts" setzen (sollte auch der entsprechende Abschnitt noch nicht existieren, fügen Sie ihn einfach ebenfalls ein):

```
[Fulltextsearch]
activateFts=1
```

Einstellungen

Für alle Einstellungen existieren Standardwerte; im Normalfall ist also keine weitere Konfiguration für die Volltextsuche notwendig und meist auch nicht sinnvoll. Lediglich der Parameter `max_locks_per_transaction` der PostgreSQL-Instanz sollte angepasst werden, da aufgrund der besonderen Gegebenheiten der Volltextsuche die Standardeinstellung dafür nicht ausreichend zu sein scheint.

PostgreSQL: max_locks_per_transaction

Bei der (initialen) Indexierung für die Volltextsuche werden u.U. viele gleichzeitige und relativ langlaufende Anfragen an die PostgreSQL-Datenbank gestellt. Aus diesem Grund kann es nötig sein, den Parameter `max_locks_per_transaction` in der Datei `/etc/postgresql/8.4/main/postgresql.conf` (der Pfad kann je nach benutzter Version und Konfiguration ggf. abweichen) zu erhöhen.

Im Normalfall sollte es keine Probleme machen, diesen Wert einfach auf z.B. 1024 zu setzen, was auch für die Indexierung vollkommen ausreichend sein sollte. Nachdem der Wert in obiger Konfigurationsdatei geändert wurde, muss die PostgreSQL-Instanz durchgestartet werden.

Betriebssystem: Mögliche Anzahl gleichzeitig offener Dateien

Je nach Betriebssystem und Konfiguration kann es sein, dass der Wert für die mögliche Anzahl gleichzeitig offener Dateien erhöht werden muss. Ein Symptom dafür sind entsprechende Fehlermeldungen während des Betriebs der Applikation. Da der zu setzende Wert je nach System und Konfiguration verschieden sein kann, können wir hierzu allerdings keine allgemeingültigen Anweisungen oder Standardwerte geben.

indexAllByDefault

Im Gegensatz zu früheren Versionen der Volltextsuche werden jetzt standardmäßig *keine* Entitäten in den Index aufgenommen; nur Entitäten, die im Schema explizit mittels `<fulltext`

`indexed="yes"/>` markiert wurden, werden für die Volltextsuche indexiert.

Um diesen Standard zu ändern, so dass erst einmal *alle* (abgesehen von einer Handvoll systeminterner) Entitäten in den Index aufgenommen werden, können Sie folgende Einstellung verwenden:

Standardmäßig (fast) alle Entitäten indexieren:

```
[Fulltextsearch]
activateFts=1
indexAllByDefault=1
```

indexDeletedBOs

Standardmäßig werden auch als gelöscht markierte Objekte für die Volltextsuche indexiert. Wenn Sie dies nicht möchten oder benötigen können Sie die Indexierung von gelöschten Objekten wie folgt deaktivieren:

Gelöschte Objekte nicht indexieren:

```
[Fulltextsearch]
activateFts=1
indexDeletedBOs=0
```

spellcheck

Wenn Sie die Spellcheck/"Meinten sie vielleicht..."-Funktionalität zum Vorschlagen von alternativen Suchwörtern nutzen wollen, müssen Sie diese explizit aktivieren:

Spellcheck/"Meinten sie vielleicht..."-Funktionalität aktivieren:

```
[Fulltextsearch]
activateFts=1
spellcheck=1
```

Um diese Funktionalität nutzen zu können muss sich die JAR-Datei lucene-spellchecker.jar im Classpath befinden.

fetchSize

Mit dieser Einstellung kann bestimmt werden, in welchen "Packen" Objekte bei der Indexierung aus der Datenbank geladen werden. Abfragen, die sehr lange laufen, werden irgendwann automatisch abgebrochen; mit diesem Parameter kann verhindert werden, dass Abfragen zu viel Zeit in Anspruch nehmen, indem die Anzahl der pro Abfrage zu ladenden Objekte limitiert wird.

fetchSize auf 100.000 erhöhen:

```
[Fulltextsearch]
activateFts=1
fetchSize=100000
```

Im Normalfall werden Sie diese Einstellung jedoch selten benötigen; der Standardwert 50.000 wird normalerweise ok sein.

maxFieldLength und unlimitedFieldLength

Wenn indexierte Objekte Felder mit sehr großen/langen (Text)werten enthalten, werden von diesen standardmäßig nur die ersten 10.000 Zeichen indexiert. In gewissen Fällen kann dies nicht ausreichend sein, oder alternativ zu viel und unnötig sein, so dass sie diese Grenze verändern können.

Die ersten 50.000 Zeichen von Feldinhalten indexieren:

```
[Fulltextsearch]
activateFts=1
maxFieldLength=50000
```

Um (praktisch) beliebig lange Feldinhalte zu indexieren, können sie die Einstellung `unlimitedFieldLength` aktivieren:

Gesamten Feldinhalt von (praktisch) beliebiger Länge indexieren:

```
[Fulltextsearch]
activateFts=1
unlimitedFieldLength=1
```

indexPath

Es hat sich herausgestellt, dass auf manchen Systemen ein Eintrag `indexPath=niofs:///<DURCH KORREKTES PROJEKTVERZEICHNIS ERSETZEN>/index` notwendig ist, um einen Fehler ("Setting type of FS directory is a JVM level setting, you can not set different values within the same JVM") beim Serverstart zu vermeiden.

Diese Einstellung für die Volltextsuche betrifft das Verzeichnis, in dem die Dateien des Index gespeichert werden. Über "indexPath=ein/pfad/im/dateisystem" können sie bestimmen, wo diese Dateien abgelegt werden.

Ein Beispiel, unter Linux:

```
[Fulltextsearch]
activateFts=1
indexPath=/var/lib/mytism/ftsindex
```

Ein Beispiel, unter Windows:

```
[Fulltextsearch]
activateFts=1
indexPath=C:\Daten\MyTISM\FTS-Index
```

Im Normalfall werden Sie diese Einstellung jedoch selten benötigen; wenn kein Eintrag für "indexPath" vorhanden ist, wird der Standardpfad benutzt. In diesem Fall werden die Index-Dateien unterhalb eines Verzeichnisses namens `index` im Projekt-Verzeichnis der MyTISM-Installation abgelegt, also z.B. unter `/.is/index`.

Interessant in diesem Zusammenhang könnte evtl. sein, dass über diesen Parameter auch noch Einstellungen für den Dateisystemzugriff gemacht werden können, die sich evtl. auf die Performance auswirken können. Siehe hierzu auch 4.1. File System Store.

Java 1.4 NIO zum Zugriff benutzen:

```
[Fulltextsearch]
activateFts=1
indexPath=niofs:///.is/index
```

maxThreads

Um die Leistung des Servers optimal zu nutzen, werden bei der Indexierung parallel mehrere Abfragen abgesetzt, um die in den Suchindex aufzunehmenden Objekte zu laden.

Je nach Leistungsfähigkeit des Servers können mehr oder weniger Abfragen gleichzeitig bearbeitet werden. Ausserdem wird die Anzahl der gleichzeitig möglichen Abfragen durch die erlaubte Anzahl von Verbindungen zur Datenbank, etc. begrenzt. Mit dieser Einstellung kann die maximale Anzahl der gleichzeitig für die Indexierung laufenden Threads begrenzt werden.

Eine geringe Angabe für `maxThreads` führt lediglich dazu, dass die ggf. vorhandenen Ressourcen des Systems nicht optimal genutzt werden und die Indexierung länger dauern kann, als eigentlich erforderlich. Eine zu hohe Angabe kann dagegen dazu führen, dass die Indexierung aufgrund zu grosser Anforderungen an das System fehlschlägt, was normalerweise zur Folge hat, dass die Indexierung wieder komplett neu gestartet werden muss.

Der Standardwert für `maxThreads` ist die Anzahl der initial für die Java Virtual Machine verfügbaren Prozessoren und sollte im Normalfall ok sein. Wenn Sie wichtige Gründe dafür haben, können Sie diesen Wert verringern oder erhöhen. Wenn Sie hier `-1' angeben, sind beliebig viele parallel laufende Threads erlaubt; konkret heisst das, dass für jede im Schema definierte (und für die Indexierung vorgesehene) Entity ein eigener Thread gestartet wird.

Wenn Sie sehr viele gleichzeitige Abfragen benutzen wollen müssen Sie ggf. auch den Wert für `max_connections` in der PostgreSQL-Konfiguration (s.o.) erhöhen - und zwar *mindestens* auf den Wert, den Sie für maxThreads angegeben haben - da es sonst während der Indexierung zu Fehlern kommen kann.

maxThreads auf 15 setzen:

```
[Fulltextsearch]
activateFts=1
maxThreads=15
```

directoryWrapper

Über diese Einstellung kann der Zugriff auf den Index über einen Wrapper gekapselt werden, was ggf. Verbesserungen bei der Performance bringen kann. Siehe hierzu auch 4.6. Lucene Directory Wrapper.

```
[Fulltextsearch]
activateFts=1
directoryWrapper=org.compass.core.lucene.engine.store.wrapper.AsyncMemoryMirrorDirecto
ryWrapperProvider
```

compassConfig

Mittels dieser Einstellung können Sie den Pfad zu einer Konfigurationsdatei angeben, mit der Sie praktisch beliebige Einstellungen direkt zum verwendeten Compass-Suchframework durchreichen können.

Weitere Infos dazu siehe Compass-Dokumentation.

Der Index

Grundlage der Volltextsuche ist der sogenannte Index; grob gesagt handelt es sich dabei um eine Struktur, die die Daten der in der MyTISM-Datenbank gespeicherten Objekte in einer Form enthält, welche eine einfaches Auffinden nach zu eingegebenen Suchbegriffen passenden Objekten ermöglicht.

Initiale Erstellung

Dieser Index wird normalerweise einmal erstellt und im Weiteren dann automatisch aktualisiert, wenn Änderungen an den Objekten in der Datenbank vorgenommen werden. Ist die Volltextsuche aktiviert und noch kein Index vorhanden, wird beim Starten der MyTISM-Instanz automatisch ein Index erzeugt. Die Volltextsuche ist erst verfügbar, wenn der Index fertig komplett wurde.

Für die Erstellung müssen *alle* Objekte, die mittels der Volltextsuche gefunden werden können sollen, geladen und ihre Daten in den Index eingespeichert werden. Je nach Grösse der MyTISM-Datenbank und der Anzahl der dort gespeicherten Objekte sowie der Leistungsfähigkeit der Server-Machine, auf der die MyTISM-Instanz läuft, kann dieser Vorgang von einigen Minuten bis hin zu vielen Stunden (oder noch länger) in Anspruch nehmen.

Während dieser Zeit sind der Server und die MyTISM-Instanz aufgrund der vielen und umfangreichen Abfragen normalerweise stark ausgelastet, was ggf. natürlich Beeinträchtigungen für die normale Benutzung mit sich bringen kann.

Ausserdem sollte darauf geachtet werden, dass die Erstellung des Index nicht unterbrochen wird (z.B. durch Herunterfahren der MyTISM-Instanz oder des gesamten Servers), da es wahrscheinlich ist, dass sich bei einer Unterbrechung der Index in einem halbfertigen, nicht benutzbaren Zustand befindet und daher die Indexerstellung später noch einmal komplett neu angestossen werden muss.

Wie bereits oben erwähnt muss die Indexerstellung jedoch im Normalfall nur ein einziges Mal gemacht werden, so dass es sich hierbei um eine einmalige Einschränkung handelt.

Erneute Erstellung / Re-Indexierung

Soll der Index aus irgendeinem Grund vollständig neu erstellt werden, gibt es zwei Möglichkeiten, dass zu erreichen:

1. Anlegen einer Datei namens ".force-fts-index-rebuild" im MyTISM-Projektverzeichnis. Dies ist die normale, bevorzugte Methode. Der Inhalt der Datei ist unwichtig, sie kann leer sein.
   Die Datei wird nach der Indexerstellung automatisch gelöscht.
2. Händiges Löschen des Index-Verzeichnisses inkl. aller darin enthaltenen Unterverzeichnisse und Dateien. Diese Methode muss ggf. angewandt werden, wenn die Indexerstellung unterbrochen wurde (s.o.).

In beiden Fällen wird der Index von Grund auf neu erstellt, wie im vorherigen Abschnitt beschrieben, mit allen dort erwähnten Einschränkungen für die Benutzung der Anwendung

währenddessen.

Verteilen des Index für synchronisierende Server

Da der Index keinerlei Instanz-spezifische Informationen enthält, kann ein einmal erstellter Index auch für eventuell vorhandenen, synchronisierende Server verwendet werden. Dies ist natürlich insb. bei grossen Datenbanken sinnvoll, damit die aufwändige Indexerstellung nicht mehrmals erfolgen muss.

Das Verteilen des Index ist z.Zt. jedoch noch nicht automatisch möglich und daher muss der Index händig auf die entsprechende Server-Machine kopiert werden. Dies geschieht einfach durch Kopieren des gesamten Index-Verzeichnisses, inklusive aller darin enthaltenen Unterverzeichnisse und Dateien, in das MyTISM-Projektverzeichniss (bzw. das in der Konfiguration angegeben Verzeichnis) auf dem synchronisierenden Server.

Die MyTISM-Instanz, deren Index kopiert werden soll, sollte entweder ganz gestoppt oder die Volltextsuche sollte nicht aktiviert sein. Ist dies nicht der Fall, kann es sein, dass der Index gerade aktualisiert wird, was ggf. dazu führt, dass er nach dem Kopieren nicht benutzbar ist. Auch auf dem Zielserver sollte die Volltextsuche nicht aktiv sein, wenn die Index-Dateien dorthin kopiert werden; die MyTISM-Instanz an sich kann aber laufen.

Um nach dem Kopieren der Index-Dateien die Volltextsuche auf dem Zielserver zur Verfügung zu stellen, muss diese in der Konfiguration aktiviert werden und die MyTISM-Instanz auf dem Zielserver danach neu gestartet werden.

Konfiguration für die in den Index aufzunehmenden Daten

Im Normalfall werden alle textuellen Daten aller Objekte in der MyTISM-Datenbank für die Volltextsuche aufbereitet und im Index eingespeichert. Als Entwickler einer MyTISM-Anwendung können Sie die Indexierung allerdings weitergehend konfigurieren und z.B. bestimmen, dass bestimmte Objekte oder bestimmte Daten von Objekten nicht indexiert werden sollen.

Da diese Möglichkeit jedoch nur beim Bauen einer MyTISM-Anwendung besteht und im fertigen Produkt nicht mehr weiter konfigurierbar ist finden sich ausführliche Informationen hierzu in der MyTISM-Entwicklerdokumentation.

Benutzung der Volltextsuche

Wenn die Volltextsuche in der MyTISM-Konfiguration aktiviert und der Index vollständig erstellt wurde, kann die entsprechende Funktionalität genutzt werden.

Standard-Abfragen

Volltextsuche-Kriterien können in Abfragen (Queries) als MEX-Ausdrücke eingefügt werden. Die entsprechende Syntax lautet: `Fulltext [from <Entitätsname>] matches <Volltext-Suchklausel(n)>`

> *Beispiel:* Eingabe von `[\{Fulltext matches Schmitt}` in der Suchzeile eines Kunden-Lesezeichen findet alle Kunden(-Objekte) die in irgendeinem ihrer (indexierten) Attribute die Zeichenkette "Schmitt" enthalten.

Normalerweise wird über alle Attribute der Objekte gesucht; es können aber auch nur bestimmte Attribute in die Suche einbezogen werden.

> *Beispiel:* Eingabe von `[\{Fulltext matches Name:Schmitt}` in der Suchzeile eines Kunden-Lesezeichen findet alle Kunden(-Objekte) die in ihrem Attribut "Name" die Zeichenkette "Schmitt" enthalten.

Die Namen der Attribute entsprechen dabei genau den Namen, die im MyTISM-Schema angegeben sind; Gross- und Kleinschreibung sind dabei zu beachten.

Weitere Angaben zur Abfrage-Syntax finden sich auch noch in der Compass-Dokumentation.

Einschränkungen der Entität

FIXME Im Normalfall keine Angabe nötig, Default ist Entität der Abfrage.

Grooql (Groovy Object Query Language)

Eine alternative Möglichkeit, Objektmengen abzufragen. Hat Ähnlichkeiten/Überschneidungen mit OQL und BOMasken. Besteht aus Filterskripten, in einer eingeschränkten Groovy-Version geschrieben.

Im Moment fast nur direkt aus Programmcode heraus zu benutzen, noch nicht unterstützt z.B. in Lesezeichen (ist aber geplant). Außerdem via `GrooqlBOMaske`.

Paket `de.ipcon.db.grooql`, "Hauptklasse" `GrooqlFilter`; Javadoc dort:

```
* {@code GrooqlFilter} allow to query {@code BOs} that match given
* criteria from the DB and also check if given {@code BOs} match these
* criteria, both accomplished using only one single criteria definition.
* <p>
* {@code GrooqlFilter} could thus be seen as a combination of OQL queries
* and {@code BOMasken}.
* <p>
* The criteria definition is given as a script in a subset of the Groovy
* language. The script is used directly to check if given {@code BOs}
* match these criteria using the {@code fits()} method. For querying
* matching {@code BOs} from the DB the script is automatically transformed
* into an OQL query which retrieves <em>a superset</em> of the matching
* {@code BOs} which are then post-filtered with the script.
* <p>
* All attributes of the {@code BO} that is currently checked are available
* as variables in the script with their simple name; for example if a filter
* was defined for the class {@code Benutzer} {@code Name},
* {@code Beschreibung}, {@code AnmeldungVerweigern} etc. would be
* available as variables in the script under the exact above names.
```

Sprachumfang

Unterstützt werden z.Zt.:

- Logische Verknüpfungen: `&&` (und), `||` (oder), `!` (nicht)
- Operatoren für Skalare: `==`, `!=`, `<`, `>`, `>=`
- Methoden für Zeichenketten: `.startsWith( foo )`, `.endsWith( foo )`, `.contains( foo )`, `matchesSimple('bla*')`, `.lower()`, `.upper()`, `.trim()`
- Arithmetik/Zahlen: `+`, `-`, `*`, `/`, `%` (Modulo)
- Methoden für Datums- und Zeitwerte:
  - Vergleich: `.after(someDate)`, `.before(someDate)`
  - Extrahieren von Datums-"Teilen": `.day`, `.month`, `.year` bzw. alternativ `.getDay()`, `.getMonth()`, `.getYear()`
  - Genauigkeit/Granularität vergröbern: `.thatDay()` (setzt h/min/sec auf 0), `.thatMonth()` (setzt d/h/min/sec auf 0), `.thatWeek()` (setzt h/min/sec auf 0 und Tag auf Anfangstag (Montag) der entsprechenden Woche), `.thatMonth()` (setzt d auf 1, h/min/sec auf 0), `.thatYear()` (setzt M/d auf 1, h/min/sec auf 0)
  - Zukunft: `.addDay(1)`, `.addMonth(-1)`, `.addYear(3)`
  - Vergangenheit: `.subDay(2)`, `.subMonth(-3)`, `.subYear(1)`
- BOs:
  - Zugriff auf Attributketten.
  - Id in einer gegebenen Liste: `.idInList(<id-Liste>)`
- Methoden für Maps (Long → Objekte, insb. MyTISM-BO-Relationen-Attribute): `.containsId(<id>)`, `.containsAllIds(<ids>)`

Kommentare - sowohl mit `//` als auch `/* */` - werden ebenfalls unterstützt.

NICHT unterstützt werden (nur Beispiele, Liste ist keineswegs vollständig):

- Aufrufe von Methoden, außer den oben genannten
- print/println oder Logausgaben
- instanceof <Interface> (Würde mittlerweile prinzipiell gehen, müsste aber nachgebaut werden)
- import
- ...

Beispiele für Filterskripte

Vorraussetzung: GrooqlFilter, der Objekte von Entität (GrooqlFilter.Entity) "Dokument" sucht; diese hat Attribute "Name" (String), "ErstellungsDatum" (Date), "Summe1" (Integer), "Summe2" (Integer). Beispiele für GrooqlFilter.FilterSource:

Alle Dokumente mit bestimmtem Namen:

```
Name = "Bilanz 1"
```

Alle Dokumente mit Namen der mit "Bilanz" beginnt:

```
Name.startsWith("Bilanz 1")
```

Alle Dokumente aus dem Jahr 2011:

```
ErstellungsDatum.year = 2011
```

oder

```
ErstellungsDatum.getYear() = 2011
```

Alle Dokumente neuer als 2011:

```
ErstellungsDatum.year > 2011
```

oder

```
ErstellungsDatum.getYear() > 2011
```

Alle Bilanzen von 2011:

```
Name.startsWith("Bilanz") && ErstellungsDatum.getYear() = 2011
```

Alle Dokumente ohne Namen:

```
Name == null || Name.trim() = ""
```

Alle Dokumente mit Summe1 + Summe2 > 1000:

```
Summe1 + Summe2 > 1000
```

Einstellungen-Variablen

Einstellungen-Variablen dienen dazu, Werte für bestimmte Einstellungen zu setzen, welche dann z.B. in Skripten abgefragt und benutzt werden können. Diese Werte können global gültig oder gruppen- oder benutzerabhängig sein.

Geänderte oder neue Einstellungen-Variablen werden erst nach einer erneuten Anmeldung an der GUI wirksam.

Definition der vorhandenen/verfügbaren Variablen

Normalerweise werden Einstellungen-Variablen vom Administrator oder von Entwicklern, je nach dem Bedarf der spezifischen MyTISM-Anwendung, definiert. Eine Variable(ndefinition) hat folgende Eigenschaften:

Name

Pflichtfeld - Der Name oder Titel einer Variable sollte diese kurz und prägnant benennen. Der Name kann frei gewählt werden; ein wirkliches einheitliches Schema für die Benamsung existiert (bisher) noch nicht.

Beschreibung

Optional - Die Beschreibung kann einen längeren Kommentar bzw. eine längere Beschreibung der Variable beinhalten und ggf. erklären wo bzw. wofür sie benutzt wird.

Standardwert

Optional - Dies ist der Wert, den die Variable normalerweise hat und der bei der Abfrage z.B. in Skripten zurückgeliefert wird, wenn kein spezieller Wert für bestimmte Benutzer oder Gruppen gesetzt wurde (s.u.). Variablenwerte hier sind immer Zeichenketten, eine weiter Typisierung (z.B. für Nummern oder Wahrheitswerte) gibt es nicht. Wenn kein Wert gesetzt wird, ist der Standardwert einfach null.

Ueberschreibbar

Optional - Wenn dieses Flag gesetzt ist, können für einzelne Benutzer oder Gruppen vom Standardwert abweichende Werte für diese Variable definiert werden (oder genauer gesagt: Wenn solche Werte definiert wurden, werden sie auch berücksichtigt; s.u.). Wenn das Flag nicht gesetzt ist, gilt für alle Benutzer oder Gruppen immer nur der Standardwert der Variable.

Abfrage von Einstellungen-Variablen in Skripten

Variablenwerte können wie folgt abgefragt werden (Beispiel aus dem vorgebauten JahrMonatTag-Filter-Codebaustein):

```
def val = ctx.getCurrentUser().getEVWert("jahrMonatTagFilter.Monat")
```

Hat man die Objektinstanz des gewünschten Benutzers in der Hand (hier ist dies der aktuelle, mit Hilfe des `ClientContext` ermittelte Benutzer), kann man mittels der Methode `getEVWert()` den Wert einer beliebigen existierenden Einstellungsvariablen abfragen, indem deren Name der Methode übergeben wird.

Setzen von abweichenden Werten für Benutzer oder Gruppen

Wenn für bestimmte Benutzer oder Gruppen vom Standardwert abweichende, spezielle Werte für eine Variable gesetzt werden sollen, geschieht das durch Anlegen von `EinstellungenVarWertBenutzer` - oder `EinstellungenVarWertGruppe`-Objekten.

In diesen Objekten gibt man an, für welche Variable der Wert "überschrieben" werden soll, für welchen Benutzer oder Gruppe der abweichende Wert gelten soll und natürlich den Wert selbst.

Die Auswertung bzw. Bestimmung welcher Wert für einen spezifischen Benutzer letztendlich zurückgeliefert wird erfolgt so:

1. Wenn eine Variable mit dem gewünschten Namen nicht existiert, wird null zurückgegeben.
2. Wenn die Variable existiert und *Ueberschreibbar* NBSP *nicht* gesetzt ist, wird *immer* der Standardwert der Variable zurückgegeben.
3. Wenn *Ueberschreibbar* gesetzt ist und eine `EinstellungenVarWertBenutzer`-Instanz für den Benutzer und die Variable existiert, wird der dort angegebene Wert zurückgegeben.
4. Wenn *Ueberschreibbar* gesetzt ist, keine passende `EinstellungenVarWertBenutzer`-Instanz existiert, aber eine `EinstellungenVarWertGruppe`-Instanz für die Variable und eine Gruppe, in der der Benutzer Mitglied ist, existiert, wird der dort angegebene Wert zurückgegeben. Wenn mehrere passende Instanzen für die Variable und unterschiedliche Gruppen, in denen der Benutzer Mitglied ist, existieren, so wird der Wert zurückgegeben, der für die Gruppe mit der kleinsten Id definiert wurde.

Lesezeichen und Anzeige in Benutzer- und Gruppen-Formularen

Im Ordner der Gruppe "Benutzer" gibt es ein vorgebautes Lesezeichen, in dem alle für den angemeldeten Benutzer geltende `EinstellungenVarWerte` (sowohl für Benutzer als auch Gruppe) angezeigt werden; allerdings nur solche, für deren zugehörige Variable das Flag Ueberschreibbar gesetzt ist!

In den vorgebauten Formularen für `Benutzer` und `Gruppe` gibt es ebenfalls einen Reiter Variablen; in der dortigen Tabelle werden *alle* für den jeweiligen Benutzer bzw. die jeweilige Gruppe definierten `EinstellungenVarWertBenutzer` bzw. `EinstellungenVarWertGruppe` angezeigt.

Scripted Attributes

Bei den `Scripted Attributes` handelt es sich um `Virtual Properties`, die zur Laufzeit (KEIN Server-Restart oder Client-Neuanmeldung nötig!) an ein BO hinzugefügt werden können - sei es in einem Lesezeichen, einem Formular oder im Report. Als Programmiersprache der `Scripted Attributes` kommt Groovy zum Einsatz, welches nahezu 100% kompatibel zu Java ist.

Der Tag heisst `virtualProperty` und kennt folgende Parameter:

Table 4. virtualProperty-Parameter

Parameter	Beschreibung	Default	Pflichtfeld
entity	Name der Entität, an die das virtuelle Attribut "angebaut" werden soll	-	ja
name	Wie das zu bauende virtuelle Attribut heissen soll	-	ja
type	Von welchem Datentyp das virtuelle Attribut ist; mögliche Typen: String, Integer, Long, Decimal, Date sowie MyTISM-Objekte in Kombination mit einer `relation`-Angabe (z.B. BO, Artikel, Rechnung, ...)	"String"	ja, sofern abweichend vom Default
relation	Handelt es sich um eine Relation des Typs `n-1` oder `1-n`	-	ja, sofern es sich beim Typ um ein MyTISM-Objekt handelt
readonly	Ist das virtuelle Attribut beschreibbar? Wird eine setter-Methode explizit definiert impliziert dies ein *readonly="true"*	"false"	siehe Beschreibung
cached	Bestimmt die Cachingstrategie des Ergebnisses. Siehe Abschnitt "cached"	Deaktiviert	nein
default	Ein Groovy-Ausdruck, der den Standardwert definiert; nur sinnvoll für non-readonly vattrs; siehe Abschnitt "default"	-	nein

Außerdem kann man ein Unterelement namens `init` mit einem Groovy-Skript verwenden, das beim erstmaligen Zugriff (get, set, add, remove) auf diese `Virtual Property` einer Objektinstanz ausgeführt wird; siehe Abschnitt "init"

Beispiele für Virtual Properties

Virtual Property in einem Lesezeichen:

```
<Table entity="Rechnung">
  <virtualProperty entity="Rechnung" name="PostenAnzahl">
    <get>bo.Posten.size()</get>
  </virtualProperty>
  <Query type="Text"/>
  <View>
    <Column property="BelegNr" sort="DESC" sortLevel="2"/>
    <Column property="Wartend"/>
    <Column property="Adressat.AbstraktePerson" title="Kunde"/>
    <Column property="Belegdatum"/>
    <Column property="GesamtSumme"/>
    <Column property="Waehrung"/>
    <Column property="PostenAnzahl"/>
  </View>
</Table>
```

`Virtual Properties` in einem Formular:

```
<View>
  <virtualProperty entity="BX" name="Scanzeile">
    <set>
      if (value == null) {
        return
      }
      command = value.substring(0, 1)
      param = value.substring(1)
      switch (command.toUpperCase()) {
        case 'M':
          println 'M-Nr'
          bo.LogInfo = "Kommando $command"
          // hier ggfs. Code zur Verarbeitung der M-Nr
          break
        case 'S':
          println 'S-Nr'
          bo.LogInfo = "Kommando $command"
          // hier ggfs. Code zur Verarbeitung der S-Nr
          break
        default:
          bo.LogInfo = "Error: unbekanntes Kommando \"$command\""
      }
    </set>
  </virtualProperty>
  <virtualProperty entity="BX" name="LogInfo" readonly="false"/>
  <!-- Formular-Definition -->
  <Element>
    <Text property="ScanZeile" align="CENTER" fontStyle="bold">
      <Action cmd="beep" accKey="ENTER" shortDescription="keep focus after enter key
here">
        <onAction>ftx.sync()</onAction>
      </Action>
    </Text>
  </Element>
  <Element label="LogInfo">
    <Text property="LogInfo"/>
  </Element>
</View>
```

Quellcode 2:

```
<element>
   <einElement>inhalt</einElement>
   <nochEinElement/>
   <!-- War: <element attribut="zwei"/> -->
   <Include name="codebaustein" attrWert="zwei"/>
   <wiederumEinElement attr="wert"/>
</element>
```

Nicht alle Bereiche eines Strukturelements können auf Virtual Properties zugreifen. Beispielsweise ist dies innerhalb einer <enabledOn> - Bedingung einer Action nicht möglich. FIXME Complete and/or correct the list of sections where virtual properties can/cannot be accessed.

Caching

Manche "virtualProperties" sind so aufwändig zu berechnen, dass es sich lohnt das Ergebnis zu cachen. Ein einfaches Caching über transientProperties (FIXME Erklärung?) existiert in vielen Projekten, ist üblicherweise aber nicht synchronisiert. Dadurch werden teure Berechnungen und Queries parallel mehrfach ausgeführt, was sowohl hohen Netzwerktraffic als auch Serverlast verursachen kann. Der dazu nötige Boilerplate-Code verringert zudem die Wartbarkeit.

Daher gibt es eine Standardmöglichkeit, Werte solcher Properties zur mehrfachen Verwendung zu speichern. Diese wird bei der Definition mit dem XML-Attribut `cached` aktiviert.

Mögliche Cachemodi

false *oder* NONE

Der Wert wird nicht im Cache gespeichert, der Getter wird bei jedem Aufruf erneut berechnet. Entspricht dem **Default**-Verhalten, falls das `cached` Attribute weggelassen wurde.

true *oder* VERSIONED

Der zurückgegebene Wert ist nur für die aktuelle BO-Version gültig. Wird das BO verändert, dann wird der Getter erneut aufgerufen und ein aktualisierter Wert berechnet. Für alle Strukturelemente empfehlenswert.

SIMPLE

Der zurückgebene Wert ist dauerhaft gültig und wird (praktisch gesehen) über die Lebensdauer des BOs nicht neu berechnet. Er wird erst verworfen, wenn der komplette Cache des "scripted Attribute" geleert wird, was z.B. durch Leeren des CachingBOLoader-Caches geschieht.

Dieser Modus sollte nur verwendet werden, wenn der berechnete Wert garantiert unabhängig vom aktuellen Zustand des BOs ist.

Neuberechnung bei `true` oder `VERSIONED`

Der Cache für `VERSIONED`-Attribute wird invalidiert, wenn am BO ein `bumpVersion()` aufgerufen wird. Änderungen an anderen, z.B. vom BO referenzierten, BOs sind da egal.

Das passiert in den folgenden Fällen automatisch:

- Ein persistentes Attribut am BO wird geändert
- FIXME add/remove vermutlich auch, aber ist nicht ganz sicher
- Ein "scriptedAttribut" am BO wird geändert. Das triggert zum einen den `TransactionMessageQueue` und macht auch ein `bumpVersion()`, selbst wenn eigentlich nichts am BO selbst geändert wird.
- BO#bumpVersion = invalidiere `VERSIONED`-Caches
- BO#notifyMessageBus = invalidiere `VERSIONED`-Caches und stelle sicher das sich Tabellen etc. aktualisieren

`cached`-Angabe direkt im Schema

Es ist auch möglich, direkt bei der Definition eines Attributes im Schema `cached` anzugeben.

Die Schema-Variante speichert den Cache als "transientProperty" am BO und macht ein Synchronize beim Lesen und Schreiben. Die "scriptedAttribute" Variante (s.o.) speichert das am ScriptedAttribute intern, benutzt Futures und ist insgesamt Multi-Threading-performanter.

Bei beschreibbaren Schema-vattrs muss man das Event in der MessageQueue selbst erzeugen. Dafür aktualisieren sich dann Tabellen auch korrekt.

Positiv-Beispiel

☺ Beispiel:

```
<virtualProperty entity="BO" name="TeureSumme" type="Long" cached="true"> ①
  <get>return bo.BOLoader.queryBO("sum(Id) from BO a where not Ldel").find()</get> ②
</virtualProperty>
<virtualProperty entity="BO" name="TeurerSummeNull" type="Boolean">  ③
  <get>return bo.TeureSumme == null</get>
</virtualProperty>
```

① `cached="true"` aktiviert das automatische Caching im Modus 'VERSIONED'.

② Sollte noch kein BO in der Datenbank existieren, dann ist die Summe 0 und find() gibt *null* zurück. Der Getter enthält nur die reine Berechnung des Wertes.

③ Eine einfach zu berechnende Property hängt vom Ergebnis der teuren Berechnung ab. Diese muss nicht unbedingt als cached markiert sein.

Negativ-Beispiel

Die Semantik des cached-Flags **SIMPLE** entspricht in etwa der folgenden Implementierung, mit einigen wichtigen Vorteilen:

1. Die Synchronisation kann viel enger gefasst werden, wodurch verschiedene virtualProperties des gleichen BOs parallel berechnet werden können.
2. *null*-Werte werden korrekt im Cache gespeichert.
3. Standardmässige Versionierung, d.h. der Wert wird nach Benutzereingaben automatisch aktualisiert.

😐 Don’t do this, just for reference!

```
<virtualProperty entity="BO" name="TeureSumme" type="Long">
  <get><![CDATA[
    def cacheValue = bo._TeureSumme ①
    if (cacheValue <> null) {
      return cacheValue
    }
    synchronized (bo) {
      cacheValue = bo._TeureSumme ②
      if (cacheValue <> null) {
        return cacheValue
      }
      cacheValue = bo.BOLoader.queryBO("sum(Id) from BO a where not Ldel").find() ③
      bo._TeureSumme = cacheValue ④
    }
    return cacheValue
  ]]></get>
</virtualProperty>
```

① Prüfe auf existierenden Cache, entspricht einem synchronisiertem Zugriff auf die Transient Property Map des BOs.

② Erneuter Check nötig, evtl. wurde dieser Thread am synchronized aufgehalten während ein anderer das Ergebnis bereits berechnet hat.

③ Die eigentliche Berechnung…

④ bzw. `bo.setTransientProperty('_TeureSumme', cacheValue, true)` für Versionierung.

Standard-Werte

Schreibbare `Scripted Attributes` können einen Standardwert zugewiesen bekommen. Der Standardwert wird über einen Groovy-Ausdruck im Attribut `default` des virtualProperty-Elements angegeben.

```
<virtualProperty entity="BO" name="Name" readonly="false" default="'Grumpy Cat'"/> ①

<virtualProperty entity="BO"
  name="AnswerToTheUltimateQuestionOfLifeTheUniverseAndEverything" type="Long"
  readonly="false" default="42"/> ②

<virtualProperty entity="BO" name="Einheit" readonly="false" type="Einheit"
relation="n-1"
  default="Einheit.forMeter(bo.BOLoader)"/> ③
```

① Der initiale Wert des `ScriptedAttributes` vom Typ `String` wird auf den Wert "Grumpy Cat" gesetzt.

② Der initiale Wert des `ScriptedAttributes` vom Typ `Long` wird auf den Wert 42 gesetzt.

③ Als initialer Wert für das `ScriptedAttribute` vom Typ `Einheit` wird das Initialdaten-Objekt, das "Meter" repräsentiert, via BOLoader des BOs besorgt.

Initialisierungsskript

`Scripted Attributes` können ein `init`-Unterelement haben, das ein Groovy-Skript enthält, welches beim erstmaligen Zugriff (get, set, add, remove) auf diese `Virtual Property` einer Objektinstanz ausgeführt wird. Typischerweise kann ein solches Skript Datenstrukturen oder Caches initialisieren. Das Skript kann theoretisch auch den Wert einer `Virtual Property` setzen und damit ggfs. den Wert, der über den Ausdruck im `default`-Attribut gesetzt wurde, wieder überschreiben; dies wird jedoch als Warning im Client-Log vermerkt.

```
<virtualProperty entity="BO" name="Name" readonly="false">
  <init>bo.Name = 'Happy Dog'</init> ①
</virtualProperty>

<virtualProperty entity="BO"
  name="AnswerToTheUltimateQuestionOfLifeTheUniverseAndEverything" type="Long"
  readonly="false">
  <init> ②
    bo.initNutrimaticDrinksDispenser()
    bo.resetInfiniteImprobabilityDrive()
    bo.applyThinkingCap()
    bo.assureTowel()
  </init>
</virtualProperty>
```

① Der initiale Wert des `ScriptedAttributes` vom Typ `String` wird per init-Skript auf den Wert "Happy Dog" gesetzt.

② Essentielle Initialisierungsroutinen zur Berechnung des Werts des `ScriptedAttributes` vom Typ `Long` werden durchgeführt. = Troubleshooting - Probleme und (hoffentlich) deren Lösungen

FIXME: wie man Fehler meldet (Weg, Inhalt (was wir wissen muessen), ...

Probleme beim Start des Clients

`de.ipcon.tools.IRuntimeException: Vergroesserung des Pools fehlgeschlagen, IOException aufgetreten: Malformed reply from SOCKS server`

Beim SOCKS Server handelt es sich um einen Proxy-Server. Stellen Sie sicher, dass im JavaWebstart die Option "Direktverbindung" eingestellt ist anstelle einen Proxy-Server zu verwenden. Desweiteren öffnen Sie "/Start/Einstellungen/Systemsteuerung/Internetoptionen" und wechseln dort auf den Reiter "Verbindungen". Im unteren Drittel befinden sich die "LAN-Einstellungen". Dort deaktivieren Sie bitte ALLE Checkboxen und schliessen den Dialog mit OK.

FAQ - Immer wiederkehrende Fragen und deren Beantwortung

Benutzer-Passwort ändern / Change user password / Changer mot de passe

Benutzer-Passwort ändern

Nach der Anmeldung finden Sie im linken Menü-Baum ganz oben einen Eintrag mit Ihrem Login-Namen. Wenn Sie auf diesen Eintrag mit der rechten Maustaste klicken, öffnet sich ein Kontext-Menü, aus dem Sie den Eintrag *Information* auswählen. Es öffnet sich ein Formular, in dem Sie ein neues Passwort setzen können.

Change user password

After the login you will find in menu tree (left side) at the top an entry with your login name. Right-clicking on this entry will open a context menu where you can choose the entry *Information*. A form will open where you can change your password.

Changer mot de passe

Après la connexion vous trouvez dans la navigation (côté gauche) en haut l'entrée de votre nom d'utilisateur. Faites un clique droit sur votre nom d'utilisateur pour ouvrir un menu où vous choisissez l'entrée *Informations*. Un formulaire vous permet de changer votre mot de passe.

JavaWebstart-Cache löschen unter Windows

Geben Sie unter START / AUSFÜHREN folgenden Befehl ein und drücken RETURN

```
javaws -viewer
```

Es öffnet sich das “Java Control Panel” und evtl. sogar direkt schon “Java Cache Viewer”

Wenn sich der Cache Viewer nicht öffnet, dann klicken Sie im Control Panel bei “Temporäre Internetdateien” auf “Anzeigen”

Markieren Sie im “Java Cache Viewer” die jeweilige Anwendung (einmal klicken) und löschen Sie diese dann durch Anklicken des grossen roten “X” in der Menüzeile des Cache Viewers.

Schliessen Sie den "Java Cache Viewer" und das "Java Control Panel" und versuchen Sie sich erneut anzumelden, indem Sie die Anwendung erneut herunterladen.

Anzeige der Symbole auf SVGs umstellen

Um die Verwendung von Symbolen in Vektorgraphik zu aktivieren, erstellen Sie - falls noch nicht vorhanden - eine Variable vom Typ `Boolean` mit dem Namen `theme.useSVGIcons` und erstellen Sie anschließend eine EinstellungenVariable für diese Variable mit dem Wert `true`. Anschließend müssen Sie den Solstice-Client neu starten.

Der Windows-Task-Manager zeigt mehr verwendeten Speicher an als der About-Dialog von MyTISM

Wenn ein Java-Prozess gestartet wird, fordert die JVM die Speichermenge an, die in der Option -Xmx in den an den Java-Prozess gelieferten VM args angegeben ist. Dieser Gesamtspeicher wird von Windows für die JVM "reserviert", aber bis er verwendet wird, wird er zunächst nicht zugewiesen. Dies ist die "Memory (Private Working Set)"-Nutzung, die man im Task-Manager (unter Details) sieht.

Die Diskrepanz kommt daher, dass unser Report im About-Dialog nur die Heap-Usage anzeigt, d.h. die "Used"-Angabe stellt eine Annäherung an die Gesamtmenge an Speicher dar, die derzeit für Objekte verwendet wird, gemessen in Mebibytes. Das wird ausgerechnet aus der Differenz aus dem "total" memory (= Die Gesamtmenge an Speicher, die derzeit für aktuelle und zukünftige Objekte verfügbar ist) und dem "free" memory (=eine Annäherung an die derzeit für zukünftige zugewiesene Objekte verfügbare Gesamtmenge an Speicher).

Der Speicher der JVM teilt sich in der Regel auf diese Bereiche auf:

- Heap-Speicher, der für Java-Objekte vorgesehen ist (hier würde sich i.d.R. auch ein memory-leak zeigen).
- Nicht-Heap-Speicher, d.h. der Ort, an dem Java geladene Klassen und Metadaten sowie den JVM-Code speichert.
- Nativer Speicher, d.h. Speicher, der für dll's und nativen Java-Code (sehr niedrige Ebene) reserviert ist.

Der Windows Task-Manager zeigt dies nicht an. Er zeigt nur den gesamten von der Anwendung verwendeten Speicher an (Heap + nicht-Heap + nativer Teil).

Normalerweise ist den Prozessen, die mehr Speicher vom Betriebssystem anforderten, der Speicher auch dann noch zugewiesen, wenn die eigentliche Anwendung den Speicher schon wieder "freigegegen" hat. Die entsprechenden Speicherseiten sind von Windows als Teil des Adressraums des Prozesses abgebildet worden. Es obliegt Windows, die Working Set Size wieder zurückzufahren, nachdem Java den Speicher wieder freigegeben hat, was es allerdings nicht unbedingt "sofort" tut. Im Task-Manager nimmt der Speicher also nicht unbedingt immer ab, aber das deutet dann nicht zwingend auf ein Speicherleck in der Anwendung hin.

Allerdings könnte es helfen, die Applikation kurz zu minimieren, dann passt Windows die Working Set Size normalerweise nochmal an den tatsächlichen Bedarf an. (Quelle)

Fazit: die Speichernutzung, die der Task-Manager zeigt spiegelt nicht unbedingt das wider, was das Programm aktuell verwendet. Maßgeblich ist eher das, was im Dialog in MyTISM zu sehen ist an (Heap-)Speicher. Das sollte irgendwann bzw. beim Schließen aller Fenster und Löschen der Caches abnehmen und wenn nicht, deutet das dann wirklich auf ein Speicherleck hin.

Der TaskManager ist allgemein - gelinde gesagt - nicht so toll. Wir empfehlen daher die Verwendung des Process Explorers aus der Microsoft Sysinternals Suite, der i.A. bessere Ergebnisse zeigt (wenn auch in diesem Fall nicht, da der Heap auch dort nicht getrennt ausgewiesen wird).