



Das Administrator-Handbuch

Version: Version 0.27.32 - 2011-12-12

Inhaltsverzeichnis

Vorwort	iii
1. Erste Schritte mit MyTISM	1
1.1. Einrichtung einer kompilierfähigen MyTISM-Umgebung unter Windows (15.07.2005)	1
1.2. Kompilieren eines MyTISM-Projekts	2
1.3. Starten eines MyTISM-Servers	3
1.4. Umziehen eines syncenden MyTISM-Servers auf eine andere Hardware	3
1.5. Erneut-Aufsetzen eines syncenden MyTISM-Servers aus einem Backup des autoritati- ven Servers	4
1.6. Starten eines MyTISM-Clients (SOLSTICE)	4
1.7. Die ".checked*" -Dateien	5
1.8. Die ".init*" -Dateien	5
1.9. Services	6
2. Rechteverwaltung	9
2.1. Grundlagen	9
2.2. Benutzer	9
2.3. Gruppen	10
2.4. BOMasken	10
2.5. RechteZuweisungen	12
2.6. Noch ein Beispiel	12
2.7. Grundlegende benötigte Rechte	13
3. Fehlerbehebung	18
3.1. Doppelte IDs in der Datenbank korrigieren	18

Vorwort

MyTISM ist plattformunabhängiges, objektorientiertes, dezentrales, multiuserähiges, individuell anpassbares und quelloffenes 3-Tier-Datenbank- und Anwendungs-Framework incl. GUI und Web-Application-Server, entwickelt und betreut von OAshi s.a r.l.

In diesem Handbuch finden Sie alle Informationen, die Sie für das Aufsetzen, Warten und Administrieren eines MyTISM-Systems benötigen.



Anmerkung

Beachten Sie bitte, dass sich dieses Dokument noch im Aufbaustadium befindet und noch grosse Lücken aufweist, die wir natürlich nach und nach füllen werden.

Bei Fragen, Problemen oder Anregungen, sei es bzgl. MyTISM selber oder dieser Dokumentation, wenden Sie sich bitte an uns; Kontaktinfos finden Sie im WWW unter <http://www.mytism.de/mytism/contact>.

Kapitel 1. Erste Schritte mit MyTISM

1.1. Einrichtung einer kompilierfähigen MyTISM-Umgebung unter Windows (15.07.2005)

Im folgenden Text wird auf die einzelnen Schritte der Installation des PostgreSQL-Datenbank-Servers und des Java-Developer-Kits unter Windows eingegangen.

1.1.1. PostgreSQL



Achtung

Die Installation sollte als angemeldeter Administrator durchgeführt werden, da Installationsprobleme mit Benutzer-Konten berichtet wurden, die der Administrator-Gruppe "nur zugeordnet" waren.

Am Anfang steht der Download des Installations-Archivs. Dieses kann von

```
http://www.postgresql.org/ftp/win32
```

bezogen werden (sollte der Link irgendwann nicht mehr stimmen, gehe man direkt auf

```
http://www.postgresql.org
```

und klicke sich bis zum Windows-Download durch). Hier wählt man das ZIP-Archiv `postgresql-8.0.3.zip` zum Download aus.

Das heruntergeladene Archiv muss entpackt werden. U.a. liegen dann zwei Dateien auf der Festplatte die mit `postgres` beginnt und die Endung `.msi` hat - hiervon ruft man die kleinere Datei auf zum Starten der PostgreSQL-Installation.

Die Installationsabfolge ist weitestgehend unspektakulär, so dass nur auf einige "Spezialitäten" eingegangen wird.

Zu Beginn wählt man lediglich die Installationssprache aus, mit der man durch den weiteren Installationsverlauf geleitet wird.

Im Dialog "Installations-Optionen" stellt man ein, dass alle Sprachen installiert werden sollen und setzt das Installationsverzeichnis auf `c:\db`

Installationsoptionen-Dialog der PostgreSQL-Installation

Im Dialog "Dienste-Konfiguration" ändert man den Namen des Dienstes auf `postgres` und vergibt KEIN Passwort. Den folgenden Dialog, in dem man gefragt wird ob das fehlende Benutzerkonto angelegt werden soll, bestätigt mal - das anschliessend angezeigte Passwort kann man sich notieren, muss man aber nicht.

Im Dialog "Datenbank-Cluster initialisieren" würde man theoretisch das Encoding auf `UTF-8` umstellen. Da es damit aber unter Windows noch ein paar Problemchen gibt, lässt man das einfach auf `SQL-ASCII` stehen.



Achtung

Wenn man auf der Kommandozeile per `createdb` eine neue Datenbank anlegt muss man jetzt aber daran denken das Encoding auf UTF-8 zu stellen.

```
createdb -U postgres -E UTF-8 DBNAME
```

Im gleichen Dialog setzt man das Passwort auf `postgres`

Alle weiteren Dialoge kann man einfach bestätigen.

Nachdem die Installation abgeschlossen ist, muss noch eine Einstellung in der Datei `c:\db\data\pg_hba.conf` vorgenommen werden: am Ende der Datei in der nicht auskommentierten Zeile den Text `md5` durch `trust` ersetzen. An dieser Stelle wird auch ersichtlich warum man während der Installation ein doch vermeintlich schwaches Passwort wählen konnte - es werden nämlich eh nur Verbindungen direkt von der lokalen Maschine (127.0.0.1) akzeptiert und durch die Angabe von `trust` erspart man sich die Passwort-Abfrage.

Folgende Tuning-Massnahmen an der Datei `c:\db\data\postgresql.conf` sind nicht zwingend nötig, bringen aber doch einiges an Performance-Gewinn:

```
fsync = true
wal_buffers = 2000
commit_delay = 10000
commit_siblings = 500
```

Damit diese und obige Änderung wirksam werden, muss der PostgreSQL-Server "durchgestartet" werden.

```
net stop postgres
net start postgres
```

1.1.2. Java Developer Kit (JDK)

Auch hier muss man sich erst einmal das Installations-Archiv von <http://java.sun.com> besorgen (dort dann in der Download-Sektion die aktuellste JDK als "Windows-Offline-Version" herunterladen. Die Installation wird per Doppelklick gestartet und verläuft recht unspektakulär. Man sollte sich lediglich das Installationsverzeichnis merken (meist etwas in der Art von `c:\Programme\Java\jdkxxxx`

1.2. Kompilieren eines MyTISM-Projekts

Hierfür werden natürlich die Sourcen benötigt, die man mittels des Programms SmartCVS aus dem CVS-Repository abrufen kann. Der Download des Programms ist auf <http://www.smartcvs.com> kostenfrei möglich.

1.2.1. Auschecken aus dem CVS-Repository mit SmartCVS

Zunächst muss man im Repository Manager das Repository definieren. Dazu verwendet man folgende Daten:

```
Access Method: sserver
```

```
Server Name: cvs.mytism.de
Repository Path: cvs
```

Zum Auschecken aus dem CVS per SmartCVS wählt man nun im Menü "Project" den Eintrag "Checkout" und gelangt zu einem Wizard. Dort trägt man folgende Werte ein:

```
Repository: {das eben definierte Repository}
Module(s): nrx
Local Directory: {das Verzeichnis, in das ausgecheckt werden soll; ein Verzeichnis nrx wird dort automatisch angelegt}
Checkout options: Default (keep sticky)
Project Name: nrx
Text File Encoding: Cp1252
Text File Encoding in Repository: ASCII
```

Zum Kompilieren wechselt man unterhalb des nrx-Verzeichnisses in das entsprechende Projekt-Verzeichnis und startet die Kompilierung mit folgendem Befehl:

```
j -ant
```

Das Ergebnis der Kompilierung liegt im Build-Verzeichnis (wo dieses zu finden ist, steht im Build-File des jeweiligen Projektes in der Variablen `build.dir`).

Sofern nicht bereits geschehen muss in PostgreSQL natürlich noch eine Datenbank angelegt werden. Dies macht man mithilfe des zentralen mytism-Scripts im Build-Verzeichnis.

```
./mytism init_db
```

1.3. Starten eines MyTISM-Servers

Sofern alles klappt ist unter Linux mit dem Aufruf von

```
./mytism start
```

aus dem Build-Verzeichnis heraus alles erledigt.

1.3.1. Unmittelbar nach Aufruf von `./mytism start` kommt die Fehlermeldung `"bash: ./server: Keine Berechtigung"`

Die Datei "mytism" ist aufgrund des fehlenden Executable-Flags nicht ausführbar. Dieses setzt man man unter Linux mittels

```
chmod 755 DATEINAME
```

1.4. Umziehen eines syncenden MyTISM-Servers auf eine andere Hardware

FIXME



Achtung

Die BTs müssen in jedem Fall vollständig bleiben und dürfen auf keinen Fall geflusht werden, wenn man eine Instanz umzieht.

Grund ist, dass der SyncChecker sonst nicht prüfen kann, welche Transaktionen evtl. lokal noch aufgrund von Timing-Problemen beim Syncen (lang dauernde Speichervorgänge auf dem Server, von dem gesyncet wird, die später vom SyncChecker nachgezogen werden) fehlen.

1.5. Erneut-Aufsetzen eines syncenden MyTISM-Servers aus einem Backup des autoritativen Servers

FIXME



Achtung

Beim Neu-Aufsetzen von Knoten aus einem aktuellen Backup, das bereits alle alten Daten des erneut aufgesetzten Servers enthält, muss nichts besonderes beachtet werden. Hierbei geht man am besten so vor, dass man die syncende Instanz alle lokalen Änderungen zunächst auf den autoritativen Server syncen lässt. Danach fährt man den neu aufzusetzenden Server herunter und zieht erst dann ein frisches Backup vom autoritativen Server, mit dem man dann den syncenden Server neu aufsetzt wie oben beschrieben.

Im Gegensatz dazu muss beim Neu-Aufsetzen von Knoten aus einem alten Backup, das nicht alle alten Daten des erneut aufgesetzten Servers enthält, eine neue Node-Nummer vergeben werden.

Hintergrund: Wenn die BN gleich bleibt, so werden beim Sync alle Transaktionen übersprungen, die von diesem Knoten stammten (da diese ja "eigentlich" auf dem Knoten vorhanden sein müssten, dieser war ja der Ursprung der Transaktionen / Daten).

Am einfachsten kann man eine neue Node-Nummer setzen, indem man die Nodenummer in der mytism.ini einfach leer lässt. Es wird dann bei der ersten Verbindung mit dem autoritativen Server automatisch eine Nummer vergeben und in die mytism.ini eingetragen.

1.6. Starten eines MyTISM-Clients (SOLSTICE)

Der Client lässt sich per Kommandozeile aus dem Build-Verzeichnis starten

```
java -jar deploy/XXX-Client.jar localhost
```

oder per JavaWebStart

```
/usr/lib/java/jre/javaws/javaws "http://localhost:8080/deploy/"  
// statt localhost kann auch eine IP-Adresse angegeben werden
```

Das Logfile bzw. Meldungen des Client werden beim Start von der Kommandozeile in der jeweiligen

Shell ausgegeben, in der der Client gestartet wurde. Ausserdem wird im Temp-Verzeichnis (unter Linux /tmp/) eine Log-Datei namens "client-log.txt" angelegt.



Anmerkung

Unter Windows-Systemen wird die heruntergeladene Anwendung (Datei mit der Endung ".jnlp") im Profil gespeichert, was bei Systemen, deren Profil auf einem zentralen Server liegt den An- bzw. Abmelde-Prozess verlangsamen kann. Im JavaWebStart lässt sich der Speicherort der heruntergeladenen Anwendung festlegen. Hier wählt man dann einen geeigneteren Speicherort aus.

1.7. Die ".checked*" -Dateien

Das Vorhandensein dieser Dateien signalisiert dem Server, dass bestimmte (normalerweise eher recht lange dauernde) Überprüfungen bereits durchgeführt wurden.

.PROJEKT/.checked-firstnodestart

FIXME (findet sich nur auf synchronisierenden Servern)

.PROJEKT/.checked-initialdata

Sind die nötigen Benutzer und Gruppen da, Auto-Formulare bauen, usw.

.PROJEKT/.checked-integrity

Überprüfung der Datenintegrität, z.B. auf nicht referenzierte BOs

.PROJEKT/.checked-metadata

Vergleich Schema gegen SQL-Definitionen (Tabellen, usw.)

.PROJEKT/.checked-sync

FIXME (findet sich nur auf synchronisierenden Servern)

Um eine der Überprüfungen (nochmal) ablaufen zu lassen, z.B. nach dem Einspielen einer Datenbank-Sicherungskopie, einfach die entsprechende(n) Datei(en) löschen; nach dem Neustart des Servers wird die entsprechende Überprüfung dann durchgeführt.

1.8. Die ".init*" -Dateien

FIXME (Erklärung)

.PROJEKT/.init-keygen

FIXME (findet sich nur auf synchronisierenden Servern)

.PROJEKT/.init-streamcopy

FIXME

.PROJEKT/.init-syncaccount

FIXME (findet sich nur auf synchronisierenden Servern)

FIXME - Was darf wann und zu welchem Zweck gelöscht werden?

1.9. Services

FIXME Einleitende Erklärung zu BN, BU, BS, Skript-Service, Import-Service

1.9.1. BusinessNode (BN)

1.9.1.1. Wozu dienen BNs?

FIXME

1.9.1.2. Wie legt man sie an?

Seit dem Core-Codestand vom 16.11.2009 werden initiale BNs automatisch erstellt.

Wenn keine `nodeNumber/nodeID` eingetragen ist (weder in `mytism.ini` noch in `.init-syncaccount` - wobei Letzteres eigentlich nicht auftreten kann, da dann bereits vorher ein Fehler auftritt) wird beim Serverstart automatisch eine (zum Hostnamen des Server-Rechners passende) BN gesucht (falls keine solche existiert, angelegt) und ein entsprechender "nodeNumber"-Eintrag in der `mytism.ini` eingefügt. Genaueres siehe `DBMan.assureServerBN()`.

Normalerweise wird eine BN für Server oder `SyncService` nur über `nodeNumber/nodeID` gefunden. Wenn allerdings beim Aufruf des `SyncService` keine Datei `.init-syncaccount` existiert bzw. wenn beim Start des `DBMan` keine `nodeNumber/nodeID` gefunden werden konnte, wird erst eine BN mit `Name = misc.getHostname()` gesucht und ggf. dann diese benutzt (und auch deren `Id` als `nodeNumber/nodeID` automatisch in die jeweilige Konfigurationsdatei eingetragen, d.h. das passiert nur einmal). Ansonsten spielt der Name intern AFAIK keine Rolle.

1.9.2. BusinessUnit (BU)

1.9.2.1. Wozu dienen BUs?

Mittels BUs kann man Node-bezogen z.B. Nummernkreise o.ä. realisieren.

1.9.2.2. Wie legt man sie an?

Man findet unter `/Admins/MyTISM/Interna/BUs` das Lesezeichen und die Schablone.

Will man z.B. eine BU zur automatischen Erstellung von fortlaufenden Rechnungsnummern anlegen, erstellt man mittels der BU-Schablone und trägt für die einzelnen Felder folgende Werte ein:

- *Name*: `de.oashi.bo.Rechnung.BelegNr`
- *Beschreibung*: BU für Rechnungsnummern
- *Next*: `240113`
- *Min*: `240000`
- *Max*: `999999`

- *Increment*: 1
- *Valid*: TRUE
- *Node*: hier die BN auswählen, für die die BU gelten soll

Obige BU liefert uns nun in 1er-Schritten (Increment) Rechnungsnummern. Begonnen wurde mal mit der Rechnungsnummer 240000 und wenn die Rechnungsnummer 999999 erreicht ist, ist Schluss. Als nächster Wert würde die 240113 vergeben werden. Falls mal was schiefgehen sollte und man "aus Versehen" eine Rechnungsnummer gezogen hat, kann man durch anpassen des "Next"-Wertes korrigierend eingreifen. "Valid" besagt lediglich ob die BU aktiv ist oder nicht.

Name und Beschreibung können prinzipiell willkürlich gewählt werden, doch es bietet sich an den Namen so zu wählen, dass man schnell erkennen kann, wo die von der BU generierte Nummer verwendet wird. In unserem Bsp. also in der Entität "Rechnung" und dort im Attribut "BelegNr".

1.9.2.3. Wie benutzt man sie?

Um die Server-seitige Vergabe von, um bei unserem Beispiel zu bleiben, Rechnungsnummern zu aktivieren, muss man in der jeweiligen BO-Klasse das SaveVeto-Interface implementieren (Beispiele hierfür finden sich in ausreichender Menge im Sourcecode).

```
method verifyOnServer(nodeNumber=Long, user=Benutzer, tx=Transaction)
  if cancelRecalc() then
    return
  super.verifyOnServer( nodeNumber, user, tx )
  if \ getWartendNN() & getBelegNr() == null then
    -- eindeutige BelegNr ziehen und zuweisen
    setBelegNr( BU.nextValueAsString( getClass().getName().BelegNr', nodeNumber, user, tx, getBOLoader() ) )
```

In obigem Beispiel ziehen wir uns nur eine Rechnungsnummer, wenn die Rechnung nicht auf "wartend" steht und noch keine Rechnungsnummer (BelegNr) hat.

1.9.3. BusinessService (BS)

1.9.3.1. Wozu dienen BSe?

Um z.B. Scripte ständig oder wiederkehrend auszuführen

1.9.3.2. Wie legt man sie an?

Man findet unter /Admins/MyTISM/Interna/BSs das Lesezeichen und die Schablone.

Will man z.B. einen BS zur automatischen Erstellung von Erinnerungs-Mails anlegen, erstellt man einen neuen Service mittels der BS-Schablone und trägt für die einzelnen Felder folgende Werte ein:

- *Name*: Erinnerung an 'etwas'
- *Beschreibung*: BS für Erinnerungen
- *Java-Klasse*: de.ipcon.db.sync.ScriptService

- *Parameter:*

```
<Sync>
  <mytism-connection url="socket://localhost" user="Benutzer eintragen" pass="einPasswort"/>
  <script language="groovy">
    tx = api.getNewTx()
    absender = "support@oashi.com"
    mailserver = "smtp.gmail.com"
    authMethod = de.ipcon.messaging.email.SimpleMailSender.AM_SMTPAUTH
    betreff = "Erinnerung"
    text = "Erinnerung an ETWAS!!"
    sm = new de.ipcon.messaging.email.SimpleMailSender( absender, mailserver, authMethod, "jemand", "passw
    email = "me@domain.com"
    sm.sendMail( email, text, betreff, absender )
  </script>
</Sync>
```

- *Aktiv:* TRUE

- *Ausführungs-Vorschrift:*

```
<ExecutionPolicy>
  <CronJob interrupt="false">
    Wenn 'interrupt' (optional) auf true steht, wird ein bereits laufender Prozess des gleichen Jobs bei
    <Commands>
      Die Syntax orientiert sich an der unix crontab.
      <Command>10 15 * * * </Command>
    </Commands>
  </CronJob>
</ExecutionPolicy>
```

oder

```
<ExecutionPolicy>
  <ExecutionPolicyKeepRunning/>
</ExecutionPolicy>
```

- *Nodes:* hier die BN auswählen, für die der BS gelten soll

Name und Beschreibung können prinzipiell willkürlich gewählt werden

1.9.3.3. Wie benutzt man sie?

FIXME

1.9.4. SkriptServices

FIXME Einleitende Erklärung

Kapitel 2. Rechteverwaltung



Anmerkung

Alle genannten Formulare, Schablonen und Lesezeichen liegen normalerweise im Ordner "Benutzerverwaltung". Das Wissen über die grundlegende Bedienung von Solstice setze ich mal vorraus :-)

2.1. Grundlagen

Rechteverwaltung bedeutet, dass man bestimmten Personen das Lesen, Ändern, Neuanlegen oder Löschen von bestimmten Objekten oder Daten (in MyTISM also BOs bzw. deren Attribute) erlauben oder verweigern will. Hierbei sollte man sich folgende Fragen stellen:

Wer?

Welchem "Personenkreis" will ich irgendwelche Dinge erlauben oder verbieten?

Was?

Für welche Menge von Objekten/Daten will ich Sachen erlauben oder verbieten?

Wie?

Wie sollen die Rechte aussehen, d.h. was genau soll erlaubt oder verboten werden?

Beispiel:

- *Alle Mitglieder der Geschäftsführung (Wer?) dürfen die Mitarbeiter-Daten (Was?) ansehen und ändern (Wie?).*
- *Alle anderen (Wer?) dürfen die Mitarbeiter-Daten (Was?) nur ansehen (Wie?).*

Im Folgenden soll kurz erklärt werden, wie man diese Rechteverwaltung in MyTISM realisieren kann.

Erste Voraussetzung für die Vergabe von Rechten ist natürlich, dass es auch irgend jemanden gibt, für den man irgendwelche Rechte definieren kann. In MyTISM gibt es dafür, ähnlich wie auch in vielen anderen Anwendungen, das System von Benutzern und Gruppen.

2.2. Benutzer

Ein *Benutzer* steht für eine einzelne reale oder virtuelle Person (oder genauer eigentlich: Für ein Benutzerkonto; auch wenn es nicht vorkommen sollte kann ein Benutzer(konto) im Prinzip von mehreren Personen gemeinsam - wenn auch nicht unbedingt gleichzeitig - benutzt werden). Die definierten Benutzer bestimmen, wer überhaupt auf ein MyTISM-System zugreifen darf.

Der Benutzer "Admin", der alles sehen kann und alles darf, wird für jedes MyTISM-System automatisch angelegt. Weitere Benutzer können dann je nach Bedarf hinzugefügt werden, haben allerdings dann noch keinerlei Rechte. Ein Hinweis direkt hierzu: Allein um sich überhaupt anmelden zu können, müssen Benutzer schon gewisse Leserechte bekommen! Diese hier im Einzelnen aufzuführen

wäre zu aufwändig - einfacher ist es eine "MT_ALL"-BO-Maske zu erstellen.

Das Benutzer-Formular in Solstice

Für dieses Tutorial nehmen wir mal an, dass folgende Benutzer definiert sind:

- *Admin*
- Alice
- Bob
- Claire

2.3. Gruppen

Mehrere Benutzer kann man in einer *Gruppe* zusammenfassen. Da Rechte in MyTISM nur an Gruppen und nicht an einzelnen Benutzern hängen können, muss jeder Benutzer mindestens einer Gruppe angehören.

Eine Gruppe "Admins" wird automatisch gebaut und enthält (am Anfang nur) den Benutzer "Admin".

Das Gruppe-Formular in Solstice, hier die Liste der für die Gruppe eingetragenen Benutzer

In diesem Tutorial benutzen wir folgende Gruppen:

- *Admins*
- Benutzer (enthält Alice, Bob und Claire)
- Chefs (enthält Claire)

Nachdem man mittels Benutzer und Gruppen einen "Personenkreis" definiert hat, für den man Rechte vergeben will, kann man nun nach Bedarf die einzelnen Berechtigungen definieren.

2.4. BOMasken

Um Rechte für bestimmte BOs festzulegen, muss man die entsprechenden BOs natürlich irgendwie auswählen. Dies geschieht mit Hilfe der sog. *BOMasken*; damit definiert man eine Menge von BOs, für welche man dann einer oder mehreren Gruppen bestimmte Rechte erlauben oder verweigern will.

Nehmen wir an, wir haben BOs vom Typ "Kunde" in unserer Datenbank:

- Orange Computers, Inc.
- Daughters & Sons
- Lost Cargo Ltd.

- ...

Dann können wir wie folgt eine BOMaske bauen, die alle vorhandenen Kunden-BOs auswählt:

Das BOMaske-Formular in Solstice

Unter "Name" tragen wir einen passenden Namen für die BOMaske ein. Der Name kann frei gewählt werden, sollte aber irgendwie sinnvoll sein :-). In unserem Beispiel hätten wir genauso gut z.B. auch "Alle Kunden" nehmen können. (FIXME es gibt eine Namenskonvention?)

Unter Beschreibung sollte man einen kurzen Text eintragen, der in ein paar Worten "menschenslesbar" erklärt, welche BOs mit dieser Maske ausgewählt werden.

Der wichtigste Punkt ist "Entitaet": Hier bestimmt man, BOs welchen Typs diese BOMaske auswählt. Da wir in unserem Beispiel alle Kunden (also BOs vom Typ "Kunde") wollen, tragen wir hier den BO-Typ "Kunde" ein.

Wenn Sie unter Attribut auch einen Wert angeben, so bezieht sich die Maske nur auf dieses Attribut der BOs, d.h. sie können hiermit Rechte für einzelne Eigenschaften von BOs vergeben.

2.4.1. Script

"Script" ist was für Fortgeschrittene: Hier kann man ein Groovy-Skript eintragen, welches für jedes BO (vom oben unter "Entitaet" eingetragenen BO-Typ) ausgewertet wird. Nur wenn das Skript "true" zurückliefert, wird das entsprechende BO berücksichtigt. Mit dieser Methode kann man noch weitere beliebig komplizierte Einschränkungen/Auswahlen implementieren.

Es stehen folgende vordefinierte Variablen zur Verfügung:

bo

Das zu überprüfende BO.

schema

Das verwendete Schema.

entity

Die in der BOMaske angegebene Entitaet.

boclass

Die Java-Klasse die der Entitaet entspricht.

attribute

Das in der BOMaske angegebene Attribut (ggf. null).

maske

Verweis auf die BOMaske selbst.

log

Ein Logger mit Hilfe dessen (Fehler)Meldungen ins Log ausgegeben werden können.

user

Der aktuelle Benutzer.

(Eher unsinniges :-) Beispiel:

```
return bo.Id.longValue() > 1000
```

Weiteres Beispiel (IstSchoen ist ein Boolean-Attribut):

```
return bo.IstSchoenNN
```

FIXME Info zu Performance-Einbussen, etc.

Wir haben jetzt Gruppen, denen wir Rechte geben können und haben eine Möglichkeit, Mengen von BOs definieren, für welche diese Rechte gelten sollen. Was fehlt ist natürlich noch die Definition dieser Rechte.

2.5. RechteZuweisungen

In den RechteZuweisungen werden alle Komponenten zusammengeführt. Hier gibt man an, wer (Gruppen) mit was (BOMaske) wie arbeiten darf.

Das RechteZuweisungs-Formular in Solstice

Wir wollen in unserem Beispiel allen Benutzern etwas erlauben, also tragen wir unter "Gruppe" die bereits definierte Gruppe "Benutzer" ein.

"Position" wird z.Zt. noch nicht benutzt und bleibt leer.

Wir wollen Rechte für alle Kunden-BOs setzen, also tragen wir unter "Maske" unsere eben definierte BOMaske "Kunden" ein, die ja alle vorhandenen Kunden-BOs auswählt.

Jetzt können wir auch bestimmen, was die "Benutzer" mit allen "Kunden" machen dürfen. Da Änderungen an den Kundendaten Chefsache sind, erlauben wir nur das "Lesen".

Zu guter Letzt sollte man auch hier wieder einen kurzen "menschenslesbaren" Kommentar eintragen, der aussagt, was diese RechteZuweisung genau bewirkt. Dieser im Feld "Bemerkung" hinterlegte Kommentar wird auch in der Meldung angezeigt, die dem Anwender angezeigt wird, wenn er für die jeweilige Aktion keine Rechte besitzt.

Nachdem wir diese RechteZuweisung abgespeichert haben, dürfen fortan alle Benutzer die Kundendaten einsehen!

2.6. Noch ein Beispiel

Soweit, so gut, aber irgend jemand muss ja auch die Kundendaten auf dem aktuellen Stand halten. Da das Sache der Chefs ist, benötigen diese natürlich auch entsprechende Rechte d.h. wir bauen noch eine weitere RechteZuweisung:

Chefs dürfen alles!

Unter "Gruppe" wählen wir logischerweise "Chefs".

Da wir auch hier wieder etwas für alle Kunden-BOs definieren wollen, tragen wir unter "Maske" wieder unsere eben definierte BOMaske "Kunden" ein.

An Rechten vergeben wir jetzt aber wesentlich mehr, nämlich alles was möglich ist:

- Lesen - Die Chefs dürfen natürlich auch Lesen (dieses Recht hier nochmal zu vergeben ist rein theoretisch nicht nötig, da wir das Lesen ja bereits vorher für alle Benutzer, also auch die Chefs, die ja ebenfalls auch in der Gruppe "Benutzer" sein sollten, erlaubt haben).
- Schreiben - Die Chefs dürfen bestehende Kunden-BOs bearbeiten und wieder abspeichern.
- Erstellen - Die Chefs dürfen auch vollkommen neue Kunden(-BOs) anlegen.
- Löschen - Die Chefs dürfen vorhandene Kunden(-BOs) aus der Datenbank löschen.
- Ablehnen - Alle oben genannten Rechte bekommen eine umgekehrte Bedeutung, das heißt, verbieten von Lesen, Schreiben, Erstellen und Löschen - das setzen wir natürlich nicht.

2.7. Grundlegende benötigte Rechte

Mit Hilfe der Ablehnen-Rechte, die immer Priorität gegenüber allen anderen Rechten für das selbe BO haben, ist natürlich eine "Alles zulassen und Ausnahmen definieren" Strategie bis zu einem gewissen Punkt tolerabel, speziell in kleinen Installationen. Hier bietet sich z.B. an die Quertabellen über die Vererbung dem gemeinen Benutzer zum Editieren zu sperren indem man eine Gruppe "KeineQuertabellenÄndern" oder ähnliches zu erstellen und über die Rechtezuweisung ein (Ablehnen Erstellen Löschen Schreiben) zuzuweisen. Einziges Manko ist dabei die Menge der zu erstellenden Gruppen und die Menge der Ausschlüsse für einen normalen Benutzer - man merke sich: Wenn ein Ablehnen-Recht einmal definiert wurde für irgendeine Gruppe dieses Benutzers, wird es zur Anwendung kommen. Eine Aufhebung eines solchen Rechts ist (bislang) nicht vorgesehen.

In etwas größeren Umgebungen mit etwas mehr Schema und entsprechender Evolution wird dieses Verfahren müßig, weil bei Updates sofort die entsprechenden "Schotten" definiert und gebaut werden müssen, um editierfreudige Benutzer davon abzuhalten, Unheil anzurichten (meist machen diese Benutzer das ja nicht absichtlich...). Im folgenden möchte ich aufzeigen, wie man eine Rechtestruktur rein auf "Grant"-Basis erstellt, die als Grundlage für eigene Verfahren dienen kann, eine komplexe Rechtestruktur für ein großes Projekt zu erstellen.

2.7.1. Benötigte Masken

Zunächst brauchen wir ein paar Masken, die die grundlegendsten Objekte definieren, um die Rechtezuweisungen darauf aufbauend implementieren zu können:

Tabelle 2.1. Definition der Maske MT_SELF_USER

Name	MT_SELF_USER
Beschreibung	Mich selbst als Benutzer

Entität	de.ipcon.db.core.Benutzer
Script	<pre>return (user.Id.equals(bo.Id));</pre>

Tabelle 2.2. Definition der Maske MT_SELF_GROUPS

Name	MT_SELF_GROUPS
Beschreibung	Gruppen in denen ich Mitglied bin
Entität	de.ipcon.db.core.Gruppe
Script	<pre>for(gr:user.GruppenIterator){ if(gr.Id==bo.Id) return true; } return false;</pre>

Tabelle 2.3. Definition der Maske MT_USER_IN_SELF_GROUPS

Name	MT_USER_IN_SELF_GROUPS
Beschreibung	Alle Benutzer, die in Gruppen sind, in denen ich auch bin
Entität	de.ipcon.db.core.Benutzer
Script	<pre>for(gr:user.GruppenIterator){ for(usr:gr.BenutzerIterator){ if(usr.Id==bo.Id) return true; } } return false;</pre>

Tabelle 2.4. Definition der Maske MT_STRUCTURE_NO_ROOT

Name	MT_STRUCTURE_NOROOT
Beschreibung	Alle Struktur-Elements, aber nichts in der Root
Entität	de.ipcon.db.core.Struktur
Script	<pre>return bo.Elter!=null;</pre>

Tabelle 2.5. Definition der Maske MT_PRINT_REPORT

Name	MT_PRINT
Beschreibung	Was zum Drucken grundsätzlich mal gebraucht wird: Reports

Entität	de.ipcon.db.core.Report
Script	<input type="text"/>

Tabelle 2.6. Definition der Maske MT_PRINT_SUBREPORTPOSTEN

Name	MT_PRINT_SUBREPORTPOSTEN
Beschreibung	Was zum Drucken grundsätzlich mal gebraucht wird: SubreportPosten
Entität	de.ipcon.db.core.SubreportPosten
Script	<input type="text"/>

Tabelle 2.7. Definition der Maske MT_PRINT_BILDPOSTEN

Name	MT_PRINT_BILDPOSTEN
Beschreibung	Was zum Drucken grundsätzlich mal gebraucht wird: BildPosten
Entität	de.ipcon.db.core.BildPosten
Script	<input type="text"/>

Tabelle 2.8. Definition der Maske MT_PRINT_BILD

Name	MT_PRINT_BILD
Beschreibung	Was zum Drucken grundsätzlich mal gebraucht wird: Bilder
Entität	de.ipcon.db.core.Bild
Script	<input type="text"/>

Tabelle 2.9. Definition der Maske MT_PRINT_DRUCKZIEL

Name	MT_PRINT_DRUCKZIEL
Beschreibung	Was zum Drucken grundsätzlich mal gebraucht wird: Druckziele
Entität	de.ipcon.db.core.Druckziel
Script	<input type="text"/>

Tabelle 2.10. Definition der Maske MT_ALL_L10NLOCALE (nur für L10n)

Name	MT_ALL_L10NLOCALE
------	-------------------

Beschreibung	Alle L10n Locales
Entität	de.ipcon.db.core.L10nLocale
Script	<input type="text"/>

2.7.2. Die Zuweisungen

Nun können wir uns eine Gruppe bauen, die die grundlegenden Rechte zur Anmeldung an das System beinhaltet. Wir prefixen diese mit dem Kürzel RG_, damit wir sie als reine Rechtegruppe identifizieren können. Zukünftige Versionen von MyTISM werden solche Gruppen möglicherweise vorab schon präparieren.

Tabelle 2.11. Definition der Gruppe RG_Solstice_Login

Name	RG_Solstice_Login	
Beschreibung	Benutzer mit minimalen Rechten zum Login mit Solstice	
Priorität		
Voreinstellungen	<input type="text"/>	
Versteckt	gesetzt	
Rechtezuweisung #1	MT_SELF_USER	Lesen, Schreiben
Rechtezuweisung #2	MT_SELF_GROUPS	Lesen
Rechtezuweisung #3	MT_USER_IN_SELF_GROUPS	Lesen
Rechtezuweisung #4	MT_STRUCTURE_NO_ROOT	Lesen
Rechtezuweisung #5	MT_ALL_L10NLOCALE	Lesen

Eine weitere Gruppe für die Möglichkeit des Druckens wäre ebenfalls schnell konstruiert.

Tabelle 2.12. Definition der Gruppe RG_Solstice_Print

Name	RG_Solstice_Print	
Beschreibung	Benutzer mit minimalen Rechten zum Drucken mit Solstice	
Priorität		
Voreinstellungen	<input type="text"/>	

Rechteverwaltung

Versteckt	gesetzt	
Rechtezuweisung #1	MT_PRINT_BILD	Lesen
Rechtezuweisung #2	MT_PRINT_BILDPOSTEN	Lesen
Rechtezuweisung #3	MT_PRINT_DRUCKZIEL	Lesen
Rechtezuweisung #4	MT_PRINT_REPORT	Lesen
Rechtezuweisung #5	MT_PRINT_SUBREPORTPOSTEN	Lesen

Auf diesem Fundament sind schnell weitere Strukturen für den Aufbau einer komplexen und umfassenden Rechteverwaltung geschaffen.

Kapitel 3. Fehlerbehebung

3.1. Doppelte IDs in der Datenbank korrigieren

Situation: Der Integrity-Check beim Serverstart meldet doppelte IDs bzw. es ist anderweitig aufgefallen, dass in der Datenbank gleiche IDs für verschiedene Objekte mehrfach vergeben wurden (kann z.B. passieren, wenn ein Backup einer Datenbank eingespielt wurde, aber vergessen wurde, vor dem darauffolgenden Serverstart die Dateien `.init-keygen` und `.checked-firstnodestart` zu löschen).

Fehlerbehebung (bei synchronisierendem System; wenn nur ein einzelner Server zu fixen ist können die "Synchronisierender Server"-Schritte weggelassen werden; alle Kommandos müssen im jeweiligen Projektverzeichnis stehend ausgeführt werden; ggf. heißt das Kommando statt `./mytism` auch `PROJEKTNAME`, also z.B. `./oashi`):

1. Synchronisierender Server: Server stoppen `./mytism stop_mytism`
2. Autoritativer Server: Server stoppen `./mytism stop_mytism`
3. Autoritativer Server: Backup ziehen `./mytism backup` (Nur zur Sicherheit)
4. Autoritativer Server: Fixer laufen lassen `./mytism run de.ipcon.db.tools.DoubleIdFixer PROJEKTINSTANZ --repairAll fix_double_ids.sql` (PROJEKTINSTANZ z.B. `./oashi`)
5. Autoritativer Server: `.checked-*`-Dateien löschen `rm .checked-*`
6. Autoritativer Server: Generiertes SQL-Script einspielen `psql -U postgres DATENBANKNAME < fix_double_ids.sql`
7. Synchronisierender Server: Backup ziehen (Nur zur Sicherheit)
8. Autoritativer Server: bi-tabelle leeren `psql -U postgres DATENBANKNAME`, dann dort `delete from bi`
9. Autoritativer Server: Server starten `./mytism start_mytism` (Diverse Aufräumarbeiten werden gemacht)
10. Autoritativer Server: Server stoppen `./mytism stop_mytism`
11. Autoritativer Server: Backup ziehen `./mytism backup`
12. Backup zum synchronisierenden Server kopieren
13. Autoritativer Server: Server starten `./mytism start_mytism`
14. Falls dort ein Grails läuft, dieses durchstarten `./mytism stop_grails`, dann `./mytism start_grails`
15. Synchronisierender Server: Backup einspielen `./mytism restore BACKUPDATEINAME`
16. Synchronisierender Server: `.checked-*`-Dateien löschen `rm .checked-*`
17. Synchronisierender Server: `.init-keygen` löschen

18. Synchronisierender Server: `.checked-firstnodestart` löschen

19. Synchronisierender Server: Server starten `./mytism start_mytism`

Beide Server sollten danach ohne Fehlermeldungen starten; auf der Status-Webseite kontrollieren, ob auf dem autoritativen Server der Sync-Account des synchronisierenden Servers angemeldet ist; im `logs/daily.log` des synchronisierenden Servers kontrollieren, ob die Synchronisation läuft (Nach Meldungen wie `"INFO 10:15:59.355 [shi->oashi.oashi.com] SyncService logSyncLocalToRemote(637)- >>>> BT[12345678] from 11-12-12 10:15:57 SrvCrea 11-12-12 10:15:57"` bzw. `"... log-SyncRemoteToLocal(...) ..."` Ausschau halten; ggf. selbst mal eine Änderung an einem Objekt auf einem der Server machen).