MyTISM - Ein Datenbank- und Anwendungs-Framework

OAshi-Mitarbeiter

Inhaltsverzeichnis

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

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

NOTE	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.

Erste Schritte mit MyTISM

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.

PostgreSQL

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

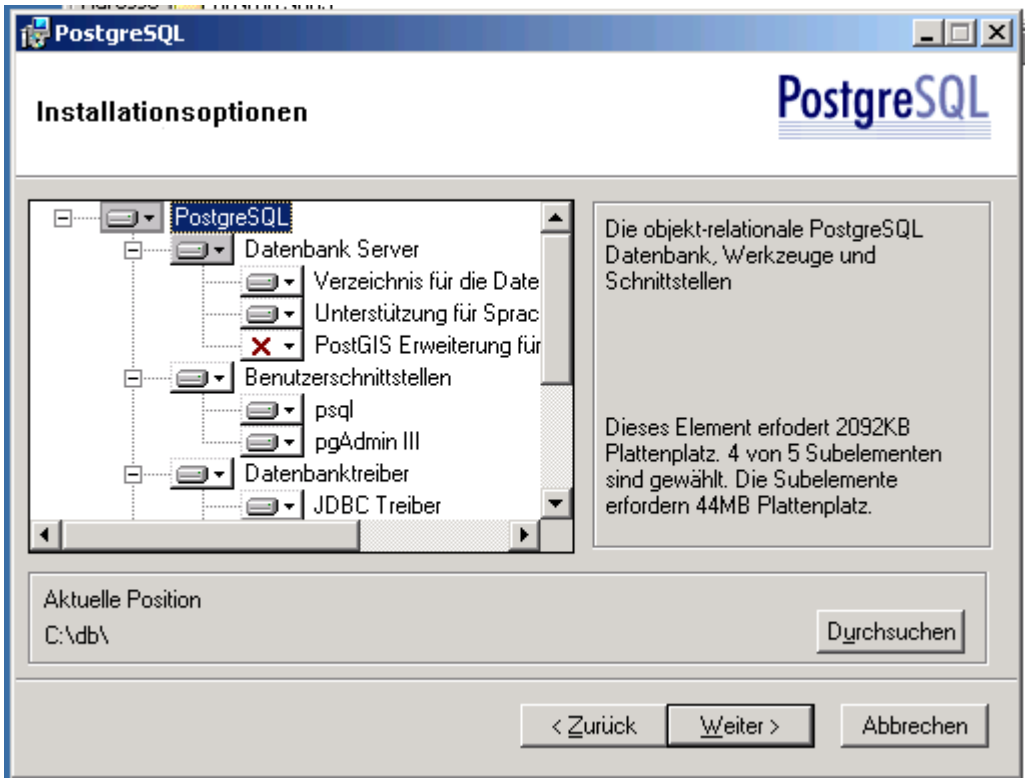
Am Anfang steht der Download des Installations-Archivs. Dieses kann von http://www.postgresql.org/ftp/win32 bezogen werden. Sollte der Link irgendwann nicht mehr stimmen, gehe man direkt auf http://www.postgresql.org und klicke sich bis zum Windows-Download durch. Hier wählt man das ZIP-Archiv `postgresql-8.0.3.zip` zum Download aus.

Das heruntergeladene Archiv muss entpackt werden. U.a. liegen dann zwei Dateien auf der Festplatte, die mit `postgres` beginnen und die Endung `.msi` haben - 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`



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 man - 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.

CAUTION	Wenn man auf der Kommandozeile per `createdb -U postgres -E UTF-8 DBNAME` eine neue Datenbank anlegt, muss man jetzt aber daran denken, das Encoding auf UTF-8 zu stellen!

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
```

Java Developer Kit (JDK)

Auch hier muss man sich erst einmal das Installations-Archiv von `http://java.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`).

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.

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.

Starten eines MyTISM-Servers

/mytism init_db

Sofern alles klappt ist unter Linux mit dem Aufruf von

/mytism start

```
aus dem Build-Verzeichnis heraus alles erledigt.
```

TIP	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`

Umziehen eines syncenden MyTISM-Servers auf eine andere Hardware

FIXME

CAUTION	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 gesynct wird, die später vom SyncChecker nachgezogen werden) fehlen.

Aufsetzen eines syncenden MyTISM-Servers aus einem Backup des authoritativen Servers

Neu-Einrichtung eines Knotens

1. Authoritativer Server:
   a. Neuen BN anlegen und speichern.
   b. .init-syncaccount-Datei aus dem BN-Formular des neuen BN exportieren.
   c. Diese .init-syncaccount-Datei auf den synchronisierenden Server kopieren.
   d. Backup ziehen "./mytism backup" oder das letzte komplette Nightly Backup verwenden.
   e. Das Backup zum synchronisierenden Server kopieren.
2. Synchronisierender Server:
   a. Backup einspielen.
   b. .checked-*-Dateien löschen "rm .checked-*" (WICHTIG! Auch die .checked-firstnodestart löschen!)
   c. .init-keygen löschen "rm .init-keygen", dies leert die lokale bi-Tabelle.
   d. Manuell muss mittels des Linux-Kommandos "rsync" fehlende filesRoot-Dateien des authoritativen Servers auf den Knoten kopiert werden.
   e. Server starten "./mytism start_mytism"
   f. Eine kleine Änderung speichern, damit der Sync das nächste Mal leichter den lokalen Ansatz finden kann.

TIP Wenn die Meldung `Server has no logs from us yet, so we can't check.` im Server-Log auftaucht, wurde Punkt 2.e nicht beachtet.

Erneutes Aufsetzen eines bestehenden Knotens

CAUTION Beim Neu-Aufsetzen sollte man darauf achten, dass die lokalen Daten alle zum Hauptserver synchronisiert wurden, bevor man das Backup vom Hauptserver zieht und damit den synchronisierenden Server neu aufsetzt.

1. Authoritativer Server:
   a. Backup ziehen "./mytism backup" oder das letzte komplette Nightly Backup verwenden, wenn die Daten des synchronisierenden Servers zu dieser Zeit bereits alle raus gesynct worden waren.
   b. Das Backup zum synchronisierenden Server kopieren.

2. Synchronisierender Server:
    a. Backup einspielen.
    b. .checked-*-Dateien löschen "rm .checked-*" (WICHTIG! Auch die .checked-firstnodestart löschen!)
    c. .init-keygen löschen "rm .init-keygen", dies leert die lokale bi-Tabelle.
    d. Manuell muss mittels des Linux-Kommandos "rsync" fehlende filesRoot-Dateien des authoritativen Servers auf den Knoten kopiert werden.
    e. Server starten "./mytism start_mytism"
    f. Eine kleine Änderung speichern, damit der Sync das nächste Mal leichter den lokalen Ansatz finden kann.

TIP Wenn die Meldung `Server has no logs from us yet, so we can't check.` im Server-Log auftaucht, wurde Punkt 2.e nicht beachtet.

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.

NOTE Unter Windows-Systemen wird die heruntergeladene Anwendung (Datei mit der Endung ".jnlp") im Profil gespeichtert, 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.

Konfiguration des Servers via mytism.ini

Die Datei `mytism.ini` befindet sich im Projektverzeichnis ("Punkt-Verzeichnis"). Die Einstellungen werden über den Abschnitt `[DBMan]` vorgenommen. Es können weitere Unterabschnitte definiert

werden.

Es folgt eine Tabelle mit Erklärungen zu den möglichen Einstellungsvariablen. In der ersten Spalte steht der Variablenname, dem ein Wert zugewiesen werden kann. In der zweiten Spalte steht der Default bzw. falls in Klammern ein Beispiel.

schemaFile	(/.demo//schema/schema.xml)	Absoluter Pfad zur schema.xml
driver	org.postgresql.Driver	fully qualified Klassenname des Datenbanktreibers
url	(jdbc:postgresql://localhost:5432/demo)	URL für die Verbindung zur Datenbank
user	postgres	Benutzername für die Verbindung zur Datenbank
pass	postgres	Passwort für die Verbindung zur Datenbank
filesRoot	(/.demo/filesRoot)	Speicherort im Dateisystem für die BLOB
fileVault	de.ipcon.db.blob.ServerFileVault	fully qualified Klassenname des File-Vault, der die BLOBs managed
fileVaultCfg	FileVault	Name für den Abschnitt mit weiterer Konfiguration für den File-Vault
persistenceMgr	de.ipcon.db.CastorPersistenceManager	fully qualified Klassenname des Persistenz-Managers
persistenceMgrCfg	Castor	Name für den Abschnitt mit weiterer Konfiguration für den Persistenz-Manager
keyGenerator	de.ipcon.db.JDBCKeyGenerator	fully qualified Klassenname des Key-Generators
keyGeneratorCfg	KeyGen	Name für den Abschnitt mit weiterer Konfiguration für den Key-Generator
protocolDriver	de.ipcon.db.SocketProtocolServer	fully qualified Klassenname des Protokoll-Treibers
protocolDriverCfg	SocketProtocol	Name für den Abschnitt mit weiterer Konfiguration für den Protokoll-Treiber
cryptoMgr	de.ipcon.db.CryptoManager	fully qualified Klassenname des Crypro-Managers

cryptoMgrCfg	Crypto	Name für den Abschnitt mit weiterer Konfiguration für den Crypro-Manager
noMetaDataCheck	0	Unterdrückung des Meta-Daten-Checks beim Serverstart durch Angabe von 1
noInitialDataCheck	0	Unterdrückung des Initial-Daten-Checks beim Serverstart durch Angabe von 1
noIntegrityCheck	0	Unterdrückung des Integritäts-Checks beim Serverstart durch Angabe von 1
noIntegrityDoubleIdsCheck	0	Unterdrückung des Checks auf doppelte Ids beim Serverstart durch Angabe von 1 (ist automatisch inaktiv, wenn der Integritäts-Check deaktiviert wurde)
tomcatPool	TomcatPool	Aktiviert den TomcatPool für JDBC-Verbindungen. Außerdem der Name für den Abschnitt mit weiterer Konfiguration für den Tomcat-Pool
hikariPool	HikariPool	Aktiviert den HikariCP-Pool für JDBC-Verbindungen. Außerdem der Name für den Abschnitt mit weiterer Konfiguration für den HikariCP-Pool
authoritative	1	Angabe, ob dieser Server authoritativ ist (1=ja, 0=nein, also synchronisierende Instanz)
syncService	de.ipcon.db.SyncService	fully qualified Klassenname des Sync-Service
syncServiceCfg	SyncService	Name für den Abschnitt mit weiterer Konfiguration für den Sync-Service

fileVault

FIXME

Konfiguration des ServerFileVault

FIXME

persistenceMgr

FIXME

Konfiguration des CastorPersistenceManager

FIXME

keyGenerator

FIXME

Konfiguration des JDBCKeyGenerator

FIXME

protocolDriver

Der Protokoll-Treiber wird über den Eintrag `protocolDriver` eingestellt, indem dort die Klasse angegeben wird (fully qualified). Die Konfiguration des Protokoll-Treibers geschieht im Abschnitt, der über den Eintrag `protocolDriverCfg` gesetzt wird (default: `SocketProtocol`)

Konfiguration des Socket-Protokoll-Treibers

Für den Socket-Protkoll-Treiber können folgende Werte eingestellt werden, in Spalte 2 der jeweilige Default.

host	-	Der Hostname, unter dem auf unverschlüsselte Verbindungen gehört wird.
port	4242	Der Port, auf dem der Server auf unverschlüsselte Verbindungen hört.
backlog	10	Die maximale Anzahl von schwebenden Verbindungen für die ServerSocket für unverschlüsselte Verbindungen.
tlsHost	-	Der Hostname, unter dem auf (verschlüsselte) TLS-Verbindungen gehört wird.
tlsPort	4243	Der Port, auf dem der Server auf (verschlüsselte) TLS-Verbindungen hört.
tlsBacklog	10	Die maximale Anzahl von schwebenden Verbindungen für die ServerSocket für (verschlüsselte) TLS-Verbindungen.
maxWaitForAuth	210000	Anzahl der Millisekunden, die gewartet werden soll, bis sich der BackendCommandHandlerI authentifiziert hat; nach Ablauf dieser Zeit wird die Verbindung hart getrennt (Bitte beachten, dass der Login-Dialog die Verbindung nach 3 Minuten Untätigkeit unterbricht, der Wert sollte also unter 3 Minuten bleiben, ansonsten beendet der Login-Dialog die Verbindung von sich aus und es kommt zu Fehlermeldungen im Server-Log.)
hardMaxUnauthedBCHPerIP	150	Wenn mehr als diese Anzahl von unauthentifizierten BackendCommandHandlerI von der gleichen IP verbunden sind, werden alle weiteren Verbindungsversuche sofort zurückgewiesen.

softMaxUnauthedBCHPerIP	50	Wenn mehr als diese Anzahl von unauthentifizierten BackendCommandHandlerI von der gleichen IP verbunden sind, werden alle weiteren Verbindungsversuche erst verzögert beantwortet.
delayFactorUnauthedBCH	200	Faktor in Millisekunden, um den jede Verbindung über softMaxUnauthedBCHPerIP von der gleichen IP verzögert wird, d.h. 200ms für die 51. Verbindung, 400ms für die 52. usw.

cryptoMgr

FIXME

Konfiguration des CryptoManager

FIXME

Konfiguration des tomcatPool

maxTotal	128	Die maximale Anzahl aktiver Verbindungen, die gleichzeitig aus diesem Pool zugewiesen werden können.
maxIdle	128	Die maximale Anzahl von Verbindungen, die immer im Pool gehalten werden sollen. Leerlaufverbindungen werden regelmäßig überprüft (falls aktiviert) und Verbindungen, die länger als 60s im Leerlauf sind, werden freigegeben.
minIdle	0	Die Mindestanzahl der hergestellten Verbindungen, die zu jeder Zeit im Pool gehalten werden sollen. Der Verbindungspool kann unterhalb dieser Nummer schrumpfen, wenn Validierungsabfragen fehlschlagen.
initialSize	0	Die anfängliche Anzahl von Verbindungen, die beim Start des Pools erstellt werden.
maxWait	10000	Die maximale Anzahl von Millisekunden, die der Pool abwarten wird (wenn keine verfügbaren Verbindungen vorhanden sind), bis eine Verbindung wieder freigegeben wird, bevor eine Exception geworfen wird.
maxAge	300000	Zeit in Millisekunden, für die diese Verbindung gehalten wird. Wenn eine Verbindung an den Pool zurückgegeben wird, überprüft der Pool, ob das now - time-when-connected > maxAge Limit erreicht ist, und wenn ja, schließt er die Verbindung, anstatt sie in den Pool zurückzugeben.

Konfiguration des hikariPool

connectionTimeout	10000	Diese Eigenschaft steuert die maximale Anzahl von Millisekunden, die ein Client auf eine Verbindung vom Pool warten wird. Wenn diese Zeit überschritten wird, ohne dass eine Verbindung verfügbar wird, wird eine SQLException ausgelöst. Niedrigste akzeptable Verbindungszeitüberschreitung ist 250 ms.
idleTimeout	120000	Diese Eigenschaft steuert die maximale Zeitdauer, die eine Verbindung im Pool leerlaufen darf. Diese Einstellung gilt nur, wenn minimumIdle kleiner als maximumPoolSize ist. Ob eine Verbindung im Leerlauf verworfen wird oder nicht, unterliegt einer maximalen Variation von +30 Sekunden und einer durchschnittlichen Variation von +15 Sekunden. Eine Verbindung im Leerlauf wird niemals vor diesem Timeout verworfen. Ein Wert von 0 bedeutet, dass Leerlaufverbindungen nie aus dem Pool entfernt werden. Der minimal zulässige Wert beträgt 10000ms (10 Sekunden).
maxLifetime	300000	Diese Eigenschaft steuert die maximale Lebensdauer einer Verbindung im Pool. Eine Verbindung in Nutzung wird niemals verworfen, sie wird nur dann entfernt, wenn sie geschlossen wurde. Wir empfehlen dringend, diesen Wert einzustellen, und es sollte mindestens 30 Sekunden weniger sein als die Verbindungszeitbegrenzung der Datenbank oder der Infrastruktur. Ein Wert von 0 bedeutet keine maximale Lebensdauer (d.h. unendliche Lebensdauer), die natürlich der idleTimeout-Einstellung unterliegt.
minimumIdle	0	Diese Eigenschaft steuert die minimale Anzahl von Leerlaufverbindungen, die HikariCP versucht, im Pool zu verwalten. Wenn die Anzahl der Leerlaufverbindungen unter diesen Wert sinkt, wird HikariCP versuchen, zusätzliche Verbindungen schnell und effizient hinzuzufügen. Für maximale Leistung und Reaktionsfähigkeit auf Spitzenlasten empfehlen wir jedoch, diesen Wert nicht zu setzen und stattdessen HikariCP als einen fixed size Verbindungspool einzusetzen.
maximumPoolSize	128	Diese Eigenschaft steuert die maximale Größe, die der Pool erreichen darf, einschließlich der Leerlaufverbindungen und der Verbindungen in Nutzung. Grundsätzlich bestimmt dieser Wert die maximale Anzahl der tatsächlichen Verbindungen zum Datenbank-Backend. Ein vernünftiger Wert dafür wird am besten durch Ihre Ausführungsumgebung bestimmt. Wenn der Pool diese Größe erreicht hat und keine Leerlaufverbindungen zur Verfügung stehen, werden die calls an getConnection() bis zu connectionTimeout Millisekunden geblockt.

syncService

FIXME

Konfiguration des SyncService

FIXME

Die ".checked*"-Dateien

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

.PROJEKT/.checked-firstnodestart	Wurde die Datenbank auf den ersten Knotenstart vorbereitet? (findet sich nur auf synchronisierenden Servern; wenn nicht vorhanden, werden die Logs soweit weg geräumt, dass der Sync den initialen Ansatz findet, d.h. dass nur noch eine lokale BT dort liegt und keine BPs mehr)
.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	Enthält die Ids der zuletzt vom Hauptserver bzw. zum Hauptserver übertragenen BTs (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.

Die ".init*"-Dateien

FIXME (Erklärung)

.PROJEKT/.init-keygen	Fehlt diese Datei, so wird die bi-Tabelle in der Datenbank geleert (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?

Services

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

BusinessNode (BN)

Wozu dienen BNs?

FIXME

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.

BusinessUnit (BU)

Wozu dienen BUs?

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

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 eine BU 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 (Incremement) 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".

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.

BusinessService (BS)

Wozu dienen BSe?

Um z.B. Scripte ständig oder wiederkehrend auszuführen. Ein möglicher Anwendungsfall wäre die Synchronisierung von Daten aus Fremdsystemen mit MyTISM.

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:

Reiter 'Dienst'

Zur Angabe von Metadaten für den Dienst.

Verantwortlicher	Der für das Skript verantwortliche Benutzer
Name	Erinnerung an 'etwas' (beliebiger Name)
Beschreibung	BS für Erinnerungen (beliebige Beschreibung)
Java-Klasse	`de.ipcon.db.sync.ScriptService` (Klasse, die das Skript ausführt. Dies ist für die meisten Skripts die im Beispiel angegebene Klasse.)
Aktiv	Checkbox, um den Aktivitätsstatus des Skripts zu setzen
Ausführungs-Vorschrift	<ExecutionPolicy> <CronJob interrupt="false"> <Commands> <Command>10 15 * * *</Command> </Commands> </CronJob> </ExecutionPolicy> oder <ExecutionPolicy> <ExecutionPolicyKeepRunning/> </ExecutionPolicy>`
BNs	hier die Business Nodes auswählen, für die der BS gelten soll. **Wichtig! ohne Node wird der Service nirgends ausgeführt.**

Reiter 'Script'

Hier wird das Skript eingegeben. Es handelt sich hierbei um XML-Skripte. Das `<Sync>`-Tag signalisiert, dass es sich um einen Dienst handelt. Darauf folgen Metadaten, die angeben, wie der Dienst sich mit dem System verbinden soll. Es folgt die angabe der Skriptsprache und dann das Skript selbst, das in einem `<![CDATA]>`-Block enthalten ist.

```
<Sync>
  <mytism-connection url="socket://localhost" user="Benutzer eintragen"
pass="einPasswort"/>
  <script language="groovy"><![CDATA[
    tx = api.getNewTx()
    tx.description = "Versende Benachrichtigung für someone@domain.com"
    def mail = MyTISMAdresseEmail.getOrCreate( tx, "someone@domain.com" )
    def ben = tx.includeNew( MyTISMBenAuftragOhneVorlage )
    ben.betreffQuelle = "Fehler in [xxx]"
    ben.textQuelle = "Bitte korrigieren Sie den Fehler [yyy]"
    ben.betreffIstFestQuelle = true
    ben.textIstFestQuelle = true
    ben.addEmpfaenger( mail )
    api.saveBO( tx )
  ]]></script>
</Sync>
```

Reiter 'Stacktrace' Hier wird der letzte Fehler (falls aufgetreten) im letzten Lauf des Skripts angezeigt.

Wie benutzt man sie?

Business Services (BS) können sowohl als Dienst im Hintergrund laufen, als auch manuell über die Kommandozeile ausgeführt werden.

Ausführung über die Kommandozeile

```
./mytism run-bs /Pfad/zum/Script
```

ExecutionPolicy - Cron

Parameter

- *interrupt* (optional) Wenn 'interrupt' auf true steht, wird ein bereits laufender Prozess des gleichen Jobs bei der nächsten Ausführung gestoppt (default: *false*).

Jedes cron-Kommando hat 5 Felder, welche jede Minute gegen die aktuelle Zeit ausgewertet werden. Die Syntax orientiert sich an der unix crontab.

Feld	erlaubte Werte
Minute	0-59, *
Stunde	0-23, *
Tag im Monat	1-31, *
Monat	1-12, *
Tag der Woche	0-6, * (0 ist Sonntag)

Stern (*) bedeutet: 'erster bis letzter'.

Bereiche sind erlaubt. Ein Bereich sind zwei Zahlen, getrennt durch ein Minus (-).

> 8-11 im 'Stunden'-Feld bedeutet: zur Stunde 8, 9, 10 und 11.

Listen sind erlaubt. Eine Liste sind zwei oder mehr Zahlen, getrennt durch ein Komma (,).

> 8,10,11 im 'Stunden'-Feld bedeutet: zur Stunde 8, 10 und 11.

Schritte sind erlaubt. Ein Schritt wird spezifiziert als Stern/Liste/Zeitraum Schrägstrich (/) Schrittweite.

> 0-23/2 im Stunden-Feld bedeuted 'jede zweite Stunde' (ist also äquivalent zu 0,2,4,6,8,10,12,14,16,18,20,22) und */2.

*/5 im Minten-Feld bedeuted 'alle 5 Minuten'

Der Ausführungstag hat zwei mögliche Felder - 'Tag im Monat' und 'Tag der Woche'. Wenn beide Felder verwendet werden, wird das Script evt. mehrfach ausgeführt.

30 4 1,15 * 5 würde den Dienst um 4:30 des 1. und 15. jedes Monats starten, und jeden Freitag.

Ereignisse, welche in Daylight-Saving Korrekturen fallen, werden stillschweigend nicht oder mehrfach ausgeführt.

Besondere Kommandos sind	
@yearly	Alias für "0 0 1 1 *"
@annually	Alias für "0 0 1 1 *"
@monthly	Alias für "0 0 1 * *"
@weekly	Alias für "0 0 * * 0"
@daily	Alias für "0 0 * * *"
@hourly	Alias für "0 * * * *"

Besondere Kommandos für Feld #5 (Tag der Woche)	
@lastOfM	Alias für 'letzter ${Tag der Woche} im aktuellen Monat'
@lastOfY	Alias für 'letzter ${Tag der Woche} im aktuellen Jahr'

"1-20/2 16-17 * * */@lastOfY" bedeutet: 'von 16:00 bis 16:20 alle 2 Minuten und von 17:00 bis 17:20 alle 2 Minuten, an jedem Tag der letzten 7 Tage des Jahres, zum Beispiel am 25.12.2012 17:16:00, aber nicht 25.12.2012 17:15:00

"@weekly" bedeutet: 'jeden Sonntag um 0:00 Uhr'

"30 6 * 5-7 4/@lastOfM" bedeutet: 'um 6:30 Uhr an jedem letzten Mittwoch in den Monaten Mai, Juni, Juli'

SkriptServices

FIXME Einleitende Erklärung

Navigationsbaum

Der Navigationsbaum stellt eine wichtige Komponente der Solstice-Benutzeroberfläche dar.

Sichtbarkeit von Elementen

Angezeigt werden im Navigationsbaum generell Strukturelemente (`Ordner`, `Lesezeichen`, `Schablonen`, `Formulare`, `Reports` und `Aliase` darauf); ein virtueller Ordner für den angemeldeten Benutzer; für Admins ein virtueller Ordner mit allen Benutzern; und nach dem Suchen von Strukturelementen ein virtueller Ordner mit Unterordnern für die Suchergebnisse.

Welche Strukturelemente im Navigationsbaum für einen angemeldeten Benutzer sichtbar sind wird von mehreren Faktoren gesteuert. Folgende Dinge werden in der angegebenen Reihenfolge geprüft:

Element gelöscht?

Gelöschte Elemente werden nie im Baum angezeigt.

Sichtbarkeitsskript

- Falls für das Element ein *Sichtbarkeitsskript* (Sprache *Groovy*) hinterlegt ist, wird dieses ausgeführt.
- Falls es "wahr" (`true`) für den angemeldeten Benutzer liefert, wird mit der nächsten Überprüfung fortgefahren; ansonsten ist das Element nicht sichtbar.

Element für Benutzer bzw. mindestens eine seiner Gruppen als "sichtbar" konfiguriert?

- Wenn *SichtbarFuerGruppen* für das Element mindestens einen Eintrag hat, wird geprüft, ob der Benutzer mindestens einer der Gruppen aus *SichtbarFuerGruppen* angehört; falls ja wird mit der nächsten Überprüfung fortgefahren; ansonsten ist das Element nicht sichtbar.
- Wenn *SichtbarFuerGruppen* keine Einträge hat oder der angemeldete Benutzer ein Administrator ist wird hier nichts geprüft.

Ausreichende Rechte vorhanden?

- Es wird geprüft, ob der Benutzer ausreichende Rechte hat, um das Element sinnvoll nutzen zu können.
- Für Schablonen oder Aliase darauf muss er Objekte der zugehörigen Klasse erstellen und schreiben dürfen; für Lesezeichen und eigenständige Reports oder Aliase darauf muss er Objekte der zugehörigen Klasse oder einer ihrer Unterklassen lesen dürfen.
- Sollte bei der Überprüfung der Rechte ein Fehler auftreten, wird das Element ebenfalls angezeigt - dieser Fall ist insb. für "kaputte" Aliase ohne Verweis auf ein Original gedacht, damit diese im Baum sichtbar sind und repariert werden können.

Element an der Wurzel des Baumes?

Elemente ohne Elter, d.h. an der Wurzel des Baumes werden nun angezeigt.

Element Kind eines bereits angezeigten Elements?

Elemente die Kinder eines bereits angezeigten Elements sind, werden ebenfalls angezeigt.

Im Normalfall reicht es, die Sichtbarkeit von Ordnern zu beschränken, da deren Kindelemente - selbst

solche, die sonst im Prinzip für den angemeldeten Benutzer sichtbar wären - natürlich ebenfalls nicht angezeigt werden, wenn der Ordner an sich bereits nicht angezeigt wird. Es ist aber auch möglich, mittels der oben beschriebenen Konfigurationsmöglichkeiten einzelne Kindelemente eines Ordners auszublenden.

Durch Eintragen von `log4j.logger.de.ipcon.form.navtree.BenanntNavigationTreeNode=TRACE` in der `/.projekt/client-log.conf` können hierzu Debug-Informationen im Client-Log ausgegeben werden.

IMPORTANT	Die Eintragungen auf den Reitern "Gruppen" für Strukturelemente spielen für die Anzeige im Baum *keine* Rolle! Diese werden nur benötigt bei Anzeige im Kontextmenü und ähnlichen Stellen.

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.

Der Ordner "Alle GUI-Benutzer" und seine Konfiguration

Am unteren Ende der Navigationsbaums befindet sich ein Ordner namens "Alle GUI-Benutzer", in dem alle Benutzer angezeigt werden, die sich über den Client in das System einloggen können.

[Der Benutzer-Ordner] | *images/grouped_view_of_users/all-users_directory.png*

Wenn es nicht sehr viele Benutzer im System gibt, könnte der Inhalt des Ordners ungefähr so aussehen:

[Inhalt bei wenig Benutzern] | *images/grouped_view_of_users/grouping_disabled.png*

Überschreitet die Anzahl der User jedoch ein bestimmtes Limit, werden sie anhand ihres Namens kategorisiert. Die genauen Kriterien der Kategorisierung lassen sich über den entsprechenden Satz `EinstellungsVariablen` beeinflussen. Generell gilt, dass die Logik dahinter versucht, ähnliche Namen in einen gemeinsamen Unterordner zu packen. (Beispielsweise alle Namen, die mit "A" beginnen, oder alle Namen, die aus Zahlen bestehen)

Das obige Beispiel würde sich dann so darstellen:

[Inhalt bei wenig Benutzern] | *images/grouped_view_of_users/grouped_users.png*

Konfigurationsparameter

Benutzer können durchnummeriert sein, Kürzel oder volle Namen verwenden. Abhängig davon, welche Namenskonvention gängig ist, macht es durchaus Sinn, die Ansicht auf den entsprechenden Fall anzupassen.

"users.view.groupingStart"

- Dieser Wert beschreibt die Mindestanzahl an Benutzer, die überstiegen werden muss, damit eine Aufteilung in Unterordner ("Gruppierung") vorgenommen wird.
- **Standardwert:** 30

"users.view.enableGrouping"

- Hierüber kann die Gruppierung in der Ansicht der Benutzer ausgeschaltet werden, selbst wenn die Anzahl der Benutzer das Limit übersteigt. So werden immer alle Benutzer untereinander angezeigt
- **Standardwert:** enabled

"users.view.group.maxElements"

- Dieser Wert beschreibt das absolute Maximum an Benutzern, die ein Unterordner aufnehmen darf, ehe ein weiterer Unterordner der Liste hinzugefügt wird. Gibt es also 15 Benutzer, deren

Name mit "A" beginnt und "maxElements" ist auf "10" gesetzt, werden diese auf mindestens zwei Unterordner aufgeteilt.

- **Standardwert:** 15

"users.view.group.minElements"

- Dieser Wert beschreibt das Minimum an Benutzer, die ein Unterorder aufnimmt, ehe er die Aufnahme weiterer Elemente aufgrund von anderen Kritierien verweigern darf. Dies bedeutet, dass die Anzahl von Benutzern pro Unterordner zwischen zwischen "minElements" und "maxElements" liegt.
- **Standardwert:** 5

"users.view.group.numericRange"

- Der numerische Bereich definiert, wie Benutzer unterteilt werden sollen, deren Namen ausschließlich aus Zahlen bestehen. Ein Wert von "10" sorgt dafür, dass Benutzer in 10er-Schritten unterteilt werden, ein Wert von "100" in 100er-Schritten, usw.
- **Standardwert:** 10

"users.view.group.maxSubgroups"

- Wenn alle Benutzer "A" und alle Benutzer "B" in einen Unterordner gesteckt werden, da das "minElement" -Limit nicht erfüllt wurde, so bilden "A" und "B" innerhalb des Unterordners sog. "Untergruppen". Auch numerische und nicht-numerische (oder gemischte) Benutzernamen bilden eigene Untergruppen. Ist dieses Limit vor "maxElements" erreicht, müssen nachfolgende Benutzer in dem nächsten Unterordner gruppiert werden.
- **Standardwert:** 4

Beispiel-Konfigurationen

"0-9,A-Z"

Gruppiere numerische und nicht-numerische Benutzernamen.

- **users.view.group.numericRange:** 100000 (sollte grob die Anzahl der numerischen Benutzernamen wiederspiegeln)
- **users.view.group.maxElements:** 9999 (oder ein ähnlich hoher Wert)
- **users.view.group.minElements:** 0
- **users.view.group.maxSubgroups:** 999 (muss kleiner sein als **maxElements**)
- [0-9]

"0,1,2... A,B,C,...Z"

Gruppiere Benutzernamen nach Anfangsbuchstaben.

- **users.view.group.numericRange:** (Da Zahlen als Zahlen interpretiert werden, sollte der Wert > 1 liegen)

- **users.view.group.maxElements:** 9999 (oder ein ähnlich hoher Wert)
- **users.view.group.minElements:** 0
- **users.view.group.maxSubgroups:** 1
- [0-9] ← Exmaple with **numericRange** = 5

Feste Größe der Unterordner

- Dazu einfach **users.view.group.minElements** = **users.view.group.maxElements** setzen

Plugins - Zuschaltbare Komponenten im Client

Eingabefeld für Befehle in der Aktionsleiste einblenden

Dieses Eingabefeld ist im Grunde ein Plugin der Solstice GUI um Befehle an den Client abzusetzen. Die nachfolgenden Liste zeigt alle Befehlsmöglichkeiten, welche in diesem Eingabefeld durchgeführt werden können.

- [0-9]+ öffnet das Objekt mit der entsprechenden Id. Zum Öffnen wird das Standardformular benutzt.
- :[0-9] öffnet das Objekt mit der entsprechenden Id. Zum Öffnen wird das Formular mit der hinter dem *:* stehenden Id genutzt (Falls die Berechtigungen dies erlauben).
- [0-9]+:auto öffnet das Objekt mit der entsprechendn Id. Zum Öffnen wird das automatisch generierte Formular genutzt.
- #[0-9]+ öffnet das Lesezeichen mit der entsprechenden Id
- #[0-9]+:.* öffnet das Lesezeichen mit der entsprechenden Id und führt direkt die angegebene Query aus.

Die Befehlseingabe befindet sich am rechten Rand der Aktionsleiste (Leiste unter der Menüleiste). Um die Befehlseingabe zu aktivieren, muss die folgende Skriptzeile im Voreinstellungsskript des Benutzers im Tag Configuration → Profile angegeben werden. Nach dem Hinzufügen dieser Zeile ist ein Neuanmelden nötig.

```
<Plugin class="de.ipcon.form.ClientToolBarManager" name="Main" orientation="HORIZONTAL"
position="NORTH" rollover="true" floatable="true" cli="true"/>
```

Server-Gesundheitsanzeige einblenden

Zur Aktivierung des Plugins fügt man folgende Zeile im Voreinstellungsskript des Benutzers im Tag

Configuration → Profile ein. Nach dem Hinzufügen dieser Zeile ist ein Neuanmelden nötig.

```
<Plugin class="de.ipcon.form.ServerHealthMonitor"/>
```

Das Plugin wird unten rechts in der Statusleiste eingeblendet. Das Plugin zeigt auf an synchronisierenden Servern angemeldeten Clients als Beschriftung den aktuellen Synchronisations-Verzug in Minuten an. Auf authoritativen Servern wird als Beschriftung ein statischer Text angezeigt. Klickt man den Knopf des Plugins an, so öffnet sich ein Dialogfeld mit weiteren Kenndaten des Servers, an dem man angemeldet ist. Die Informationen über den Server werden standardmäßig alle 60 Sekunden an die Clients, die das Plugin aktiviert haben, per DBManServerHealthEvent verteilt.

Rechteverwaltung

NOTE Alle genannten Formulare, Schablonen und Lesezeichen liegen normalerweise im Ordner *"Benutzerverwaltung"*.

CAUTION Änderungen an Gruppen- und Rechtezuweisungen wirken sich für angemeldete Benutzer erst aus, nachdem sie sich einmal ab- und wieder angemeldet haben!

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?

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?*).

Erste Vorraussetzung 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.

Benutzer

Ein *Benutzer* steht für eine einzelne reale oder virtuelle (z.B. externe Programme) Person, die mit einem MyTISM-System arbeiten kann, bzw. genauer eigentlich für das dieser Person zugordnete Benutzerkonto. 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.

Für dieses Tutorial nehmen wir an, dass folgende Benutzer bereits im System definiert wurden: Admin, Alice, Bob und Claire

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.

NOTE Um Rechte nur an einen einzelnen Benutzer zu vergeben kann man natürlich eine eigene Gruppe bauen und dann diesen Benutzer als deren einziges Mitglied definieren.

Eine Gruppe *"Admins"* wird automatisch gebaut und enthält am Anfang nur den Benutzer *"Admin"*. Ebenso gibt es eine Gruppe *"Benutzer"*, die alle Benutzer enthalten sollte.

CAUTION Es ist vorgesehen, jedoch *nicht* verpflichtend, dass jeder Benutzer Mitglied der *"Benutzer"*-Gruppe ist.

In diesem Tutorial benutzen wir folgende Gruppen: Admins, Benutzer (enthält Alice, Bob und Claire) und 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.

BO-Masken

Um Rechte für bestimmte Objekte festzulegen, muss man die entsprechenden Objekte natürlich irgendwie auswählen. Dies geschieht mit Hilfe der sog. *BO-Masken*; damit definiert man eine Menge von BOs, für welche man dann einer oder mehreren Gruppen bestimmte Rechte erlauben oder verweigern will. Diese Rechte gelten dann uneingeschränkt auch für die Kinder des Objektes (also auch für die Attribute des erbenden BOs, die in der Eltern-Entität nicht definiert sind).

BOMasken haben folgende Eigenschaften:

Name

Pflichtfeld - Eine passender, aber im Prinzip frei wählbarer Name für die BO-Maske. ***FIXME es gibt eine Namenskonvention?***

Beschreibung

Optional - Ein kurzer Text, der in ein paar Worten "menschenlesbar" erklärt, welche BOs mit dieser Maske ausgewählt werden.

...vom Typ

Pflichtfeld - Hier bestimmt man, BOs welchen Typs diese BO-Maske auswählt.

...für die dieses Skript "wahr" liefert

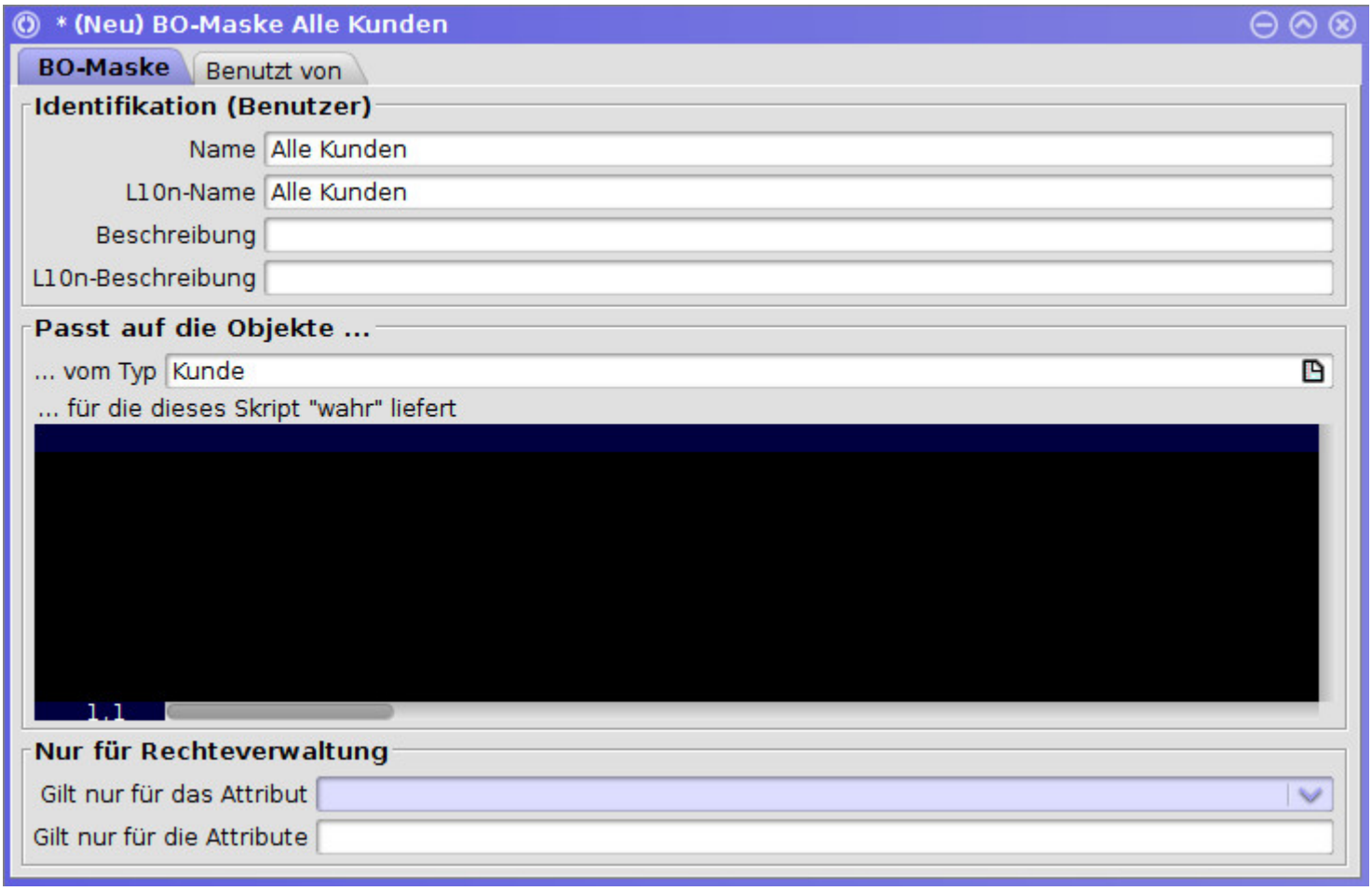
Optional - Erlaubt die Auswahl weiter einzuschränken; wird im nächsten Abschnitt genauer erklärt.

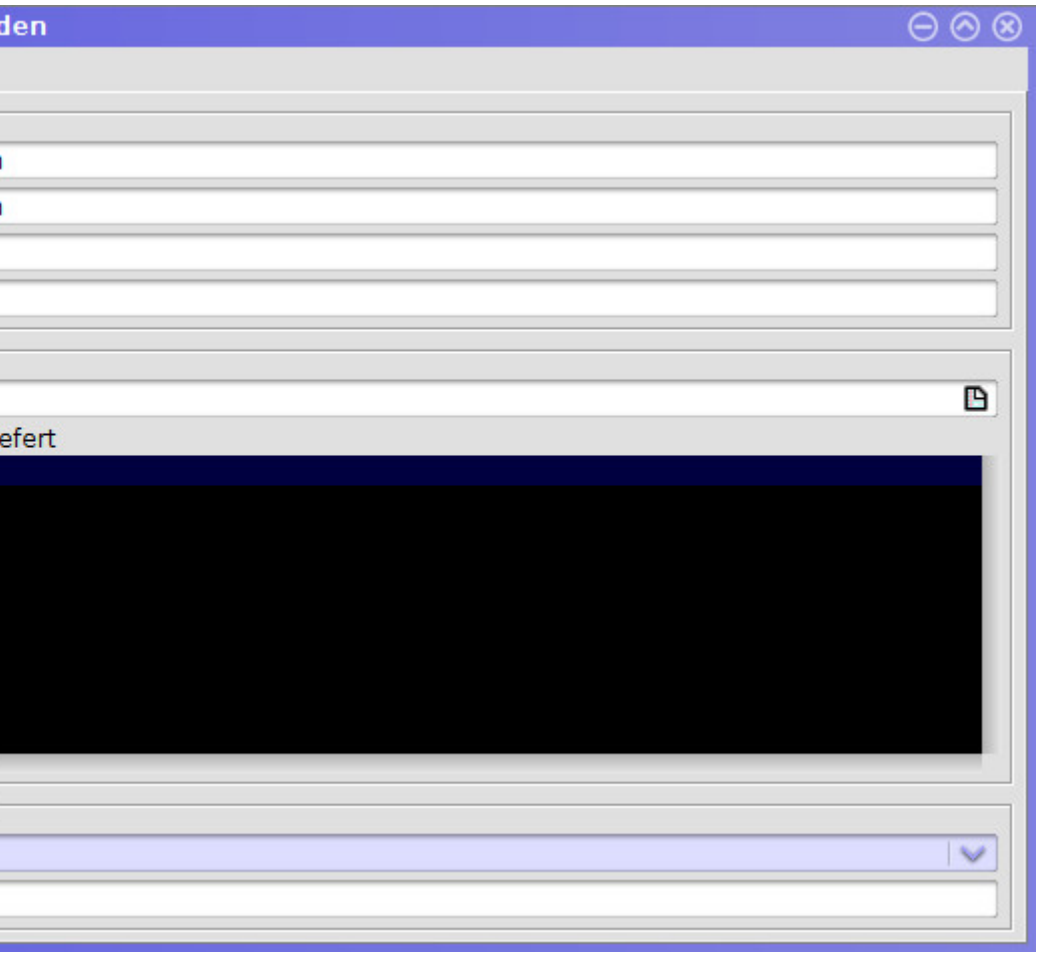
Gilt nur für das Attribut

Optional - Wenn Sie 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. Dieses Feld ist eine vereinfachte Eingabemöglichkeit, wenn Sie hier nur ein einzelnes Attribut angeben möchten. Wenn Sie unter *"Gilt nur für die Attribute"* mehr als ein Attribut aufführen, wird dieses Feld automatisch ausgeblendet.

Gilt nur für die Attribute

Optional - Wenn Sie hier einen Wert angeben, so bezieht sich die Maske nur auf diese Attribute der BOs, d.h. sie können hiermit Rechte für einzelne Eigenschaften von BOs vergeben. Geben Sie ein oder mehrere Attribute mit ihrem Namen an, getrennt mit Komma.


* (Neu) BO-Maske Alle Kunden
BO-Maske
Benutzt von
Identifikation (Benutzer)
Name
Alle Kunden
L10n-Name
Alle Kunden
Beschreibung
L10n-Beschreibung
Passt auf die Objekte ...
... vom Typ
Kunde
... für die dieses Skript "wahr" liefert
1,1
Nur für Rechteverwaltung
Gilt nur für das Attribut
Gilt nur für die Attribute



Filterskript

Mittels des optionalen Filterskripts (Sprache *Groovy*) unter *"...für die dieses Skript "wahr" liefert"* kann die Menge der Objekte, welche von der Maske ausgewählt werden, weiter eingeschränkt werden.

Es wird für jedes BO, das von der Maske "überprüft" wird ausgewertet. Nur wenn das Skript wahr (`true`) zurückliefert, wird das entsprechende BO berücksichtigt.

Im Skript stehen folgende vordefinierte Variablen zur Verfügung:

bo

Das zu überprüfende BO.

schema

Das verwendete Schema.

entity

Der in der BO-Maske angegebene Typ (*Entität*).

att

Der *Name* des Attributs, für das Rechte überprüft werden sollen. Ggf. `null` wenn kein spezielles Attribut sondern das gesamte BO überprüft werden soll. Auch `null` bei Verwendung der Maske

außerhalb des Rechtesystems.

maske

Ein Verweis auf die BO-Maske selbst.

log

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

user

Der aktuelle Benutzer.

NOTE `return` kann in Groovy weggelassen werden, der Rückgabewert eines Skripts ist automatisch das Ergebnis des letzen Ausdrucks.

Example 1. Beispiele für Filterskripte:

(Eher unsinnig :-) Maske wählt nur BOs mit einer Id > 1000 aus:

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

Maske wählt nur BOs aus, die das Flag IstSchoen (ein Boolean-Attribut) gesetzt haben:

```
bo.IstSchoenNN
```

Maske wählt nur Gruppen, in denen der aktuelle Benutzer Mitglied ist:

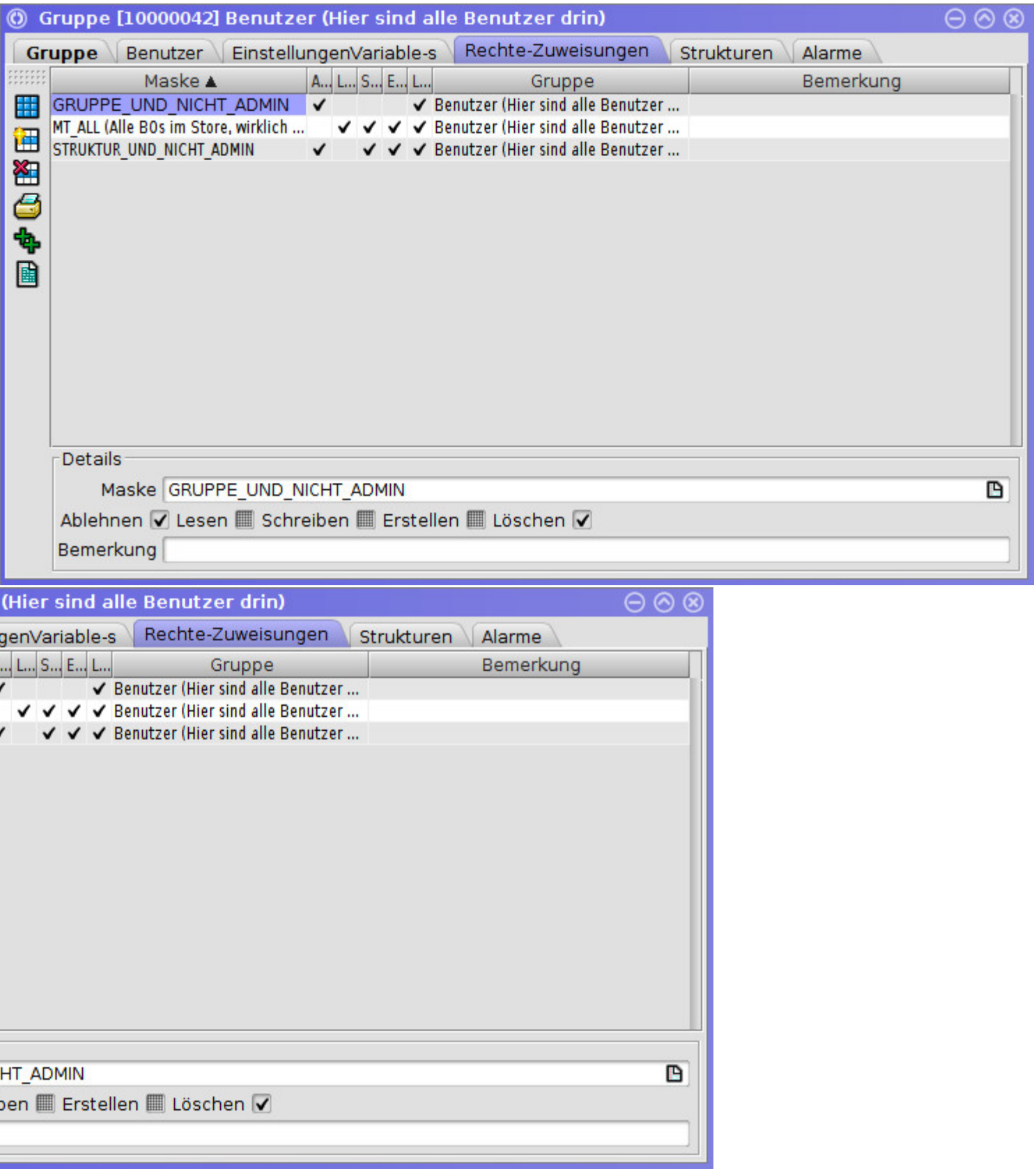
```
bo.Benutzer.containsKey( user.Id )
```

CAUTION Definieren Sie nicht leichtfertig Skripte für BO-Masken. Das Skript muss für *jedes* Objekt vom Typ der Maske aufgerufen und ausgewertet werden, wenn ein entsprechendes Objekt gelesen, bearbeitet oder gelöscht werden soll. Dies kann sich - je nach Komplexität der Skripte - negativ auf die Leistung der MyTISM-Instanz auswirken.

Rechtezuweisungen

In den *Rechtezuweisungen* werden alle bisher erklärten Komponenten zusammengeführt; sie definieren *wer* (via Gruppen) mit *welchen Objekten* (via BO-Maske) *wie* (via gesetzer oder nicht gesetzter Flags an der Rechtezuweisung) arbeiten darf.

Am einfachsten lassen sich die Rechtezuweisungen über den *"Rechte-Zuweisungen"*-Reiter im Gruppen-Formular bearbeiten.



Maske

Pflichtfeld - Hier wählt man die BO-Maske aus, die definiert, für welche Objekte man Rechte vergeben oder verweigern will.

Flags "Lesen", "Schreiben", "Erstellen" und "Löschen"

Wenn das entsprechende Flag *gesetzt* ist, wird - abhängig vom Flag *"Ablehnen"* - das entsprechende Recht für die von der Maske ausgewählten BOs gegeben oder verweigert.
Wenn das Flag *nicht gesetzt* ist (undefiniert oder nicht ausgewählt ***FIXME Sollte hier ggf. der "null"-Status nicht ausgeschlossen werden?***") wird das entsprechende Recht nicht "berührt".
Es muss mindestens eines der Flags gesetzt sein, damit die Rechtezuweisung sinnvoll ist.

Flag "Ablehnen"

Wenn dieses Flag *nicht gesetzt* ist, werden die durch die anderen Flags definierten Rechte *gewährt*.
Wenn dieses Flag *gesetzt* ist, werden diese Rechte allerdings *entzogen*.
Wenn für ein Objekt durch irgendeine Rechtezuweisung ein Recht entzogen wurde ist dies bindend, egal durch wieviele andere Rechtezuweisungen das Recht ggf. ansonsten gewährt werden würde.

Bemerkung

Optional - Hier kann man Erklärungen oder Erläuterungen zu dieser speziellen Rechtezuweisung unterbringen.

Example 2. Benutzer dürfen Kundendaten sehen/lesen

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

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 "menschenlesbaren" Kommentar eintragen, der aussagt, was diese Rechtezuweisung genau bewirkt oder warum sie angelegt wurde. 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!

Example 3. Chef darf Kundendaten bearbeiten

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:

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 BO-Maske `"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, würden Lesen, Schreiben, Erstellen oder Löschen *verbieten* - das setzen wir natürlich nicht.

Rechte vergeben

Es gibt prinzipiell zwei Methoden, um Rechte in einem MyTISM-System zu vergeben.

Blacklisting
: Standardmäßig alles erlauben und selektiv verbieten.

Whitelisting
: Standardmäßig alles verbieten und selektiv erlauben.

Blacklisting - Selektiv verbieten

Bei dieser Methode werden einer Gruppe erst einmal *alle Rechte* für *alle Objekte* gegeben. Danach werden diese Rechte selektiv eingeschränkt um bestimmte Dinge zu verbieten.

Vorteil
: Schnell und einfach einzurichten; Benutzer bekommen auf einfache Weise alle Rechte, die sie benötigen, um mit einem System zu arbeiten.

> Im Minimalfall reichen eine Gruppe, eine BO-Maske und eine Rechtezuweisung.

Nachteil
: Benutzer bekommen standardmäßig viele Rechte, die sie nicht benötigen oder die sogar gefährlich sind.

> Wenn vergessen wird, vertrauliche Daten mit einer das Lesen verbietenden Rechtezuweisung zu schützen, können diese von allen Benutzern eingesehen werden.
> Wenn vergessen wird, wichtige Daten mit mit einer das Löschen verbietenden Rechtezuweisung zu schützen, können diese von allen Benutzern absichtlich oder versehentlich gelöscht werden.

Whitelisting - Selektiv erlauben

Bei dieser Methode erhalten Gruppen *nur für genau die Objekte, die sie für ihre Arbeit benötigen* nur *genau die erforderlichen Rechte*.

Vorteil
: Benutzer bekommen nur die passenden Rechte.

Vertrauliche Daten sind standardmäßig vor Einsicht geschützt.
Wichtige Daten können nicht von jedem gelöscht oder verändert werden.

Nachteil

Je nach System komplex und aufwändig einzurichten. Es müssen u.U. viele Gruppen, BO-Masken und Rechtezuweisungen angelegt und zugewiesen werden. Welche Gruppen man anlegen soll und welche Rechte sie bekommen sollen kann schwierig zu entscheiden sein.

Allein die Standardgruppe `"RG_Solstice_Login"` (Benutzer mit minimalen Rechten zum Login mit Solstice) hat 15 Rechtezuweisungen mit ebensovielen BO-Masken.

Fehlerbehebung

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 synchonisierendem 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. Authoritativer Server: Server stoppen "./mytism stop_mytism"
3. Authoritativer Server: Backup ziehen "./mytism backup" (Nur zur Sicherheit)
4. Authoritativer Server: Fixer laufen lassen "./mytism run de.ipcon.db.tools.DoubleIdFixer PROJEKTINSTANZ --repairAll fix_double_ids.sql" (PROJEKTINSTANZ z.B. ".oashi")
5. Authoritativer Server: checked-*-Dateien löschen "rm .checked-*"
6. Authoritativer Server: Generiertes SQL-Script einspielen "psql -U postgres DATENBANKNAME < fix_double_ids.sql"
7. Synchronisierender Server: Backup ziehen (Nur zur Sicherheit)
8. Authoritativer Server: bi-tabelle leeren "psql -U postgres DATENBANKNAME", dann dort "delete from bi"
9. Authoritativer Server: Server starten "./mytism start_mytism" (Diverse Aufräumarbeiten werden gemacht)
10. Authoritativer Server: Server stoppen "./mytism stop_mytism"
11. Authoritativer Server: Backup ziehen "./mytism backup"
12. Backup zum synchronisierenden Server kopieren
13. Authoritativer 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 authoritativen 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. "… logSyncRemoteToLocal(…) …" Ausschau halten; ggf. selbst mal eine Änderung an einem Objekt auf einem der Server machen).

FAQ - Immer wiederkehrende Fragen und deren Beantwortung

FIXME