kivitendo: Installation, Konfiguration, Entwicklung

kivitendo: Installation, Konfiguration, Entwicklung

Inhaltsverzeichnis

1

Aktuelle Hinweise

Aktuelle Installations- und Konfigurationshinweise gibt es:

- im kivitendo-Forum: https://forum.kivitendo.org/
- im alten Lx-Office-Wiki unter Dokumentation (http://wiki.lx-office.org/index.php?title=Installation_Lx-Office_ERP)

2

Installation und Grundkonfiguration

2.1. Benötigte Software und Pakete

2.1.1. Betriebssystem

kivitendo ist für Linux konzipiert, und sollte auf jedem unixoiden Betriebssystem zum Laufen zu kriegen sein. Getestet ist diese Version im speziellen auf Debian und Ubuntu, grundsätzlich wurde bei der Auswahl der Pakete aber darauf Rücksicht genommen, dass es ohne große Probleme auf den derzeit aktuellen verbreiteten Distributionen läuft.

Mitte 2012 sind das folgende Systeme, von denen bekannt ist, dass kivitendo auf ihnen läuft:

- Ubuntu 10.04 LTS Lucid Lynx bis 12.04 Precise Pangolin
- Debian 5.0 Lenny und 6.0 Squeeze
- openSUSE 11.2 und 11.3
- SuSE Linux Enterprice Server 11
- Fedora 13 bis 16

2.1.2. Pakete

Zum Betrieb von kivitendo werden zwingend ein Webserver (meist Apache) und ein Datenbankserver (PostgreSQL, mindestens v8.2) benötigt.

Zusätzlich benötigt kivitendo die folgenden Perl-Pakete, die nicht Bestandteil einer Standard-Perl-Installation sind:

- parent
- Archive::Zip
- Config::Std
- DateTime
- DBI
- DBD::Pg
- Email::Address
- JSON
- List::MoreUtils
- Params::Validate
- PDF::API2

- Rose::Object
- Rose::DB
- Rose::DB::Object
- Template
- Text::CSV_XS
- Text::Iconv
- URI
- XML::Writer
- YAML

Gegenüber Version 2.6.0 sind zu dieser Liste 2 Pakete hinzugekommen, `URI` und `XML::Writer` sind notwendig. Ohne startet kivitendo nicht.

Gegenüber Version 2.6.1 sind `parent`, `DateTime`, `Rose::Object`, `Rose::DB` und `Rose::DB::Object` neu hinzugekommen. `IO::Wrap` wurde entfernt.

Gegenüber Version 2.6.3 ist `JSON` neu hinzugekommen.

`Email::Address` und `List::MoreUtils` sind schon länger feste Abhängigkeiten, wurden aber bisher mit kivitendo mitgeliefert. Beide sind auch in 2.6.1 weiterhin mit ausgeliefert, wurden in einer zukünftigen Version aber aus dem Paket entfernt werden. Es wird empfohlen diese Module zusammen mit den anderen als Bibliotheken zu installieren.

Die zu installierenden Pakete können in den verschiedenen Distributionen unterschiedlich heißen.

Für Debian oder Ubuntu benötigen Sie diese Pakete:

```
apt-get install apache2 postgresql libparent-perl libarchive-zip-perl \
  libdatetime-perl libdbi-perl libdbd-pg-perl libpg-perl \
  libemail-address-perl liblist-moreutils-perl libpdf-api2-perl \
  librose-object-perl librose-db-perl librose-db-object-perl \
  libtemplate-perl libtext-csv-xs-perl libtext-iconv-perl liburi-perl \
  libxml-writer-perl libyaml-perl libconfig-std-perl \
  libparams-validate-perl libjson-perl libclass-accessor-perl
```

Für Fedora Core benötigen Sie diese Pakete:

```
yum install httpd postgresql-server perl-parent perl-DateTime \
  perl-DBI perl-DBD-Pg perl-Email-Address perl-List-MoreUtils \
  perl-PDF-API2 perl-Rose-Object perl-Rose-DB perl-Rose-DB-Object \
  perl-Template-Toolkit perl-Text-CSV_XS perl-Text-Iconv perl-URI \
  perl-XML-Writer perl-YAML
```

Für OpenSuSE benötigen Sie diese Pakete:

```
zypper install apache2 postgresql-server perl-Archive-Zip \
  perl-DateTime perl-DBI perl-DBD-Pg perl-MailTools perl-List-MoreUtils \
  perl-PDF-API2 perl-Template-Toolkit perl-Text-CSV_XS perl-Text-Iconv \
  perl-URI perl-XML-Writer perl-YAML
```

Bei openSuSE 11 ist `parent` bereits enthalten, und braucht nicht nachinstalliert werden. Die `Rose::*` Pakete sind derzeit nicht für SuSE gepackt, und müssen anderweitig nachinstalliert werden.

kivitendo enthält ein Script, mit dem überprüft werden kann, ob alle benötigten Perl-Module installiert sind. Der Aufruf lautet wie folgt:

```
./scripts/installation_check.pl
```

2.2. Manuelle Installation des Programmpaketes

Die kivitendo ERP Installationsdatei (kivitendo-erp-2.6.3.tgz) wird im Dokumentenverzeichnis des Webservers (z.B. `/var/www/html/`, `/srv/www/htdocs` oder `/var/www/`) entpackt:

```
cd /var/www
tar xvzf kivitendo-erp-2.6.3.tgz
```

Wechseln Sie in das entpackte Verzeichnis:

```
cd kivitendo-erp
```

Alternativ können Sie auch einen Alias in der Webserverkonfiguration benutzen, um auf das tatsächliche Installationsverzeichnis zu verweisen.

Die Verzeichnisse `users`, `spool` und `webdav` müssen für den Benutzer beschreibbar sein, unter dem der Webserver läuft. Die restlichen Dateien müssen für diesen Benutzer lesbar sein. Die Benutzer- und Gruppennamen sind bei verschiedenen Distributionen unterschiedlich (z.B. bei Debian/Ubuntu `www-data`, bei Fedora core `apache` oder bei OpenSuSE `wwwrun`).

Der folgende Befehl ändert den Besitzer für die oben genannten Verzeichnisse auf einem Debian/Ubuntu-System:

```
chown -R www-data users spool webdav
```

Weiterhin muss der Webserver-Benutzer in den Verzeichnissen `templates` und `users` Unterverzeichnisse für jeden neuen Benutzer anlegen dürfen, der in kivitendo angelegt wird:

```
chown www-data templates users
```

2.3. kivitendo-Konfigurationsdatei

2.3.1. Einführung

In kivitendo gibt es nur noch eine Konfigurationsdatei, die benötigt wird: `config/kivitendo.conf` (kurz: "die Hauptkonfigurationsdatei"). Diese muss bei der Erstinstallation von kivitendo bzw. der Migration von älteren Versionen angelegt werden.

Als Vorlage dient die Datei `config/kivitendo.conf.default` (kurz: "die Default-Datei"):

```
$ cp config/kivitendo.conf.default config/kivitendo.conf
```

Die Default-Datei wird immer zuerst eingelesen. Werte, die in der Hauptkonfigurationsdatei stehen, überschreiben die Werte aus der Default-Datei. Die Hauptkonfigurationsdatei muss also nur die Abschnitte und Werte enthalten, die von denen der Default-Datei abweichen.

Anmerkung

Vor der Umbenennung in kivitendo hieß diese Datei noch `config/lx_office.conf`. Aus Gründen der Kompatibilität wird diese Datei eingelesen, sofern die Datei `config/kivitendo.conf` nicht existiert.

Diese Hauptkonfigurationsdatei ist dann eine installationsspezifische Datei, d.h. sie enthält bspw. lokale Passwörter und wird auch nicht im Versionsmanagement (git) verwaltet.

Die Konfiguration ist ferner serverabhängig, d.h. für alle Mandaten, bzw. Datenbanken gleich.

2.3.2. Abschnitte und Parameter

Die Konfigurationsdatei besteht aus mehreren Teilen, die entsprechend kommentiert sind:

- `authentication`
- `authentication/database`
- `authentication/ldap`
- `system`
- `features`
- `paths`
- `applications`
- `environment`
- `print_templates`
- `task_server`
- `periodic_invoices`
- `console`
- `debug`

Die üblicherweise wichtigsten Parameter, die am Anfang einzustellen oder zu kontrollieren sind, sind:

```
[authentication]
admin_password = geheim

[authentication/database]
host     = localhost
port     = 5432
db       = kivitendo_auth
user     = postgres
password =

[system]
dbcharset = UTF-8
```

Nutzt man wiederkehrende Rechnungen, kann man unter `[periodic_invoices]` den Login eines Benutzers angeben, der nach Erstellung der Rechnungen eine entsprechende E-Mail mit Informationen über die erstellten Rechnungen bekommt.

Nutzt man den Taskserver für wiederkehrende Rechnungen, muss unter `[task_server]` ein Login eines Benutzers angegeben werden, mit dem sich der Taskserver an kivitendo bei der Datenbank anmeldet, die dem Benutzer zugewiesen ist.

Für Entwickler finden sich unter `[debug]` wichtige Funktionen, um die Fehlersuche zu erleichtern.

2.3.3. Versionen vor 2.6.3

In älteren kivitendo Versionen gab es im Verzeichnis `config` die Dateien `authentication.pl` und `lx-erp.conf`, die jeweils Perl-Dateien waren. Es gab auch die Möglichkeit, eine lokale Version der Konfigurationsdatei zu erstellen (`lx-erp-local.conf`). Dies ist ab 2.6.3 nicht mehr möglich, aber auch nicht mehr nötig.

Beim Update von einer kivitendo-Version vor 2.6.3 auf 2.6.3 oder jünger müssen die Einstellungen aus den alten Konfigurationsdateien manuell übertragen und die alten Konfigurationsdateien anschließend gelöscht oder verschoben werden. Ansonsten zeigt kivitendo eine entsprechende Fehlermeldung an.

2.4. Anpassung der PostgreSQL-Konfiguration

PostgreSQL muss auf verschiedene Weisen angepasst werden.

2.4.1. Zeichensätze/die Verwendung von UTF-8

Bei aktuellen Serverinstallationen braucht man hier meist nicht eingreifen

Dieses kann überprüft werden: ist das Encoding der Datenbank “template1” “UTF8”, so braucht man nichts weiteres diesbezueglich unternehmen. Zum Testen:

```
su postgres
echo '\l' | psql
exit
```

Andernfalls ist es notwendig, einen neuen Datenbankcluster mit UTF-8-Encoding anzulegen und diesen zu verwenden. Unter Debian und Ubuntu kann dies z.B. für PostgreSQL 8.2 mit dem folgenden Befehl getan werden:

```
pg_createcluster --locale=de_DE.UTF-8 --encoding=UTF-8 8.2 clustername
```

Die Datenbankversionsnummer muss an die tatsächlich verwendete Versionsnummer angepasst werden.

Unter anderen Distributionen gibt es ähnliche Methoden.

Wurde PostgreSQL nicht mit UTF-8 als Encoding initialisiert und ist ein Neuanlegen eines weiteren Clusters nicht möglich, so kann kivitendo mit ISO-8859-15 als Encoding betrieben werden.

Das Encoding einer Datenbank kann in **psql** mit `\l` geprüft werden.

2.4.2. Änderungen an Konfigurationsdateien

In der Datei `postgresql.conf`, die je nach Distribution in verschiedenen Verzeichnissen liegen kann (z.B. `/var/lib/pgsql/data/` oder `/etc/postgresql/`, muss sichergestellt werden, dass TCP/IP-Verbindungen aktiviert sind. Das Verhalten wird über den Parameter `listen_address` gesteuert. Laufen PostgreSQL und kivitendo auf demselben Rechner, so kann dort der Wert `localhost` verwendet werden. Andernfalls müssen Datenbankverbindungen auch von anderen Rechnern aus zugelassen werden, was mit dem Wert `*` geschieht.

In der Datei `pg_hba.conf`, die im gleichen Verzeichnis wie die `postgresql.conf` zu finden sein sollte, müssen die Berichtigungen für den Zugriff geändert werden. Hier gibt es mehrere Möglichkeiten. sinnvoll ist es nur die nögiten Verbindungen immer zuzulassen, für eine lokal laufenden Datenbank zum Beispiel:

```
local all kivitendo password
host all kivitendo 127.0.0.1 255.255.255.255 password
```

2.4.3. Erweiterung für servergespeicherte Prozeduren

In der Datenbank `template1` muss die Unterstützung für servergespeicherte Prozeduren eingerichet werden. Melden Sie sich dafür als Benutzer “postgres” an der Datenbank an:

```
su - postgres
psql template1
```

führen Sie die folgenden Kommandos aus:

```
create language 'plpgsql';
\q
```

2.4.4. Datenbankbenutzer anlegen

Wenn Sie nicht den Datenbanksuperuser “postgres” zum Zugriff benutzen wollen, so sollten Sie bei PostgreSQL einen neuen Benutzer anlegen. Ein Beispiel, wie Sie einen neuen Benutzer anlegen können:

```
su - postgres
createuser -d -P kivitendo
```

```
exit
```

Wenn Sie später einen Datenbankzugriff konfigurieren, verändern Sie den evtl. voreingestellten Benutzer “postgres” auf “kivitendo” bzw. den hier gewählten Benutzernamen.

2.5. Webserver-Konfiguration

2.5.1. Grundkonfiguration mittels CGI

Anmerkung

Für einen deutlichen Performanceschub sorgt die Ausführung mittels FastCGI/FCGI. Die Einrichtung wird ausführlich im Abschnitt Konfiguration für FastCGI/FCGI [7] beschrieben.

Der Zugriff auf das Programmverzeichnis muss in der Apache Webserverkonfigurationsdatei `httpd.conf` eingestellt werden. Fügen Sie den folgenden Abschnitt dieser Datei oder einer anderen Datei hinzu, die beim Starten des Webservers eingelesen wird:

```
AddHandler cgi-script .pl
Alias /kivitendo-erp/ /var/www/kiviteno-erp/

<Directory /var/www/kivitendo-erp>
 Options ExecCGI Includes FollowSymlinks
</Directory>

<Directory /var/www/kivitendo-erp/users>
 Order Deny,Allow
 Deny from All
</Directory>
```

Ersetzen Sie dabei die Pfade durch diejenigen, in die Sie vorher das kivitendo-Archiv entpacket haben.

Anmerkung

Vor den einzelnen Optionen muss bei einigen Distributionen ein Plus ‘+’ gesetzt werden.

Auf einigen Webservern werden manchmal die Grafiken und Style-Sheets nicht ausgeliefert. In solchen Fällen hat es oft geholfen, die folgende Option in die Konfiguration aufzunehmen:

```
EnableSendfile Off
```

2.5.2. Konfiguration für FastCGI/FCGI

2.5.2.1. Was ist FastCGI?

Direkt aus Wikipedia[1] kopiert:

[FastCGI ist ein Standard für die Einbindung externer Software zur Generierung dynamischer Webseiten in einem Webserver. FastCGI ist vergleichbar zum Common Gateway Interface (CGI), wurde jedoch entwickelt, um dessen Performance-Probleme zu umgehen.]

2.5.2.2. Warum FastCGI?

Perl Programme (wie kivitendo eines ist) werden nicht statisch kompiliert. Stattdessen werden die Quelldateien bei jedem Start übersetzt, was bei kurzen Laufzeiten einen Großteil der Laufzeit ausmacht. Während SQL Ledger einen Großteil der Funk-

[1] http://de.wikipedia.org/wiki/FastCGI

tionalität in einzelne Module kapselt, um immer nur einen kleinen Teil laden zu müssen, ist die Funktionalität von kivitendo soweit gewachsen, dass immer mehr Module auf den Rest des Programms zugreifen. Zusätzlich benutzen wir umfangreiche Bibliotheken um Funktionaltät nicht selber entwickeln zu müssen, die zusätzliche Ladezeit kosten. All dies führt dazu dass ein kivitendo Aufruf der Kernmasken mittlerweile deutlich länger dauert als früher, und dass davon 90% für das Laden der Module verwendet wird.

Mit FastCGI werden nun die Module einmal geladen, und danach wird nur die eigentliche Programmlogik ausgeführt.

2.5.2.3. Getestete Kombinationen aus Webservern und Plugin

Folgende Kombinationen sind getestet:

- Apache 2.2.11 (Ubuntu) und mod_fcgid.
- Apache 2.2.11 (Ubuntu) und mod_fastcgi.

Dabei wird mod_fcgid empfohlen, weil mod_fastcgi seit geraumer Zeit nicht mehr weiter entwickelt wird. Im Folgenden wird auf mod_fastcgi nicht mehr explizit eingegangen.

Als Perl Backend wird das Modul `FCGI.pm` verwendet.

Warnung

FCGI 0.69 und höher ist extrem strict in der Behandlung von Unicode, und verweigert bestimmte Eingaben von kivitendo. Falls es Probleme mit Umlauten in Ihrere Installation gibt, muss auf die Vorgängerversion FCGI 0.68 ausgewichen werden.

Mit CPAN lässt sie sich die Vorgängerversion wie folgt installieren:

```
force install M/MS/MSTROUT/FCGI-0.68.tar.gz
```

2.5.2.4. Konfiguration des Webservers

Bevor Sie versuchen, eine kivitendo Installation unter FCGI laufen zu lassen, empfliehlt es sich die Installation ersteinmal unter CGI aufzusetzen. FCGI macht es nicht einfach Fehler zu debuggen die beim ersten aufsetzen auftreten können. Sollte die Installation schon funktionieren, lesen Sie weiter.

Zuerst muss das FastCGI-Modul aktiviert werden. Dies kann unter Debian/Ubuntu z.B. mit folgendem Befehl geschehen:

```
a2enmod fcgid
```

Die Konfiguration für die Verwendung von kivitendo mit FastCGI erfolgt durch Anpassung der vorhandenen `Alias`- und `Directory`-Direktiven. Dabei wird zwischen dem Installationspfad von kivitendo im Dateisystem ("`/path/to/kivitendo-erp`") und der URL unterschieden, unter der kivitendo im Webbrowser erreichbar ist ("`/url/for/kivitendo-erp`").

Folgender Konfigurationsschnipsel funktioniert mit mod_fastcgi:

```
AliasMatch ^/url/for/kivitendo-erp/[^/]+\.pl /path/to/kivitendo-erp/dispatcher.fcgi
Alias       /url/for/kivitendo-erp/          /path/to/kivitendo-erp/

<Directory /path/to/kivitendo-erp>
  AllowOverride All
  Options ExecCGI Includes FollowSymlinks
  Order Allow,Deny
  Allow from All
</Directory>

<DirectoryMatch /path/to/kivitendo-erp/users>
  Order Deny,Allow
```

```
  Deny from All
</DirectoryMatch>
```

Seit mod_fcgid-Version 2.6.3 gelten sehr kleine Grenzen für die maximale Größe eines Requests. Diese sollte wie folgt hochgesetzt werden:

```
FcgidMaxRequestLen 10485760
```

Das ganze sollte dann so aussehen:

```
AddHandler fcgid-script .fpl
AliasMatch ^/url/for/kivitendo-erp/[^/]+\.pl /path/to/kivitendo-erp/dispatcher.fpl
Alias       /url/for/kivitendo-erp/          /path/to/kivitendo-erp/
FcgidMaxRequestLen 10485760

<Directory /path/to/kivitendo-erp>
  AllowOverride All
  Options ExecCGI Includes FollowSymlinks
  Order Allow,Deny
  Allow from All
</Directory>

<DirectoryMatch /path/to/kivitendo-erp/users>
  Order Deny,Allow
  Deny from All
</DirectoryMatch>
```

Hierdurch wird nur ein zentraler Dispatcher gestartet. Alle Zugriffe auf die einzelnen Scripte werden auf diesen umgeleitet. Dadurch, dass zur Laufzeit öfter mal Scripte neu geladen werden, gibt es hier kleine Performance-Einbußen.

Es ist möglich, die gleiche kivitendo Version parallel unter CGI und FastCGI zu betreiben. Dafür bleiben die Directorydirektiven wie oben beschrieben, die URLs werden aber umgeleitet:

```
# Zugriff über CGI
Alias       /url/for/kivitendo-erp               /path/to/kivitendo-erp

# Zugriff mit mod_fcgid:
AliasMatch ^/url/for/kivitendo-erp-fcgid/[^/]+\.pl /path/to/kivitendo-erp/dispatcher.fpl
Alias       /url/for/kivitendo-erp-fcgid/          /path/to/kivitendo-erp/
```

Dann ist unter `/url/for/kivitendo-erp/` die normale Version erreichbar, und unter `/url/for/kivitendo-erp-fcgid/` die FastCGI-Version.

2.6. Der Task-Server

Der Task-Server ist ein Prozess, der im Hintergrund läuft, in regelmäßigen Abständen nach abzuarbeitenden Aufgaben sucht und diese zu festgelegten Zeitpunkten abarbeitet (ähnlich wie Cron). Dieser Prozess wird bisher nur für die Erzeugung der wiederkehrenden Rechnungen benutzt, wird aber in Zukunft deutlich mehr Aufgaben übertragen bekommen.

2.6.1. Verfügbare und notwendige Konfigurationsoptionen

Die Konfiguration erfolgt über den Abschnitt `[task_server]` in der Datei `config/kivitendo.conf`. Die dort verfügbaren Optionen sind:

`login`
: gültiger kivitendo-Benutzername, der benutzt wird, um die zu verwendende Datenbankverbindung auszulesen. Der Benutzer muss in der Administration angelegt werden. Diese Option muss angegeben werden.

`run_as`
: Wird der Server vom Systembenutzer `root` gestartet, so wechselt er auf den mit `run_as` angegebenen Systembenutzer. Der Systembenutzer muss dieselben Lese- und Schreibrechte haben, wie auch der Webserverbenutzer (siehe see Manuelle Installation des Programmpaketes [4]). Daher ist es sinnvoll, hier denselben Systembenutzer einzutragen, unter dem auch der Webserver läuft.

`debug`
: Schaltet Debug-Informationen an und aus.

2.6.2. Automatisches Starten des Task-Servers beim Booten

Der Task-Server verhält sich von seinen Optionen her wie ein reguläres SystemV-kompatibles Boot-Script. Außerdem wechselt er beim Starten automatisch in das kivitendo-Installationsverzeichnis.

Deshalb ist es möglich, ihn durch Setzen eines symbolischen Links aus einem der Runlevel-Verzeichnisse heraus in den Boot-Prozess einzubinden. Da das bei neueren Linux-Distributionen aber nicht zwangsläufig funktioniert, werden auch Start-Scripte mitgeliefert, die anstelle eines symbolischen Links verwendet werden können.

2.6.2.1. SystemV-basierende Systeme (z.B. Debian, OpenSuSE, Fedora Core)

Kopieren Sie die Datei `scripts/boot/system-v/kivitendo-server` nach `/etc/init.d/kivitendo-server`. Passen Sie in der kopierten Datei den Pfad zum Task-Server an (Zeile `DAEMON=...`). Binden Sie das Script in den Boot-Prozess ein. Dies ist distributionsabhängig:

- Debian-basierende Systeme:

```
update-rc.d kivitendo-task-server defaults
# Nur bei Debian Squeeze und neuer:
insserv kivitendo-task-server
```

- OpenSuSE und Fedora Core:

```
chkconfig --add kivitendo-task-server
```

Danach kann der Task-Server mit dem folgenden Befehl gestartet werden: **/etc/init.d/kivitendo-task-server start**

2.6.2.2. Upstart-basierende Systeme (z.B. Ubuntu)

Kopieren Sie die Datei `scripts/boot/upstart/kivitendo-task-server.conf` nach `/etc/init/kivitendo-task-server.conf`. Passen Sie in der kopierten Datei den Pfad zum Task-Server an (Zeile `exec ...`).

Danach kann der Task-Server mit dem folgenden Befehl gestartet werden: **service kivitendo-task-server start**

2.6.3. Wie der Task-Server gestartet und beendet wird

Der Task-Server wird wie folgt kontrolliert:

```
./scripts/task_server.pl Befehl
```

`Befehl` ist dabei eine der folgenden Optionen:

- `start` startet eine neue Instanz des Task-Servers. Die Prozess-ID wird innerhalb des `users`-Verzeichnisses abgelegt.
- `stop` beendet einen laufenden Task-Server.
- `restart` beendet und startet ihn neu.

- `status` berichtet, ob der Task-Server läuft.

Der Task-Server wechselt beim Starten automatisch in das kivitendo-Installationsverzeichnis.

Dieselben Optionen können auch für die SystemV-basierenden Runlevel-Scripte benutzt werden (siehe oben).

2.6.4. Task-Server mit mehreren Mandanten

Beim Task-Server wird der Login-Name des Benutzers, unter dem der Task-Server laufen soll, in die Konfigurationsdatei geschrieben. Hat man mehrere Mandanten muß man auch mehrere Konfigurationsdateien anlegen.

Die Konfigurationsdatei ist eine Kopie der Datei kivitendo.conf, wo in der Kategorie [task_server] der gewünschte "login" steht.

Der alternative Task-Server wird dann mit folgendem Befehl gestartet:

```
./scripts/task_server.pl -c config/DATEINAME.conf
```

2.7. Benutzerauthentifizierung und Administratorpasswort

Informationen über die Einrichtung der Benutzerauthentifizierung, über die Verwaltung von Gruppen und weitere Einstellungen

2.7.1. Grundlagen zur Benutzerauthentifizierung

kivitendo verwaltet die Benutzerinformationen in einer Datenbank, die im folgenden “Authentifizierungsdatenbank” genannt wird. Für jeden Benutzer kann dort eine eigene Datenbank für die eigentlichen Finanzdaten hinterlegt sein. Diese beiden Datenbanken können, müssen aber nicht unterschiedlich sein.

Im einfachsten Fall gibt es für kivitendo nur eine einzige Datenbank, in der sowohl die Benutzerinformationen als auch die Daten abgelegt werden.

Zusätzlich ermöglicht es kivitendo, dass die Benutzerpasswörter entweder gegen die Authentifizierungsdatenbank oder gegen einen LDAP-Server überprüft werden.

Welche Art der Passwortüberprüfung kivitendo benutzt und wie kivitendo die Authentifizierungsdatenbank erreichen kann, wird in der Konfigurationsdatei `config/kivitendo.conf` festgelegt. Diese muss bei der Installation und bei einem Upgrade von einer Version vor v2.6.0 angelegt werden. Eine Beispielkonfigurationsdatei `config/kivitendo.conf.default` existiert, die als Vorlage benutzt werden kann.

2.7.2. Administratorpasswort

Das Passwort, das zum Zugriff auf das Aministrationsinterface benutzt wird, wird ebenfalls in dieser Datei gespeichert. Es kann auch nur dort und nicht mehr im Administrationsinterface selber geändert werden. Der Parameter dazu heißt `admin_password` im Abschnitt `[authentication]`.

2.7.3. Authentifizierungsdatenbank

Die Verbindung zur Authentifizierungsdatenbank wird mit den Parametern in `[authentication/database]` konfiguriert. Hier sind die folgenden Parameter anzugeben:

`host`
: Der Rechnername oder die IP-Adresse des Datenbankservers

`port`
: Die Portnummer des Datenbankservers, meist 5432

`db`
Der Name der Authentifizierungsdatenbank

`user`
Der Benutzername, mit dem sich kivitendo beim Datenbankserver anmeldet (z.B. "`postgres`")

`password`
Das Passwort für den Datenbankbenutzer

Die Datenbank muss noch nicht existieren. kivitendo kann sie automatisch anlegen (mehr dazu siehe unten).

2.7.4. Passwortüberprüfung

kivitendo unterstützt Passwortüberprüfung auf zwei Arten: gegen die Authentifizierungsdatenbank und gegen einen externen LDAP- oder Active-Directory-Server. Welche davon benutzt wird, regelt der Parameter `module` im Abschnitt `[authentication]`.

Sollen die Benutzerpasswörter in der Authentifizierungsdatenbank gespeichert werden, so muss der Parameter `module` den Wert `DB` enthalten. In diesem Fall können sowohl der Administrator als auch die Benutzer selber ihre Psaswörter in kivitendo ändern.

Soll hingegen ein externer LDAP- oder Active-Directory-Server benutzt werden, so muss der Parameter `module` auf `LDAP` gesetzt werden. In diesem Fall müssen zusätzliche Informationen über den LDAP-Server im Abschnitt `[authentication/ldap]` angegeben werden:

`host`
Der Rechnername oder die IP-Adresse des LDAP- oder Active-Directory-Servers. Diese Angabe ist zwingend erforderlich.

`port`
Die Portnummer des LDAP-Servers; meist 389.

`tls`
Wenn Verbindungsverschlüsselung gewünscht ist, so diesen Wert auf '`1`' setzen, andernfalls auf '`0`' belassen

`attribute`
Das LDAP-Attribut, in dem der Benutzername steht, den der Benutzer eingegeben hat. Für Active-Directory-Server ist dies meist '`sAMAccountName`', für andere LDAP-Server hingegen '`uid`'. Diese Angabe ist zwingend erforderlich.

`base_dn`
Der Abschnitt des LDAP-Baumes, der durchsucht werden soll. Diese Angabe ist zwingend erforderlich.

`filter`
Ein optionaler LDAP-Filter. Enthält dieser Filter das Wort `<%login%>`, so wird dieses durch den vom Benutzer eingegebenen Benutzernamen ersetzt. Andernfalls wird der LDAP-Baum nach einem Element durchsucht, bei dem das oben angegebene Attribut mit dem Benutzernamen identisch ist.

`bind_dn` und `bind_password`
Wenn der LDAP-Server eine Anmeldung erfordert, bevor er durchsucht werden kann (z.B. ist dies bei Active-Directory-Servern der Fall), so kann diese hier angegeben werden. Für Active-Directory-Server kann als '`bind_dn`' entweder eine komplette LDAP-DN wie z.B. '`cn=Martin Mustermann,cn=Users,dc=firmendomain`' auch nur der volle Name des Benutzers eingegeben werden; in diesem Beispiel also '`Martin Mustermann`'.

2.7.5. Name des Session-Cookies

Sollen auf einem Server mehrere kivitendo-Installationen aufgesetzt werden, so müssen die Namen der Session-Cookies für alle Installationen unterschiedlich sein. Der Name des Cookies wird mit dem Parameter `cookie_name` im Abschnitt `[authentication]`gesetzt.

Diese Angabe ist optional, wenn nur eine Installation auf dem Server existiert.

2.7.6. Anlegen der Authentifizierungsdatenbank

Nachdem alle Einstellungen in `config/kivitendo.conf` vorgenommen wurden, muss kivitendo die Authentifizierungsdatenbank anlegen. Dieses geschieht automatisch, wenn Sie sich im Administrationsmodul anmelden, das unter der folgenden URL erreichbar sein sollte:

http://localhost/kivitendo-erp/admin.pl

2.8. Benutzer- und Gruppenverwaltung

Nach der Installation müssen Benutzer, Gruppen und Datenbanken angelegt werden. Dieses geschieht im Administrationsmenü, das Sie unter folgender URL finden:

http://localhost/kivitendo-erp/admin.pl

Verwenden Sie zur Anmeldung das Password, dass Sie in der Datei `config/kivitendo.conf` eingetragen haben.

2.8.1. Zusammenhänge

kivitendo verwendet eine Datenbank zum Speichern all seiner Informationen wie Kundendaten, Artikel, Angebote, Rechnungen etc. Um mit kivitendo arbeiten zu können, muss eine Person einen Benutzeraccount haben. Jedem Benutzeraccount wiederum wird genau eine Datenbank zugewiesen, mit der dieser Benutzer arbeiten kann. Es ist möglich und normal, dass mehreren Benutzern die selbe Datenbank zugewiesen wird, sodass sie alle mit den selben Daten arbeiten können.

Die Basisdaten der Benutzer, die in der Administration eingegeben werden können, werden in einer zweiten Datenbank gespeichert, der bereits erwähnten Authentifizierungsdatenbank. Diese ist also den Produktivdaten enthaltenden Datenbanken vorgeschaltet. Pro kivitendo-Installation gibt es nur eine Authentifizierungsdatenbank, aber beliebig viele Datenbanken mit Firmendaten.

kivitendo kann seinen Benutzern Zugriff auf bestimmte Funktionsbereiche erlauben oder verbieten. Wird der Zugriff nicht gestattet, so werden der entsprechenden Menüpunkte auch nicht angezeigt. Diese Rechte werden ebenfalls in der Authentifizierungsdatenbank gespeichert.

Um Rechte verteilen zu können, verwendet kivitendo ein Gruppen-Prinzip. Einer Gruppe kann der Zugriff auf bestimmte Bereiche erlaubt werden. Ein Benutzer wiederum kann Mitglied in einer oder mehrerer Gruppen sein. Der Benutzer hat Zugriff auf alle diejenigen Funktionen, die mindestens einer Gruppe erlaubt sind, in der der Benutzer Mitglied ist.

Die allgemeine Reihenfolge, in der Datenbanken, Gruppen und Benutzer angelegt werden sollten, lautet:

1. Datenbank anlegen
2. Gruppen anlegen
3. Benutzer anlegen
4. Benutzer den Gruppen zuordnen

2.8.2. Datenbanken anlegen

Zuerst muss eine Datenbank angelegt werden. Verwenden Sie für den Datenbankzugriff den vorhin angelegten Benutzer (in unseren Beispielen ist dies ‘`kivitendo`’).

Wenn Sie für die kivitendo-Installation nicht Unicode (UTF-8) sondern den europäischen Schriftsatz ISO-8859-15 benutzen wollen, so müssen Sie vor dem Anlegen der Datenbank in der Datei `config/kivitendo.conf` die Variable `dbcharset` im Abschnitt `system` auf den Wert ‘`ISO-8859-15`’ setzen.

Bitte beachten Sie, dass alle Datenbanken den selben Zeichensatz verwenden müssen, da diese Einstellungen momentan global in kivitendo vorgenommen wird und nicht nach Datenbank unterschieden werden kann. Auch die Authentifizierungsdatenbank muss mit diesem Zeichensatz angelegt worden sein.

2.8.3. Gruppen anlegen

Eine Gruppe wird in der Gruppenverwaltung angelegt. Ihr muss ein Name gegeben werden, eine Beschreibung ist hingegen optional. Nach dem Anlegen können Sie die verschiedenen Bereiche wählen, auf die Mitglieder dieser Gruppe Zugriff haben sollen.

Benutzergruppen sind unabhängig von Datenbanken, da sie in der Authentifizierungsdatenbank gespeichert werden. Sie gelten für alle Datenbanken, die in dieser Installation verwaltet werden.

2.8.4. Benutzer anlegen

Beim Anlegen von Benutzern werden für viele Parameter Standardeinstellungen vorgenommen, die den Gepflogenheiten des deutschen Raumes entsprechen.

Zwingend anzugeben sind der Loginname sowie die komplette Datenbankkonfiguration. Wenn die Passwortauthentifizierung über die Datenbank eingestellt ist, so kann hier auch das Benutzerpasswort gesetzt bzw. geändert werden. Ist hingegen die LDAP-Authentifizierung aktiv, so ist das Passwort-Feld deaktiviert.

In der Datenbankkonfiguration müssen die Zugriffsdaten einer der eben angelegten Datenbanken eingetragen werden.

2.8.5. Gruppenmitgliedschaften verwalten

Nach dem Anlegen von Benutzern und Gruppen müssen Benutzer den Gruppen zugewiesen werden. Dazu gibt es zwei Möglichkeiten:

1. In der Gruppenverwaltung wählt man eine Gruppe aus. Im folgenden Dialog kann man dann einzeln die Benutzer der Gruppe hinzufügen.

2. In der Gruppenverwaltung wählt man das Tool zur Verwaltung der Gruppenmitgliedschaft. Hier wird eine Matrix angezeigt, die alle im System angelegten Gruppen und Benutzer enthält. Durch Setzen der Häkchen wird der Benutzer in der ausgewählten Zeile der Gruppe in der ausgewählten Spalte hinzugefügt.

2.8.6. Migration alter Installationen

Wenn kivitendo 2.6.3 über eine ältere Version installiert wird, in der die Benutzerdaten noch im Dateisystem im Verzeichnis `users` verwaltet wurden, so bietet kivitendo die Möglichkeit, diese Benutzerdaten automatisch in die Authentifizierungsdatenbank zu übernehmen. Dies geschieht, wenn man sich nach dem Update der Installation das erste Mal im Administrationsbereich anmeldet. Findet kivitendo die Datei `users/members`, so wird der Migrationsprozess gestartet.

Der Migrationsprozess ist nahezu vollautomatisch. Alle Benutzerdaten können übernommen werden. Nach den Benutzerdaten bietet kivitendo noch die Möglichkeit an, dass automatisch eine Benutzergruppe angelegt wird. Dieser Gruppe wird Zugriff auf alle Funktionen von kivitendo gewährt. Alle migrierten Benutzern werden Mitglied in dieser Gruppe. Damit wird das Verhalten von kivitendo bis Version 2.4.3 inklusive wiederhergestellt, und die Benutzer können sich sofort wieder anmelden und mit dem System arbeiten.

2.9. Drucken mit kivitendo

Das Drucksystem von kivitendo benutzt von Haus aus LaTeX Vorlagen. Um drucken zu können, braucht der Server ein geeignetes LaTeX System. Am einfachsten ist dazu eine `texlive` Installation. Unter Debianoiden Betriebssystemen sind das die Pakete:

```
texlive-latex-base texlive-latex-extra texlive-fonts-recommended
```

Diese hinteren beiden enthalten Bibliotheken und Schriftarten die von den Standardvorlagen verwendet werden.

TODO: rpm Pakete.

In den allermeisten Installationen sollte drucken jetzt schon funktionieren. Sollte ein Fehler auftreten wirft TeX sehr lange Fehlerbeschreibungen, der eigentliche Fehler ist immer die erste Zeite die mit einem Ausrufezeichen anfängt. Häufig auftretende Fehler sind zum Beispiel:

- ! LaTeX Error: File `eurosym.sty' not found. Die entsprechende LaTeX-Bibliothek wurde nicht gefunden. Das tritt vor allem bei Vorlagen aus der Community auf. Installieren Sie die entsprechenden Pakete.
- ! Package inputenc Error: Unicode char \u8:æ¡# not set up for use with LaTeX. Dieser Fehler tritt auf, wenn sie versuchen mit einer Standardinstallation exotische utf8 Zeichen zu drucken. TeXLive unterstützt von Haus nur romanische Schriften und muss mit diversen Tricks dazu gebracht werden andere Zeichen zu akzeptieren. Adere TeX Systeme wie XeTeX schaffen hier Abhilfe.

Wird garkein Fehler angezeigt sondern nur der Name des Templates, heißt das normalerweise, dass das LaTeX Binary nicht gefunden wurde. Prüfen Sie den Namen in der Konfiguration (Standard: `pdflatex`), und stellen Sie sicher, dass pdflatex (oder das von Ihnen verwendete System) vom Webserver ausgeführt werden darf.

2.10. OpenDocument-Vorlagen

kivitendo unterstützt die Verwendung von Vorlagen im OpenDocument-Format, wie es OpenOffice.org ab Version 2 erzeugt. kivitendo kann dabei sowohl neue OpenDocument-Dokumente als auch aus diesen direkt PDF-Dateien erzeugen. Um die Unterstützung von OpenDocument-Vorlagen zu aktivieren muss in der Datei `config/kivitendo.conf` die Variable `opendocument` im Abschnitt `print_templates` auf '`1`' stehen. Dieses ist die Standardeinstellung.

Weiterhin muss in der Datei `config/kivitendo.conf` die Variable `dbcharset` im Abschnitt `system` auf die Zeichenkodierung gesetzt werden, die auch bei der Speicherung der Daten in der Datenbank verwendet wird. Diese ist in den meisten Fällen "UTF-8".

Während die Erzeugung von reinen OpenDocument-Dateien keinerlei weitere Software benötigt, wird zur Umwandlung dieser Dateien in PDF OpenOffice.org benötigt. Soll dieses Feature genutzt werden, so muss neben OpenOffice.org ab Version 2 auch der "X virtual frame buffer" (xvfb) installiert werden. Bei Debian ist er im Paket "xvfb" enthalten. Andere Distributionen enthalten ihn in anderen Paketen.

Nach der Installation müssen in der Datei `config/kivitendo.conf` zwei weitere Variablen angepasst werden: `openofficeorg_writer` muss den vollständigen Pfad zur OpenOffice.org Writer-Anwendung enthalten. `xvfb` muss den Pfad zum "X virtual frame buffer" enthalten. Beide stehen im Abschnitt `applications`.

Zusätzlich gibt es zwei verschiedene Arten, wie kivitendo mit OpenOffice kommuniziert. Die erste Variante, die benutzt wird, wenn die Variable `$openofficeorg_daemon` gesetzt ist, startet ein OpenOffice, das auch nach der Umwandlung des Dokumentes gestartet bleibt. Bei weiteren Umwandlungen wird dann diese laufende Instanz benutzt. Der Vorteil ist, dass die Zeit zur Umwandlung deutlich reduziert wird, weil nicht für jedes Dokument ein OpenOffice gestartet werden muss. Der Nachteil ist, dass diese Methode Python und die Python-UNO-Bindings benötigt, die Bestandteil von OpenOffice 2 sind.

Ist `$openofficeorg_daemon` nicht gesetzt, so wird für jedes Dokument OpenOffice neu gestartet und die Konvertierung mit Hilfe eines Makros durchgeführt. Dieses Makro muss in der Dokumentenvorlage enthalten sein und "Standard.Conversion.ConvertSelfToPDF()" heißen. Die Beispielvorlage '`templates/mastertemplates/German/invoice.odt`' enthält ein solches Makro, das in jeder anderen Dokumentenvorlage ebenfalls enthalten sein muss.

Als letztes muss herausgefunden werden, welchen Namen OpenOffice.org Writer dem Verzeichnis mit den Benutzereinstellungen gibt. Unter Debian ist dies momentan `~/.openoffice.org2`. Sollte der Name bei Ihrer OpenOffice.org-Installation anders sein, so muss das Verzeichnis `users/.openoffice.org2` entsprechend umbenannt werden. Ist der Name z.B. einfach nur `.openoffice`, so wäre folgender Befehl auszuführen:

```
mv users/.openoffice.org2 users/.openoffice
```

Dieses Verzeichnis, wie auch das komplette `users`-Verzeichnis, muss vom Webserver beschreibbar sein. Dieses wurde bereits erledigt (siehe Manuelle Installation des Programmpaketes [4]), kann aber erneut überprüft werden, wenn die Konvertierung nach PDF fehlschlägt.

2.11. Konfiguration zur Einnahmenüberschuss-rechnung/Bilanzierung: EUR

2.11.1. Einführung

kivitendo besaß bis inklusive Version 2.6.3 einen Konfigurationsparameter namens `eur`, der sich in der Konfigurationsdatei `config/lx_office.conf` befand. Somit galt er für alle Mandanten, die in dieser Installation benutzt wurden.

Mit der nachfolgenden Version wurde der Parameter zum Einen in die Mandantendatenbank verschoben und dabei auch gleich in drei Einzelparameter aufgeteilt, mit denen sich das Verhalten genauer steuern lässt.

2.11.2. Konfigurationsparameter

Es gibt drei Parameter, die die Gewinnermittlungsart, Versteuerungsart und die Warenbuchungsmethode regeln:

`profit_determination`
: Dieser Parameter legt die Berechnungsmethode für die Gewinnermittlung fest. Er enthält entweder `balance` für Betriebsvermögensvergleich/Bilanzierung oder `income` für die Einnahmen-Überschuss-Rechnung.

`accounting_method`
: Dieser Parameter steuert die Buchungs- und Berechnungsmethoden für die Versteuerungsart. Er enthält entweder `accrual` für die Soll-Versteuerung oder `cash` für die Ist-Versteuerung.

`inventory_system`
: Dieser Parameter legt die Warenbuchungsmethode fest. Er enthält entweder `perpetual` für die Bestandsmethode oder `periodic` für die Aufwandsmethode.

Zum Vergleich der Funktionalität bis und nach 2.6.3: `eur` = 1 bedeutete Einnahmen-Überschuss-Rechnung, Ist-Versteuerung und Aufwandsmethode. `eur` = 0 bedeutete hingegen Bilanzierung, Soll-Versteuerung und Bestandsmethode.

Die Konfiguration "`eur`" unter `[system]` in der Konfigurationsdatei `config/kivitendo.conf` wird nun nicht mehr benötigt und kann entfernt werden. Dies muss manuell geschehen.

2.11.3. Festlegen der Parameter

Beim Anlegen eines neuen Mandanten bzw. einer neuen Datenbank in der Admininstration können diese Optionen nun unabhängig voneinander eingestellt werden.

Beim Upgrade bestehender Mandanten wird eur ausgelesen und die Variablen werden so gesetzt, daß sich an der Funktionalität nichts ändert.

Die aktuelle Konfiguration wird unter Nummernkreise und Standardkonten unter dem neuen Punkt "Einstellungen" angezeigt (read-only). Eine spätere Änderung ist für einen bestehenden Mandanten nicht mehr möglich. Dies war auch vorher nicht möglich, bzw. vorhandene Daten wurden so belassen und haben damit die Ergebnisse verfälscht.

2.11.4. Bemerkungen zu Bestandsmethode

Die Bestandsmethode ist eigentlich eine sehr elegante Methode, funktioniert in kivitendo aber nur unter bestimmten Bedingungen: Voraussetzung ist, daß auch immer alle Einkaufsrechnungen gepflegt werden, und man beim Jahreswechsel nicht mit einer leeren Datenbank anfängt, da bei jedem Verkauf anhand der gesamten Rechnungshistorie der Einkaufswert der Ware nach dem FIFO-Prinzip aus den Einkaufsrechnungen berechnet wird.

Die Bestandsmethode kann vom Prinzip her also nur funktioneren, wenn man mit den Buchungen bei Null anfängt, und man kann auch nicht im laufenden Betrieb von der Aufwandsmethode zur Bestandsmethode wechseln.

2.11.5. Bekannte Probleme

Bei bestimmten Berichten kann man derzeit noch inviduell einstellen, ob man nach Ist- oder Sollversteuerung auswertet, und es werden im Code Variablen wie $accrual oder $cash gesetzt. Diese Codestellen wurden noch nicht angepasst, sondern nur die, wo bisher die Konfigurationsvariable `$::lx_office_conf{system}->{eur}` ausgewertet wurde.

Es fehlen Hilfetext beim Neuanlegen eines Mandanten, was die Optionen bewirken, z.B. mit zwei Standardfällen.

2.12. SKR04 19% Umstellung für innergemeinschaftlichen Erwerb

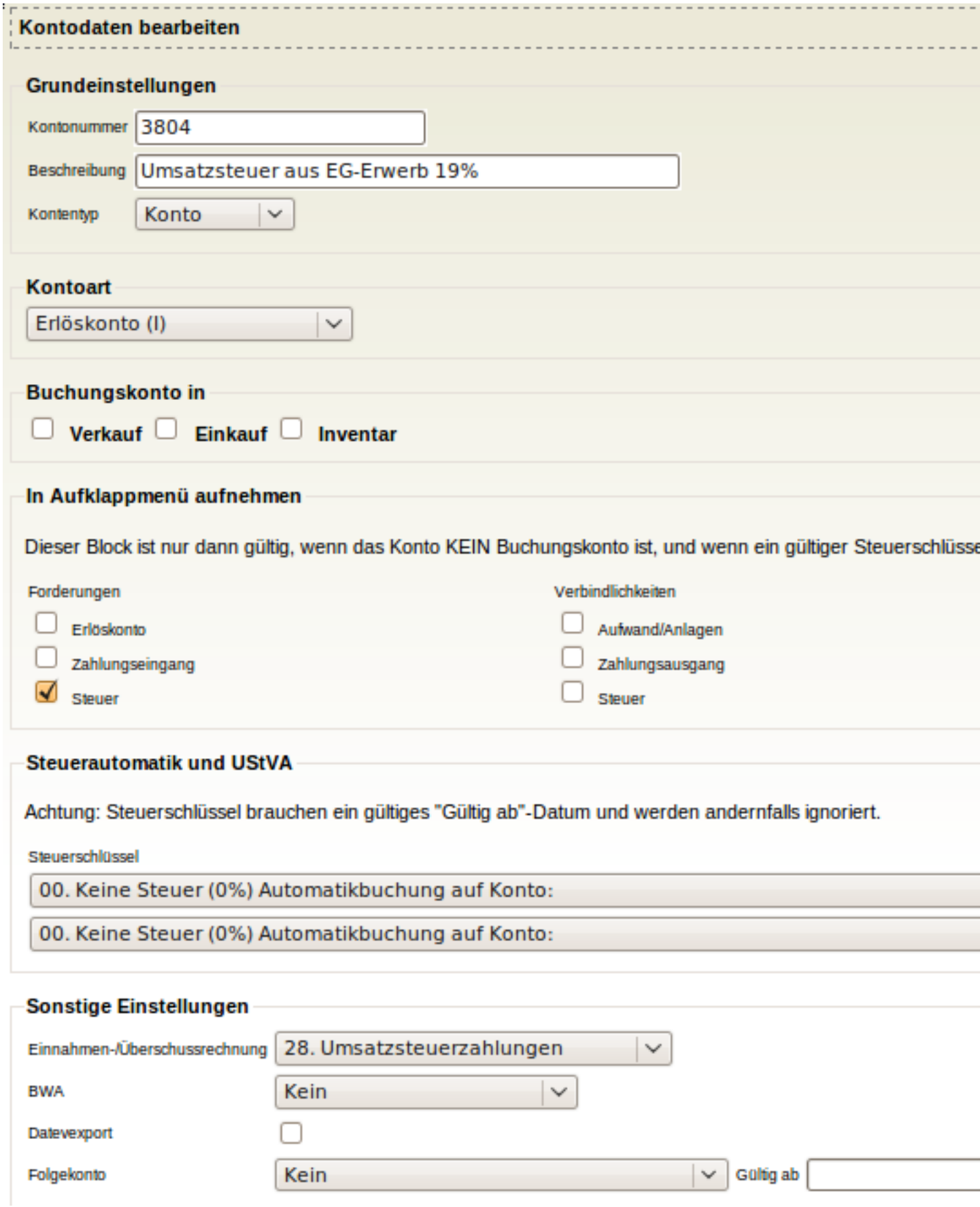
2.12.1. Einführung

Die Umsatzsteuerumstellung auf 19% für SKR04 für die Steuerschlüssel "EU ohne USt-ID Nummer" ist erst 2010 erfolgt. kivitendo beinhaltet ein Upgradeskript, das das Konto 3804 automatisch erstellt und die Steuereinstellungen korrekt einstellt. Hat der Benutzer aber schon selber das Konto 3804 angelegt, oder gab es schon Buchungen im Zeitraum nach dem 01.01.2007 auf das Konto 3803, wird das Upgradeskript vorsichtshalber nicht ausgeführt, da der Benutzer sich vielleicht schon selbst geholfen hat und mit seinen Änderungen zufrieden ist. Die korrekten Einstellungen kann man aber auch per Hand ausführen. Nachfolgend werden die entsprechenden Schritte anhand von Screenshots dargestellt.

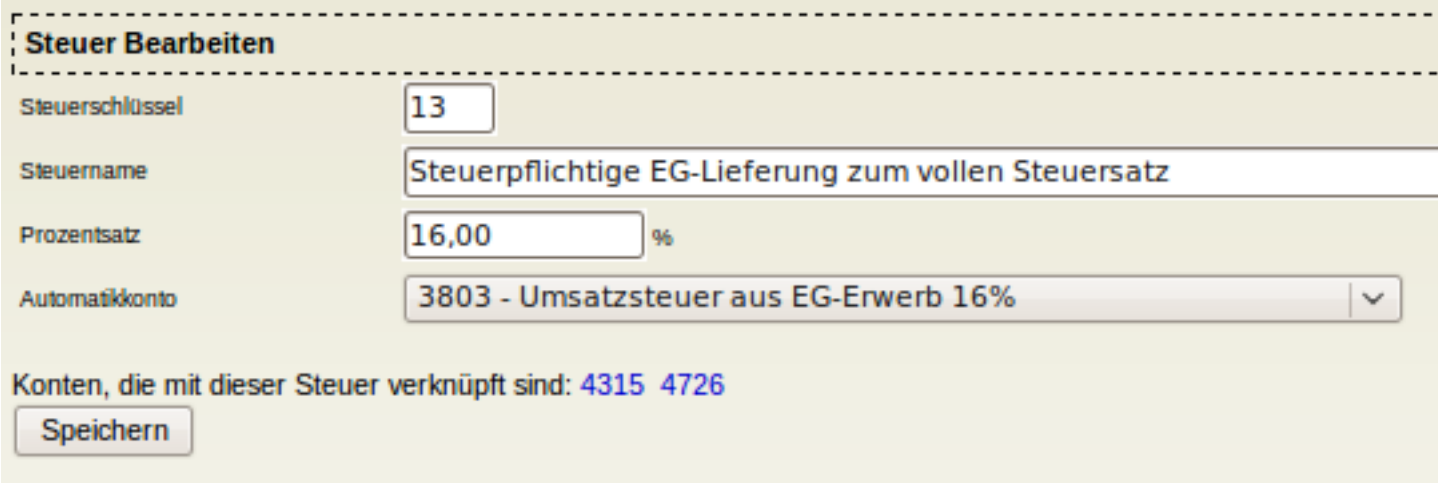
Für den Fall, daß Buchungen mit der Steuerschlüssel "EU ohne USt.-IdNr." nach dem 01.01.2007 erfolgt sind, ist davon auszugehen, dass diese mit dem alten Umsatzsteuersatz von 16% gebucht worden sind, und diese Buchungen sollten entsprechend kontrolliert werden.

2.12.2. Konto 3804 manuell anlegen

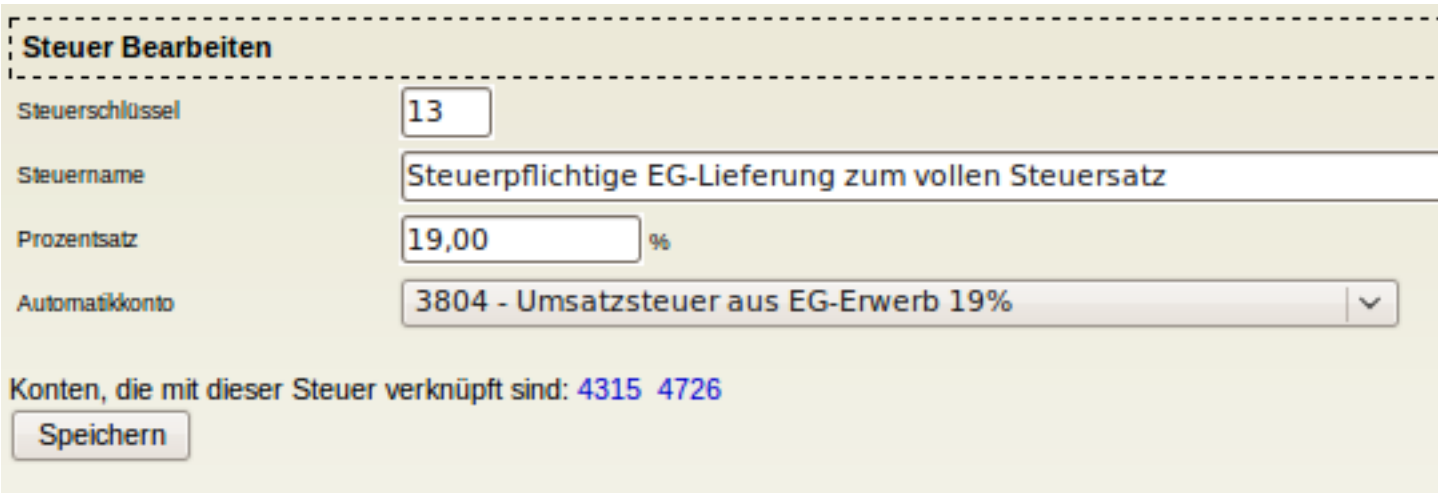
Die folgenden Schritte sind notwendig, um das Konto manuell anzulegen und zu konfigurieren. Zuerst wird in System -> Kontenübersicht -> Konto erfassen das Konto angelegt.


Kontodaten bearbeiten
Grundeinstellungen
Kontonummer
3804
Beschreibung
Umsatzsteuer aus EG-Erwerb 19%
Kontentyp
Konto
Kontoart
Erlöskonto (I)
Buchungskonto in
Verkauf
Einkauf
Inventar
In Aufklappmenü aufnehmen
Dieser Block ist nur dann gültig, wenn das Konto KEIN Buchungskonto ist, und wenn ein gültiger Steuerschlüsse
Forderungen
Erlöskonto
Zahlungseingang
Steuer
Verbindlichkeiten
Aufwand/Anlagen
Zahlungsausgang
Steuer
Steuerautomatik und UStVA
Achtung: Steuerschlüssel brauchen ein gültiges "Gültig ab"-Datum und werden andernfalls ignoriert.
Steuerschlüssel
00. Keine Steuer (0%) Automatikbuchung auf Konto:
00. Keine Steuer (0%) Automatikbuchung auf Konto:
Sonstige Einstellungen
Einnahmen-/Überschussrechnung
28. Umsatzsteuerzahlungen
BWA
Kein
Datevexport
Folgekonto
Kein
Gültig ab

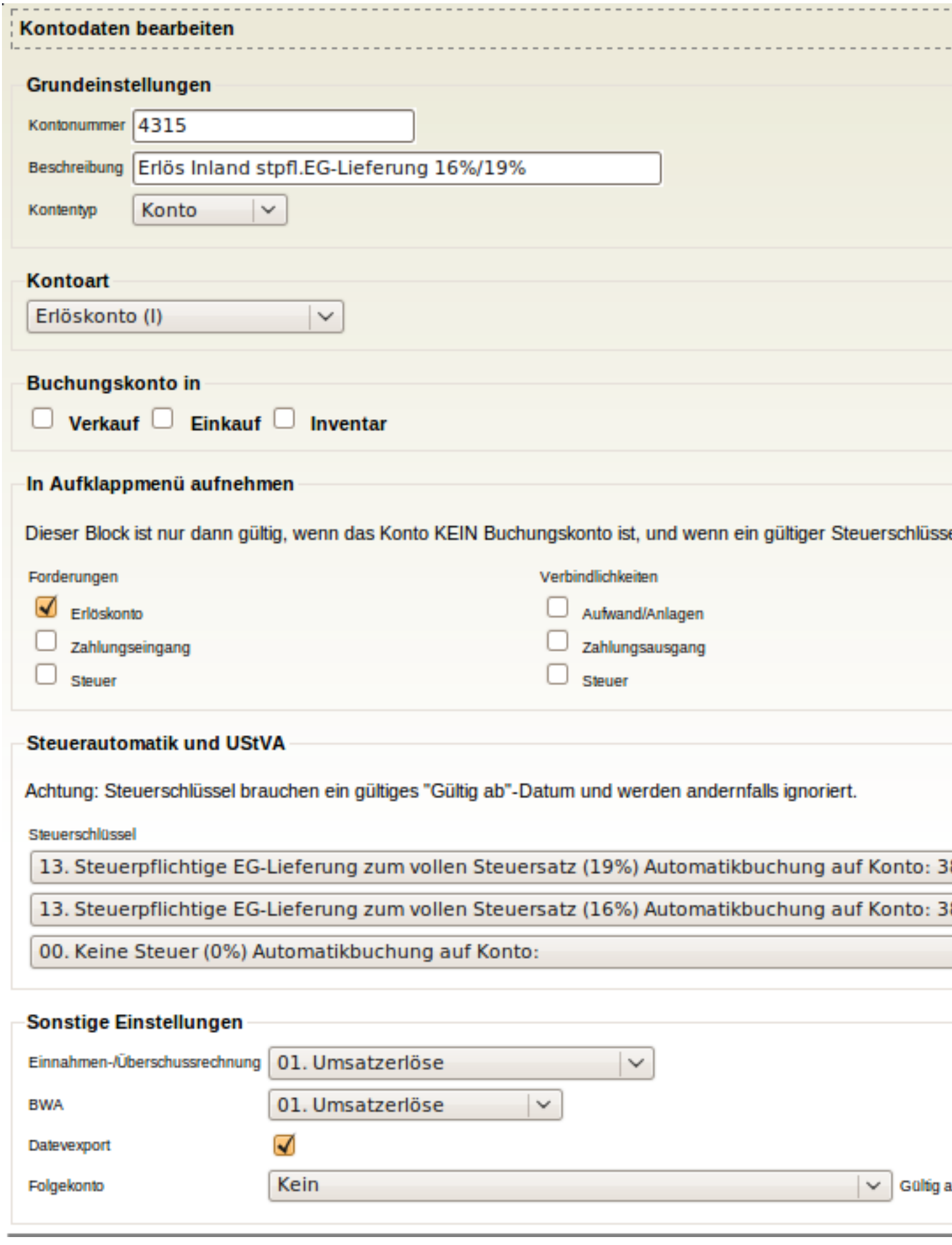
Als Zweites muss Steuergruppe 13 für Konto 3803 angepasst werden. Dazu unter System -> Steuern -> Bearbeiten den Eintrag mit Steuerschlüssel 13 auswählen und ihn wie im folgenden Screenshot angezeigt anpassen.



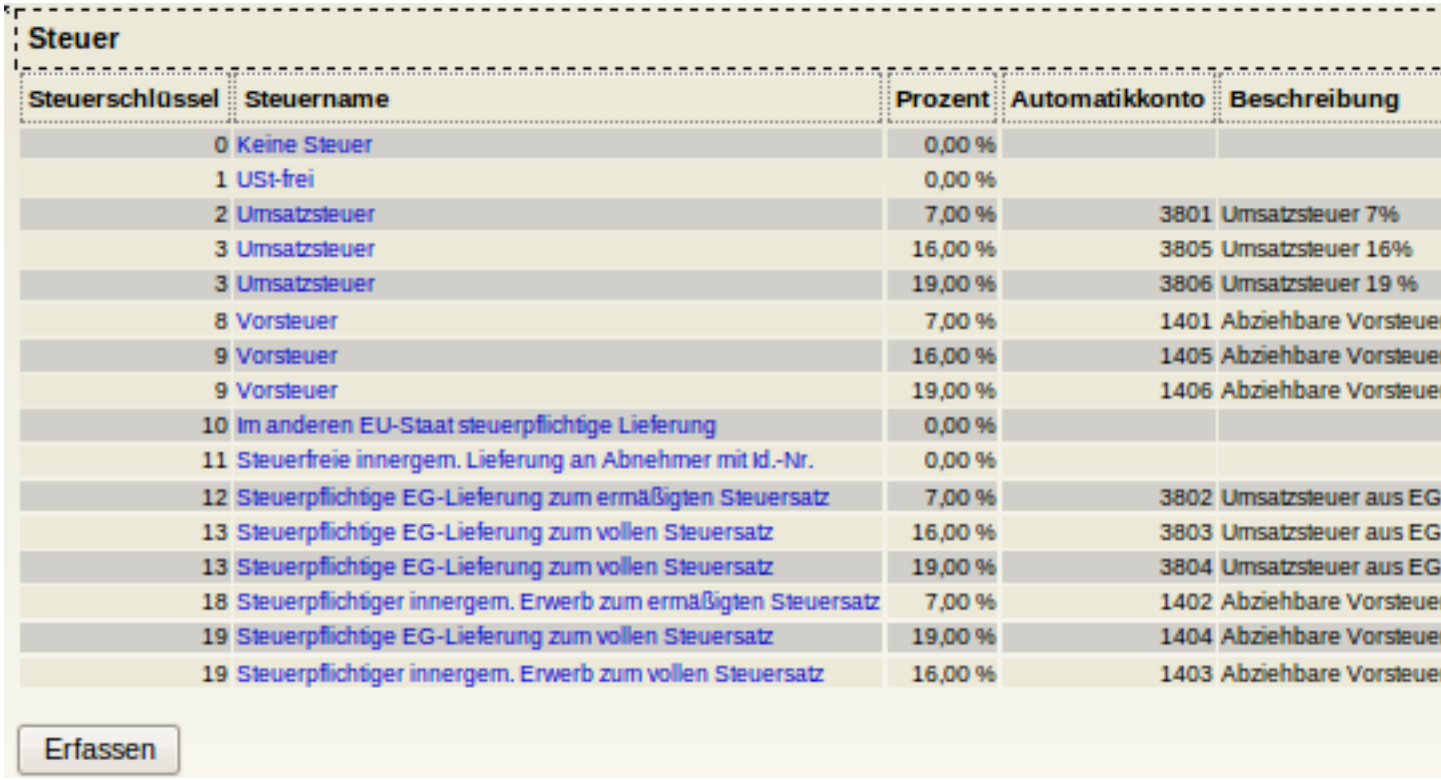
Als Drittes wird ein neuer Eintrag mit Steuerschlüssel 13 für Konto 3804 (19%) angelegt. Dazu unter System -> Steuern -> Erfassen auswählen und die Werte aus dem Screenshot übernehmen.



Als Nächstes sind alle Konten anzupassen, die als Steuerautomatikkonto die 3803 haben, sodass sie ab dem 1.1.2007 auch Steuerautomatik auf 3804 bekommen. Dies betrifft in der Standardkonfiguration die Konten 4315 und 4726. Als Beispiel für 4315 müssen Sie dazu unter System -> Kontenübersicht -> Konten anzeigen das Konto 4315 anklicken und die Einstellungen wie im Screenshot gezeigt vornehmen.


Kontodaten bearbeiten
Grundeinstellungen
Kontonummer
4315
Beschreibung
Erlös Inland stpfl.EG-Lieferung 16%/19%
Kontentyp
Konto
Kontoart
Erlöskonto (I)
Buchungskonto in
Verkauf
Einkauf
Inventar
In Aufklappmenü aufnehmen
Dieser Block ist nur dann gültig, wenn das Konto KEIN Buchungskonto ist, und wenn ein gültiger Steuerschlüsse
Forderungen
Erlöskonto
Zahlungseingang
Steuer
Verbindlichkeiten
Aufwand/Anlagen
Zahlungsausgang
Steuer
Steuerautomatik und UStVA
Achtung: Steuerschlüssel brauchen ein gültiges "Gültig ab"-Datum und werden andernfalls ignoriert.
Steuerschlüssel
13. Steuerpflichtige EG-Lieferung zum vollen Steuersatz (19%) Automatikbuchung auf Konto: 3
13. Steuerpflichtige EG-Lieferung zum vollen Steuersatz (16%) Automatikbuchung auf Konto: 3
00. Keine Steuer (0%) Automatikbuchung auf Konto:
Sonstige Einstellungen
Einnahmen-/Überschussrechnung
01. Umsatzerlöse
BWA
01. Umsatzerlöse
Datevexport
Folgekonto
Kein
Gültig a

Als Letztes sollte die Steuerliste unter System -> Steuern -> Bearbeiten kontrolliert werden. Zum Vergleich der Screenshot.



Steuer

Steuerschlüssel	Steuername	Prozent	Automatikkonto	Beschreibung
0	Keine Steuer	0,00 %		
1	USt-frei	0,00 %		
2	Umsatzsteuer	7,00 %	3801	Umsatzsteuer 7%
3	Umsatzsteuer	16,00 %	3805	Umsatzsteuer 16%
3	Umsatzsteuer	19,00 %	3806	Umsatzsteuer 19 %
8	Vorsteuer	7,00 %	1401	Abziehbare Vorsteue
9	Vorsteuer	16,00 %	1405	Abziehbare Vorsteue
9	Vorsteuer	19,00 %	1406	Abziehbare Vorsteue
10	Im anderen EU-Staat steuerpflichtige Lieferung	0,00 %		
11	Steuerfreie innergem. Lieferung an Abnehmer mit Id.-Nr.	0,00 %		
12	Steuerpflichtige EG-Lieferung zum ermäßigten Steuersatz	7,00 %	3802	Umsatzsteuer aus EG
13	Steuerpflichtige EG-Lieferung zum vollen Steuersatz	16,00 %	3803	Umsatzsteuer aus EG
13	Steuerpflichtige EG-Lieferung zum vollen Steuersatz	19,00 %	3804	Umsatzsteuer aus EG
18	Steuerpflichtiger innergem. Erwerb zum ermäßigten Steuersatz	7,00 %	1402	Abziehbare Vorsteue
19	Steuerpflichtige EG-Lieferung zum vollen Steuersatz	19,00 %	1404	Abziehbare Vorsteue
19	Steuerpflichtiger innergem. Erwerb zum vollen Steuersatz	16,00 %	1403	Abziehbare Vorsteue

Erfassen

2.13. kivitendo ERP verwenden

Nach erfolgreicher Installation ist der Loginbildschirm unter folgender URL erreichbar:

http://localhost/kivitendo-erp/login.pl

Die Administrationsseite erreichen Sie unter:

http://localhost/kivitendo-erp/admin.pl

3

Features und Funktionen

3.1. Wiederkehrende Rechnungen

3.1.1. Einführung

Wiederkehrende Rechnungen werden als normale Aufträge definiert und konfiguriert, mit allen dazugehörigen Kunden- und Artikelangaben. Die konfigurierten Aufträge werden später automatisch in Rechnungen umgewandelt, so als ob man den Workflow benutzen würde, und auch die Auftragsnummer wird übernommen, sodass alle wiederkehrenden Rechnungen, die aus einem Auftrag erstellt wurden, später leicht wiederzufinden sind.

3.1.2. Konfiguration

Um einen Auftrag für wiederkehrende Rechnung zu konfigurieren, findet sich beim Bearbeiten des Auftrags ein neuer Knopf "Konfigurieren", der ein neues Fenster öffnet, in dem man die nötigen Parameter einstellen kann. Hinter dem Knopf wird außerdem noch angezeigt, ob der Auftrag als wiederkehrende Rechnung konfiguriert ist oder nicht.

Folgende Parameter kann man konfigurieren:

Status
: Bei aktiven Rechnungen wird automatisch eine Rechnung erstellt, wenn die Periodizität erreicht ist (z.B. Anfang eines neuen Monats).

  Ist ein Auftrag nicht aktiv, so werden für ihn auch keine wiederkehrenden Rechnungen erzeugt. Stellt man nach längerer nicht-aktiver Zeit einen Auftrag wieder auf aktiv, wird beim nächsten Periodenwechsel für alle Perioden, seit der letzten aktiven Periode, jeweils eine Rechnung erstellt. Möchte man dies verhindern, muss man vorher das Startdatum neu setzen.

  Für gekündigte Aufträge werden nie mehr Rechnungen erstellt. Man kann sich diese Aufträge aber gesondert in den Berichten anzeigen lassen.

Periodizität
: Ob monatlich, quartalsweise oder jährlich auf neue Rechnungen überprüft werden soll. Für jede Periode seit dem Startdatum wird überprüft, ob für die Periode (beginnend immer mit dem ersten Tag der Periode) schon eine Rechnung erstellt wurde. Unter Umständen können bei einem Startdatum in der Vergangenheit gleich mehrere Rechnungen erstellt werden.

Buchen auf
: Das Forderungskonto, in der Regel "Forderungen aus Lieferungen und Leistungen". Das Gegenkonto ergibt sich aus den Buchungsgruppen der betreffenden Waren.

Startdatum
: ab welchem Datum auf Rechnungserstellung geprüft werden soll

Enddatum
: ab wann keine Rechnungen mehr erstellt werden sollen

Automatische Verlängerung um x Monate
: Sollen die wiederkehrenden Rechnungen bei Erreichen des eingetragenen Enddatums weiterhin erstellt werden, so kann man hier die Anzahl der Monate eingeben, um die das Enddatum automatisch nach hinten geschoben wird.

Drucken
: Sind Drucker konfiguriert, so kann man sich die erstellten Rechnungen auch gleich ausdrucken lassen.

Nach Erstellung der Rechnungen kann eine E-Mail mit Informationen zu den erstellten Rechnungen verschickt werden. Konfiguriert wird dies in der Konfigurationsdatei `config/kivitendo.conf` im Abschnitt `[periodic_invoices]`.

3.1.3. Auflisten

Unter Verkauf->Berichte->Aufträge finden sich zwei neue Checkboxen, "Wiederkehrende Rechnungen aktiv" und "Wiederkehrende Rechnungen inaktiv", mit denen man sich einen Überglick über die wiederkehrenden Rechnungen verschaffen kann.

3.1.4. Erzeugung der eigentlichen Rechnungen

Die zeitliche und periodische Überprüfung, ob eine wiederkehrende Rechnung automatisch erstellt werden soll, geschieht durch den Taskserver, einen externen Dienst, der automatisch beim Start des Servers gestartet werden sollte.

3.1.5. Erste Rechnung für aktuellen Monat erstellen

Will man im laufenden Monat eine monatlich wiederkehrende Rechnung inkl. des laufenden Monats starten, stellt man das Startdatum auf den Monatsanfang und wartet ein paar Minuten, bis der Taskserver den neu konfigurieren Auftrag erkennt und daraus eine Rechnung generiert hat. Alternativ setzt man das Startdatum auf den Monatsersten des Folgemonats und erstellt die erste Rechnung direkt manuell über den Workflow.

3.2. Dokumentenvorlagen und verfügbare Variablen

3.2.1. Einführung

Dies ist eine Auflistung der Standard-Dokumentenvorlagen und aller zur Bearbeitung verfügbaren Variablen. Eine Variable wird in einer Vorlage durch ihren Inhalt ersetzt, wenn sie in der Form `<%variablenname%>` verwendet wird. Für LaTeX- und HTML-Vorlagen kann man die Form dieser Tags auch verändern (siehe Anfang und Ende der Tags verändern [24]).

Früher wurde hier nur über LaTeX gesprochen. Inzwischen unterstützt kivitendo aber auch OpenDocument-Vorlagen. Sofern es nicht ausdrücklich eingeschränkt wird, gilt das im Folgenden gesagte für alle Vorlagenarten.

Insgesamt sind technisch gesehen eine ganze Menge mehr Variablen verfügbar als hier aufgelistet werden. Die meisten davon können allerdings innerhalb einer solchen Vorlage nicht sinnvoll verwendet werden. Wenn eine Auflistung dieser Variablen gewollt ist, so kann diese wie folgt erhalten werden:

- `SL/Form.pm` öffnen und am Anfang die Zeile "**use Data::Dumper;**" einfügen.
- In `Form.pm` die Funktion `parse_template` suchen und hier die Zeile **print(STDERR Dumper($self));** einfügen.
- Einmal per Browser die gewünschte Vorlage "benutzen", z.B. ein PDF für eine Rechnung erzeugen.
- Im `error.log` Apache steht die Ausgabe der Variablen `$self` in der Form `'key' => 'value',`. Alle `key`s sind verfügbar.

3.2.2. Variablen ausgeben

Um eine Variable auszugeben, müssen sie einfach nur zwischen die Tags geschrieben werden, also z.B. `<%variablenname%>`.

Optional kann man auch mit Leerzeichen getrennte Flags angeben, die man aber nur selten brauchen wird. Die Syntax sieht also so aus: `<%variablenname FLAG1 FLAG2%>`. Momentan werden die folgenden Flags unterstützt:

- `NOFORMAT` gilt nur für Zahlenwerte und gibt den Wert ohne Formatierung, also ohne Tausendertrennzeichen mit mit einem Punkt als Dezimaltrennzeichen aus. Nützlich z.B., wenn damit in der Vorlage z.B. von LaTeX gerechnet werden soll.
- `NOESCAPE` unterdrückt das Escapen von Sonderzeichen für die Vorlagensprache. Wenn also in einer Variablen bereits gültiger LaTeX-Code steht und dieser von LaTeX auch ausgewertet und nicht wortwörtlich angezeigt werden soll, so ist dieses Flag sinnvoll.

Beispiel:

```
<%quototal NOFORMAT%>
```

3.2.3. Verwendung in Druckbefehlen

In der Admininstration können Drucker definiert werden. Auch im dort eingebbaren Druckbefehl können die hier aufgelisteten Variablen und Kontrollstrukturen verwendet werden. Ihr Inhalt wird dabei nach den Regeln der gängigen Shells formatiert, sodass Sonderzeichen wie `` `...` `` nicht zu unerwünschtem Verhalten führen.

Dies erlaubt z.B. die Definition eines Faxes als Druckerbefehl, für das die Telefonnummer eines Ansprechpartners als Teil der Kommandozeile verwendet wird. Für ein fiktives Kommando könnte das z.B. wie folgt aussehen:

```
send_fax --number <%if cp_phone2%><%cp_phone2%><%else%><%cp_phone1%><%end%>
```

3.2.4. Anfang und Ende der Tags verändern

Der Standardstil für Tags sieht vor, dass ein Tag mit dem Kleinerzeichen und einem Prozentzeichen beginnt und mit dem Prozentzeichen und dem Größerzeichen endet, beispielsweise `<%customer%>`. Da diese Form aber z.B. in LaTeX zu Problemen führen kann, weil das Prozentzeichen dort Kommentare einleitet, kann pro HTML- oder LaTeX-Dokumentenvorlage der Stil umgestellt werden.

Dazu werden in die Datei Zeilen geschrieben, die mit dem für das Format gültigen Kommentarzeichen anfangen, dann `config:` enthalten, die entsprechende Option setzen und bei HTML-Dokumentenvorlagen mit dem Kommentarendzeichen enden. Beispiel für LaTeX:

```
% config: tag-style=($ $)
```

Dies würde kivitendo dazu veranlassen, Variablen zu ersetzen, wenn sie wie folgt aussehen: `($customer$)`. Das äquivalente Beispiel für HTML-Dokumentenvorlagen sieht so aus:

```
<!-- config: tag-style=($ $) -->
```

3.2.5. Zuordnung von den Dateinamen zu den Funktionen

Diese folgende kurze Auflistung zeigt, welche Vorlage bei welcher Funktion ausgelesen wird. Dabei ist die Dateiendung `".ext"` geeignet zu ersetzen: `".tex"` für LaTeX-Vorlagen und `".odt"` für OpenDocument-Vorlagen.

`bin_list.ext`
: Lagerliste

`check.ext`
: ?

`invoice.ext`
: Rechnung

`packing_list.ext`
: Packliste

`pick_list.ext`
: Sammelliste

`purchase_delivery_order.ext`
: Lieferschein (Einkauf)

`purcharse_order.ext`
: Bestellung an Lieferanten

`request_quotation.ext`
: Anfrage an Lieferanten

`sales_delivery_order.ext`
: Lieferschein (Verkauf)

`sales_order.ext`
: Bestellung

`sales_quotation.ext`
: Angebot an Kunden

`zahlungserinnerung.ext`
: Mahnung (Dateiname im Programm konfigurierbar)

`zahlungserinnerung_invoice.ext`
: Rechnung über Mahngebühren (Dateiname im Programm konfigurierbar)

3.2.6. Sprache, Drucker und E-Mail

Angeforderte Sprache und Druckerkürzel in den Dateinamen mit eingearbeitet. So wird aus der Vorlage `sales_order.ext` bei Sprache `de` und Druckerkürzel `lpr2` der Vorlagenname `sales_order_de_lpr2.ext`. Zusätzlich können für E-Mails andere Vorlagen erstellt werden, diese bekommen dann noch das Kürzel `_email`, der vollständige Vorlagenname wäre dann `sales_order_email_de_lpr2.ext`. In allen Fällen kann eine Standarddatei `default.ext` hinterlegt werden. Diese wird verwendet, wenn keine der anderen Varianten gefunden wird.

Die vollständige Suchreihenfolge für einen Verkaufsauftrag mit der Sprache "de" und dem Drucker "lpr2", der per E-Mail im Format PDF verschickt wird, ist:

1. `sales_order_email_de_lpr2.tex`
2. `sales_order_de_lpr2.tex`
3. `sales_order.tex`
4. `default.tex`

Die kurzen Varianten dieser Vorlagentitel müssen dann entweder Standardwerte anzeigen, oder die angeforderten Werte selbst auswerten, siehe dazu Metainformationen zur angeforderten Vorlage [25].

3.2.7. Allgemeine Variablen, die in allen Vorlagen vorhanden sind

3.2.7.1. Metainformationen zur angeforderten Vorlage

Diese Variablen liefern Informationen darüber welche Variante einer Vorlage der Benutzer angefragt hat. Sie sind nützlich für Vorlagenautoren, die aus einer zentralen Layoutvorlage die einzelnen Formulare einbinden möchten.

`template_meta.formname`
: Basisname der Vorlage. Identisch mit der Zurordnung zu den Dateinamen ohne die Erweiterung. Ein Verkaufsauftrag enthält hier `sales_order`.

`template_meta.language.description`
Beschreibung der verwendeten Sprache

`template_meta.language.template_code`
Vorlagenürzel der verwendeten Sprache, identisch mit dem Kürzel das im Dateinamen verwendetet wird.

`template_meta.language.output_numberformat`
Zahlenformat der verwendeten Sprache in der Form "`1.000,00`". Experimentell! Nur interessant für Vorlagen die mit unformatierten Werten arbeiten.

`template_meta.language.output_dateformat`
Datumsformat der verwendeten Sprache in der Form "`dd.mm.yyyy`". Experimentell! Nur interessant für Vorlagen die mit unformatierten Werten arbeiten.

`template_meta.format`
Das angeforderte Format. Kann im Moment die Werte `pdf`, `postscript`, `html`, `opendocument`, `opendocument_pdf` und `excel` enthalten.

`template_meta.extension`
Dateierweiterung, wie im Dateinamen. Wird aus `format` entschieden.

`template_meta.media`
Ausgabemedium. Kann zur Zeit die Werte `screen` für Bildschirm, `email` für E-Mmail (triggert das `_email` Kürzel im Dateinamen), `printer` für Drucker, und `queue` für Warteschlange enthalten.

`template_meta.printer.description`
Beschreibung des ausgewählten Druckers

`template_meta.printer.template_code`
Vorlagenürzel des ausgewählten Druckers, identisch mit dem Kürzel das im Dateinamen verwendetet wird.

`template_meta.tmpfile`
Datei-Prefix für temporäre Dateien.

3.2.7.2. Stammdaten von Kunden und Lieferanten

`account_number`
Kontonummer

`bank`
Name der Bank

`bank_code`
Bankleitzahl

`bic`
Bank-Identifikations-Code (Bank Identifier Code, BIC)

`business`
Kunden-/Lieferantentyp

`city`
Stadt

`contact`
Kontakt

`country`
Land

`cp_email`
Email des Ansprechpartners

`cp_givenname`
: Vorname des Ansprechpartners

`cp_greeting`
: Anrede des Ansprechpartners

`cp_name`
: Name des Ansprechpartners

`cp_phone1`
: Telefonnummer 1 des Ansprechpartners

`cp_phone2`
: Telefonnummer 2 des Ansprechpartners

`cp_title`
: Titel des Ansprechpartners

`creditlimit`
: Kreditlimit

`customeremail`
: Email des Kunden; nur für Kunden

`customerfax`
: Faxnummer des Kunden; nur für Kunden

`customernotes`
: Bemerkungen beim Kunden; nur für Kunden

`customernumber`
: Kundennummer; nur für Kunden

`customerphone`
: Telefonnummer des Kunden; nur für Kunden

`discount`
: Rabatt

`email`
: Emailadresse

`fax`
: Faxnummer

`homepage`
: Homepage

`iban`
: Internationale Kontonummer (International Bank Account Number, IBAN)

`language`
: Sprache

`name`
: Firmenname

`payment_description`
: Name der Zahlart

`payment_terms`
: Zahlungskonditionen

`phone`
: Telefonnummer

`shiptocity`
: Stadt (Lieferadresse) *

`shiptocontact`
: Kontakt (Lieferadresse) *

`shiptocountry`
: Land (Lieferadresse) *

`shiptodepartment1`
: Abteilung 1 (Lieferadresse) *

`shiptodepartment2`
: Abteilung 2 (Lieferadresse) *

`shiptoemail`
: Email (Lieferadresse) *

`shiptofax`
: Fax (Lieferadresse) *

`shiptoname`
: Firmenname (Lieferadresse) *

`shiptophone`
: Telefonnummer (Lieferadresse) *

`shiptostreet`
: Straße und Hausnummer (Lieferadresse) *

`shiptozipcode`
: Postleitzahl (Lieferadresse) *

`street`
: Straße und Hausnummer

`taxnumber`
: Steuernummer

`ustid`
: Umsatzsteuer-Identifikationsnummer

`vendoremail`
: Email des Lieferanten; nur für Lieferanten

`vendorfax`
: Faxnummer des Lieferanten; nur für Lieferanten

`vendornotes`
: Bemerkungen beim Lieferanten; nur für Lieferanten

`vendornumber`
: Lieferantennummer; nur für Lieferanten

`vendorphone`
: Telefonnummer des Lieferanten; nur für Lieferanten

`zipcode`
: Postleitzahl

Anmerkung

Anmerkung: Sind die `shipto*`-Felder in den Stammdaten nicht eingetragen, so haben die Variablen `shipto*` den gleichen Wert wie die die entsprechenden Variablen der Lieferdaten. Das bedeutet, dass sich einige `shipto*`-Variablen so nicht in den Stammdaten wiederfinden sondern schlicht Kopien der Lieferdatenvariablen sind (z.B. `shiptocontact`).

3.2.7.3. Informationen über den Bearbeiter

`employee_address`
: Adressfeld

`employee_businessnumber`
: Firmennummer

`employee_company`
: Firmenname

`employee_co_ustid`
: Usatzsteuer-Identifikationsnummer

`employee_duns`
: DUNS-Nummer

`employee_email`
: Email

`employee_fax`
: Fax

`employee_name`
: voller Name

`employee_signature`
: Signatur

`employee_taxnumber`
: Steuernummer

`employee_tel`
: Telefonnummer

3.2.7.4. Informationen über den Bearbeiter

`salesman_address`
: Adressfeld

`salesman_businessnumber`
: Firmennummer

`salesman_company`
: Firmenname

`salesman_co_ustid`
: Usatzsteuer-Identifikationsnummer

`salesman_duns`
: DUNS-Nummer

`salesman_email`
: Email

`salesman_fax`
Fax

`salesman_name`
voller Name

`salesman_signature`
Signatur

`salesman_taxnumber`
Steuernummer

`salesman_tel`
Telefonnummer

3.2.7.5. Variablen für die einzelnen Steuern

`tax`
Steuer

`taxbase`
zu versteuernder Betrag

`taxdescription`
Name der Steuer

`taxrate`
Steuersatz

3.2.8. Variablen in Rechnungen

3.2.8.1. Allgemeine Variablen

`creditremaining`
Verbleibender Kredit

`currency`
Währung

`cusordnumber`
Bestellnummer beim Kunden

`deliverydate`
Lieferdatum

`duedate`
Fälligkeitsdatum

`globalprojectnumber`
Projektnummer des ganzen Beleges

`globalprojectdescription`
Projekbeschreibung des ganzen Beleges

`intnotes`
Interne Bemerkungen

`invdate`
Rechnungsdatum

`invnumber`
: Rechnungsnummer

`invtotal`
: gesamter Rechnungsbetrag

`notes`
: Bemerkungen der Rechnung

`orddate`
: Auftragsdatum

`ordnumber`
: Auftragsnummer, wenn die Rechnung aus einem Auftrag erstellt wurde

`payment_description`
: Name der Zahlart

`payment_terms`
: Zahlungskonditionen

`quodate`
: Angebotsdatum

`quonumber`
: Angebotsnummer

`shippingpoint`
: Versandort

`shipvia`
: Transportmittel

`subtotal`
: Zwischensumme aller Posten ohne Steuern

`total`
: Restsumme der Rechnung (Summe abzüglich bereits bezahlter Posten)

`transaction_description`
: Vorgangsbezeichnung

`transdate`
: Auftragsdatum wenn die Rechnung aus einem Auftrag erstellt wurde

3.2.8.2. Variablen für jeden Posten auf der Rechnung

`bin`
: Stellage

`description`
: Artikelbeschreibung

`discount`
: Rabatt als Betrag

`discount_sub`
: Zwischensumme mit Rabatt

`drawing`
: Zeichnung

`ean`
: EAN-Code

`image`
: Grafik

`linetotal`
: Zeilensumme (Anzahl * Einzelpreis)

`longdescription`
: Langtext

`microfiche`
: Mikrofilm

`netprice`
: Nettopreis

`nodiscount_linetotal`
: Zeilensumme ohne Rabatt

`nodiscount_sub`
: Zwischensumme ohne Rabatt

`number`
: Artikelnummer

`ordnumber_oe`
: Auftragsnummer des Originalauftrags, wenn die Rechnung aus einem Sammelauftrag erstellt wurde

`p_discount`
: Rabatt in Prozent

`partnotes`
: Die beim Artikel gespeicherten Bemerkungen

`partsgroup`
: Warengruppe

`price_factor`
: Der Preisfaktor als Zahl, sofern einer eingestellt ist

`price_factor_name`
: Der Name des Preisfaktors, sofern einer eingestellt ist

`projectnumber`
: Projektnummer

`projectdescription`
: Projektbeschreibung

`qty`
: Anzahl

`reqdate`
: Lieferdatum

`runningnumber`
: Position auf der Rechnung (1, 2, 3...)

`sellprice`
: Verkaufspreis

`serialnumber`
Seriennummer

`tax_rate`
Steuersatz

`transdate_oe`
Auftragsdatum des Originalauftrags, wenn die Rechnung aus einem Sammelauftrag erstellt wurde

`unit`
Einheit

`weight`
Gewicht

Für jeden Posten gibt es ein Unterarray mit den Informationen über Lieferanten und Lieferantenartikelnummer. Diese müssen mit einer `foreach`-Schleife ausgegeben werden, da für jeden Artikel mehrere Lieferanteninformationen hinterlegt sein können. Die Variablen dafür lauten:

`make`
Lieferant

`model`
Lieferantenartikelnummer

3.2.8.3. Variablen für die einzelnen Zahlungseingänge

`payment`
Betrag

`paymentaccount`
Konto

`paymentdate`
Datum

`paymentmemo`
Memo

`paymentsource`
Beleg

3.2.8.4. Benutzerdefinierte Kunden- und Lieferantenvariablen

Die vom Benutzer definierten Variablen für Kunden und Lieferanten stehen beim Ausdruck von Einkaufs- und Verkaufsbelegen ebenfalls zur Verfügung. Ihre Namen setzen sich aus dem Präfix `vc_cvar_` und dem vom Benutzer festgelegten Variablennamen zusammen.

Beispiel: Der Benutzer hat eine Variable namens `number_of_employees` definiert, die die Anzahl der Mitarbeiter des Unternehmens enthält. Diese Variable steht dann unter dem Namen `vc_cvar_number_of_employees` zur Verfügung.

3.2.9. Variablen in Mahnungen und Rechnungen über Mahngebühren

3.2.9.1. Namen der Vorlagen

Die Namen der Vorlagen werden im System-Menü vom Benutzer eingegeben. Wird für ein Mahnlevel die Option zur automatischen Erstellung einer Rechnung über die Mahngebühren und Zinsen aktiviert, so wird der Name der Vorlage für diese Rechnung aus dem Vorlagenname für diese Mahnstufe mit dem Zusatz `_invoice` gebildet. Weiterhin werden die Kürzel für die ausgewählte Sprache und den ausgewählten Drucker angehängt.

3.2.9.2. Allgemeine Variablen in Mahnungen

Die Variablen des Verkäufers stehen wie gewohnt als `employee_...` zur Verfügung. Die Adressdaten des Kunden stehen als Variablen `name`, `street`, `zipcode`, `city`, `country`, `department_1`, `department_2`, und `email` zur Verfügung.

Weitere Variablen beinhalten:

`dunning_date`
: Datum der Mahnung

`dunning_duedate`
: Fälligkeitsdatum für diese Mahhnung

`dunning_id`
: Mahnungsnummer

`fee`
: Kummulative Mahngebühren

`interest_rate`
: Zinssatz per anno in Prozent

`total_amount`
: Gesamter noch zu zahlender Betrag als `fee` + `total_interest` + `total_open_amount`

`total_interest`
: Zinsen per anno über alle Rechnungen

`total_open_amount`
: Summe über alle offene Beträge der Rechnungen

3.2.9.3. Variablen für jede gemahnte Rechnung in einer Mahnung

`dn_amount`
: Rechnungssumme (brutto)

`dn_duedate`
: Originales Fälligkeitsdatum der Rechnung

`dn_dunning_date`
: Datum der Mahnung

`dn_dunning_duedate`
: Fälligkeitsdatum der Mahnung

`dn_fee`
: Kummulative Mahngebühr

`dn_interest`
: Zinsen per anno für diese Rechnung

`dn_invnumber`
: Rechnungsnummer

`dn_linetotal`
: Noch zu zahlender Betrag (ergibt sich aus `dn_open_amount` + `dn_fee` + `dn_interest`)

`dn_netamount`
: Rechnungssumme (netto)

`dn_open_amount`
: Offener Rechnungsbetrag

`dn_ordnumber`
Bestellnummer

`dn_transdate`
Rechnungsdatum

`dn_curr`
Währung, in der die Rechnung erstellt wurde. (Die Rechnungsbeträge sind aber immer in der Hauptwährung)

3.2.9.4. Variablen in automatisch erzeugten Rechnungen über Mahngebühren

Die Variablen des Verkäufers stehen wie gewohnt als `employee_...` zur Verfügung. Die Adressdaten des Kunden stehen als Variablen `name`, `street`, `zipcode`, `city`, `country`, `department_1`, `department_2`, und `email` zur Verfügung.

Weitere Variablen beinhalten:

`duedate`
Fälligkeitsdatum der Rechnung

`dunning_id`
Mahnungsnummer

`fee`
Mahngebühren

`interest`
Zinsen

`invamount`
Rechnungssumme (ergibt sich aus `fee` + `interest`)

`invdate`
Rechnungsdatum

`invnumber`
Rechnungsnummer

3.2.10. Variablen in anderen Vorlagen

3.2.10.1. Einführung

Die Variablen in anderen Vorlagen sind ähnlich wie in der Rechnung. Allerdings heißen die Variablen, die mit `inv` beginnen, jetzt anders. Bei den Angeboten fangen sie mit `quo` für "quotation" an: `quodate` für Angebotsdatum etc. Bei Bestellungen wiederum fangen sie mit `ord` für "order" an: `ordnumber` für Bestellnummer etc.

Manche Variablen sind in anderen Vorlagen hingegen gar nicht vorhanden wie z.B. die für bereits verbuchte Zahlungseingänge. Dies sind Variablen, die vom Geschäftsablauf her in der entsprechenden Vorlage keine Bedeutung haben oder noch nicht belegt sein können.

Im Folgenden werden nur wichtige Unterschiede zu den Variablen in Rechnungen aufgeführt.

3.2.10.2. Angebote und Preisanfragen

`quonumber`
Angebots- bzw. Anfragenummer

`reqdate`
Gültigkeitsdatum (bei Angeboten) bzw. Lieferdatum (bei Preisanfragen)

`transdate`
Angebots- bzw. Anfragedatum

3.2.10.3. Auftragsbestätigungen und Lieferantenaufträge

`ordnumber`
Auftragsnummer

`reqdate`
Lieferdatum

`transdate`
Auftragsdatum

3.2.10.4. Lieferscheine (Verkauf und Einkauf)

`cusordnumber`
Bestellnummer des Kunden (im Verkauf) bzw. Bestellnummer des Lieferanten (im Einkauf)

`donumber`
Lieferscheinnummer

`transdate`
Lieferscheindatum

Für jede Position eines Lieferscheines gibt es ein Unterarray mit den Informationen darüber, von welchem Lager und Lagerplatz aus die Waren verschickt wurden (Verkaufslieferscheine) bzw. auf welchen Lagerplatz sie eingelagert wurden. Diese müssen mittels einer `foreach`-Schleife ausgegeben werden. Diese Variablen sind:

`si_bin`
Lagerplatz

`si_chargenumber`
Chargennummer

`si_bestbefore`
Mindesthaltbarkeit

`si_number`
Artikelnummer

`si_qty`
Anzahl bzw. Menge

`si_runningnumber`
Positionsnummer (1, 2, 3 etc)

`si_unit`
Einheit

`si_warehouse`
Lager

3.2.10.5. Variablen für Sammelrechnung

`c0total`
Gesamtbetrag aller Rechnungen mit Fälligkeit < 30 Tage

`c30total`
Gesamtbetrag aller Rechnungen mit Fälligkeit >= 30 und < 60 Tage

`c60total`
Gesamtbetrag aller Rechnungen mit Fälligkeit >= 60 und < 90 Tage

`c90total`
Gesamtbetrag aller Rechnungen mit Fälligkeit >= 90 Tage

`total`
Gesamtbetrag aller Rechnungen

Variablen für jede Rechnungsposition in Sammelrechnung:

`invnumber`
Rechnungsnummer

`invdate`
Rechnungsdatum

`duedate`
Fälligkeitsdatum

`amount`
Summe der Rechnung

`open`
Noch offener Betrag der Rechnung

`c0`
Noch offener Rechnungsbetrag mit Fälligkeit < 30 Tage

`c30`
Noch offener Rechnungsbetrag mit Fälligkeit >= 30 und < 60 Tage

`c60`
Noch offener Rechnungsbetrag mit Fälligkeit >= 60 und < 90 Tage

`c90`
Noch offener Rechnungsbetrag mit Fälligkeit >= 90 Tage

3.2.11. Blöcke, bedingte Anweisungen und Schleifen

3.2.11.1. Einfürhung

Der Parser kennt neben den Variablen einige weitere Konstrukte, die gesondert behandelt werden. Diese sind wie Variablennamen in spezieller Weise markiert: **<%anweisung%> ... <%end%>**

Anmerkung zum **<%end%>**: Der besseren Verständlichkeit halber kann man nach dem **end** noch beliebig weitere Wörter schreiben, um so zu markieren, welche Anweisung (z.B. **if** oder **foreach**) damit abgeschlossen wird.

Beispiel: Lautet der Beginn eines Blockes z.B. **<%if type == "sales_quotation"%>**, so könnte er mit **<%end%>** genauso abgeschlossen werden wie mit **<%end if%>** oder auch **<%end type == "sales_quotation"%>**.

3.2.11.2. Der if-Block

```
<%if variablenname%>
...
<%end%>
```

Eine normale "if-then"-Bedingung. Die Zeilen zwischen dem "if" und dem "end" werden nur ausgegeben, wenn die Variable `variablenname` gesetzt und ungleich 0 ist.

Die Bedingung kann auch negiert werden, indem das Wort `not` nach dem `if` verwendet wird. Beispiel:

```
<%if not cp_greeting%>
...
<%end%>
```

Zusätzlich zu dem einfachen Test, ob eine Variable gesetzt ist oder nicht, bietet dieser Block auch die Möglichkeit, den Inhalt einer Variablen mit einer festen Zeichenkette oder einer anderen Variablen zu vergleichen. Ob der Vergleich mit einer Zeichenkette oder einer anderen Variablen vorgenommen wird, hängt davon ab, ob die rechte Seite des Vergleichsoperators in Anführungszeichen gesetzt wird (Vergleich mit Zeichenkette) oder nicht (Vergleich mit anderer Variablen). Zwei Beispiele, die beide Vergleiche zeigen:

```
<%if var1 == "Wert"%>
```

Testet die Variable `var1` auf übereinstimmung mit der Zeichenkette `Wert`. Mittels `!=` anstelle von `==` würde auf Ungleichheit getestet.

```
<%if var1 == var2%>
```

Testet die Variable `var1` auf übereinstimmung mit der Variablen `var2`. Mittel `!=` anstelle von `==` würde auf Ungleichheit getestet.

Erfahrere Benutzer können neben der Tests auf (Un-)Gleichheit auch Tests auf übereinstimmung mit regulären Ausdrücken ohne Berücksichtigung der Groß- und Kleinschreibung durchführen. Dazu dient dieselbe Syntax wie oben nur mit `=~` und `!~` als Vergleichsoperatoren.

Beispiel für einen Test, ob die Variable `intnotes` (interne Bemerkungen) das Wort `schwierig` enthält:

```
<%if intnotes =~ "schwierig"%>
```

3.2.11.3. Der foreach-Block

```
<%foreach variablenname%>
...
<%end%>
```

Fügt die Zeilen zwischen den beiden Anweisungen so oft ein, wie das Perl-Array der Variablen `variablenname` Elemente enthät. Dieses Konstrukt wird zur Ausgabe der einzelnen Posten einer Rechnung / eines Angebots sowie zur Ausgabe der Steuern benutzt. In jedem Durchlauf werden die zeilenbezogenen Variablen jeweils auf den Wert für die aktuelle Position gesetzt.

Die Syntax sieht normalerweise wie folgt aus:

```
<%foreach number%>
Position: <%runningnumber%>
Anzahl: <%qty%>
Artikelnummer: <%number%>
Beschreibung: <%description%>
...
<%end%>
```

Besonderheit in OpenDocument-Vorlagen: Tritt ein `<%foreach%>`-Block innerhalb einer Tabellenzelle auf, so wird die komplette Tabellenzeile so oft wiederholt wie notwendig. Tritt er außerhalb auf, so wird nur der Inhalt zwischen `<%foreach%>` und `<%end%>` wiederholt, nicht aber die komplette Zeile, in der er steht.

3.2.12. Markup-Code zur Textformatierung innerhalb von Formularen

Wenn der Benutzer innhalb von Formularen in kivitendo Text anders formatiert haben möchte, so ist dies begrenzt möglich. kivitendo unterstützt die Textformatierung mit HTML-ähnlichen Tags. Der Benutzer kann z.B. bei der Artikelbeschreibung auf

einer Rechnung Teile des Texts zwischen Start- und Endtags setzen. Dieser Teil wird dann automatisch in Anweisungen für das ausgewählte Vorlagenformat (HTML oder PDF über LaTeX) umgesetzt.

Die unterstützen Formatierungen sind:

Text
: Text wird in Fettdruck gesetzt.

Text
: Text wird kursiv gesetzt.

<u>Text</u>
: Text wird unterstrichen.

<s>Text</s>
: Text wird durchgestrichen. Diese Formatierung ist nicht bei der Ausgabe als PDF über LaTeX verfügbar.

<bullet>
: Erzeugt einen ausgefüllten Kreis für Aufzählungen (siehe unten).

Der Befehl **<bullet>** funktioniert momentan auch nur in Latex-Vorlagen.

3.3. Excel-Vorlagen

3.3.1. Zusammenfassung

Dieses Dokument beschreibt den Mechanismus, mit dem Exceltemplates abgearbeitet werden, und die Einschränkungen, die damit einhergehen.

3.3.2. Bedienung

Der Excel Mechanismus muss in der Konfigurationsdatei aktiviert werden. Die Konfigurationsoption heißt `excel_templates = 1` im Abschnitt `[print_templates]`.

Eine Excelvorlage kann dann unter dem Namen einer beliebigen anderen Vorlage mit der Endung `.xls` gespeichert werden. In den normalen Verkaufsmasken taucht nun `Excel` als auswählbares Format auf und kann von da an wie LaTeX- oder OpenOffice-Vorlagen benutzt werden.

Der Sonderfall der Angebote aus der Kundenmaske ist ebenfalls eine Angebotsvorlage und wird unter dem internen Namen der Angebote `sales_quotation.xls` gespeichert.

3.3.3. Variablensyntax

Einfache Syntax: **<<varname>>**

Dabei sind << und >> die Delimiter. Da Excel auf festen Breiten besteht, kann der Tag künstlich verlängert werden, indem weitere < oder > eingefügt werden. Der Tag muss nicht symmetrisch sein. Beispiel:

```
<<<<<varname>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Um die Limitierung der festen Breite zu reduzieren, können weitere Variablen in einem Block interpoliert werden. Whitespace wird dazwishen dann erhalten. Beispiel:

```
<<<<<varname1 varname2   varname3>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Die Variablen werden interpoliert, und linksbündig mit Leerzeichen auf die gewünschte Länge aufgefüllt. Ist der String zu lang, werden überzählige Zeichen abgeschnitten.

Es ist ausserdem möglich, Daten rechtsbündig darzustellen, wenn der Block mit einem Leerzeichen anfängt. Beispiel:

```
<<<<<<          varname>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Dies würde rechtsbündig triggern. Wenn bei rechtsbündiger Ausrichtung Text abgeschnitten werden muss, wird er vom linken Ende entfernt.

3.3.4. Einschränkungen

Das Excelformat bis 2002 ist ein binäres Format, und kann nicht mit vertretbarem Aufwand editiert werden. Der Templatemechanismus beschränkt sich daher darauf, Textstellen exakt durch einen anderen Text zu ersetzen.

Aus dem gleichen Grund sind die Kontrolllstrukturen **<%if%>** und **<%foreach%>** nicht vorhanden. Der Delimiter **<% %>** kommt in den Headerinformationen evtl. vor. Deshalb wurde auf den sichereren Delimiter **<<** und **>>** gewechselt.

4 Entwicklerdokumentation

4.1. Globale Variablen

4.1.1. Wie sehen globale Variablen in Perl aus?

Globale Variablen liegen in einem speziellen namespace namens "main", der von überall erreichbar ist. Darüber hinaus sind bareword globs global und die meisten speziellen Variablen sind... speziell.

Daraus ergeben sich folgende Formen:

`$main::form`
: expliziter Namespace "main"

`$::form`
: impliziter Namespace "main"

`open FILE, "file.txt"`
: `FILE` ist global

`$_`
: speziell

Im Gegensatz zu PHP™ gibt es kein Schlüsselwort wie "`global`", mit dem man importieren kann. `my`, `our` und `local` machen was anderes.

`my $form`
: lexikalische Variable, gültig bis zum Ende des Scopes

`our $form`
: `$form` referenziert ab hier `$PACKAGE::form`.

`local $form`
: Alle Änderungen an `$form` werden am Ende des scopes zurückgesetzt

4.1.2. Warum sind globale Variablen ein Problem?

Das erste Problem ist FCGI™.

SQL-Ledger™ hat fast alles im globalen namespace abgelegt, und erwartet, dass es da auch wiederzufinden ist. Unter FCGI™ müssen diese Sachen aber wieder aufgeräumt werden, damit sie nicht in den nächsten Request kommen. Einige Sachen wiederum sollen nicht gelöscht werden, wie zum Beispiel Datenbankverbindungen, weil die sehr lange zum Initialisieren brauchen.

Das zweite Problem ist `strict`. Unter `strict` werden alle Variablen die nicht explizit mit `Package`, `my` oder `our` angegeben werden als Tippfehler angemarkert, dies hat, seit der Einführung, u.a. schon so manche langwierige Bug-Suche verkürzt. Da globale Variablen aber implizit mit Package angegeben werden, werden die nicht geprüft, und somit kann sich schnell ein Tippfehler einschleichen.

4.1.3. Kanonische globale Variablen

Um dieses Problem im Griff zu halten gibt es einige wenige globale Variablen, die kanonisch sind, d.h. sie haben bestimmte vorgegebenen Eigenschaften, und alles andere sollte anderweitig umhergereicht werden.

Diese Variablen sind im Moment die folgenden neun:

- `$::form`
- `%::myconfig`
- `$::locale`
- `$::lxdebug`
- `$::auth`
- `$::lx_office_conf`
- `$::instance_conf`
- `$::dispatcher`
- `$::request`

Damit diese nicht erneut als Müllhalde missbraucht werden, im Folgenden eine kurze Erläuterung der bestimmten vorgegebenen Eigenschaften (Konventionen):

4.1.3.1. $::form

- Ist ein Objekt der Klasse "`Form`"
- Wird nach jedem Request gelöscht
- Muss auch in Tests und Konsolenscripts vorhanden sein.
- Enthält am Anfang eines Requests die Requestparameter vom User
- Kann zwar intern über Requestgrenzen ein Datenbankhandle cachen, das wird aber momentan absichtlich zerstört

`$::form` wurde unter SQL Ledger™ als Gottobjekt für alles misbraucht. Sämtliche alten Funktionen unter SL/ mutieren `$::form`, das heißt, alles was einem lieb ist (alle Variablen die einem ans Herz gewachsen sind), sollte man vor einem Aufruf (!) von zum Beispiel `IS->retrieve_customer()` in Sicherheit bringen.

Z.B. das vom Benutzer eingestellte Zahlenformat, bevor man Berechnung in einem bestimmten Format durchführt (SL/Form.pm Zeile 3552, Stand version 2.7beta), um dies hinterher wieder auf den richtigen Wert zu setzen:

```
my $saved_numberformat    = $::myconfig{numberformat};
$::myconfig{numberformat} = $numberformat;
# (...) div Berechnungen
$::myconfig{numberformat} = $saved_numberformat;
```

Das Objekt der Klasse Form hat leider im Moment noch viele zentrale Funktionen die vom internen Zustand abhängen, deshalb bitte nie einfach zerstören oder überschreiben (zumindestens nicht kurz vor einem Release oder in Absprache über bspw. die devel-Liste ;-). Es geht ziemlich sicher etwas kaputt.

`$::form` ist gleichzeitig der Standard Scope in den Template::Toolkit™ Templates außerhalb der Controller: der Ausdruck `[% var %]` greift auf `$::form->{var}` zu. Unter Controllern ist der Standard Scope anders, da lautet der Zugriff `[% FORM.var %]`. In Druckvorlagen sind normale Variablen ebenfall im `$::form` Scope, d.h. `<%var%>` zeigt auf `$::form->{var}`. Nochmal von der anderen Seite erläutert, innerhalb von (Web-)Templates sieht man häufiger solche Konstrukte:

```
[%- IF business %]
# (... Zeig die Auswahlliste Kunden-/Lieferantentyp an)
[%- END %]
```

Entweder wird hier dann $::form->{business} ausgewertet oder aber der Funktion `$form->parse_html_template` wird explizit noch ein zusätzlicher Hash übergeben, der dann auch in den (Web-)Templates zu Verfügung steht, bspw. so:

```
$form->parse_html_template("is/form_header", \%TMPL_VAR);
```

Innerhalb von Schleifen wird `$::form->{TEMPLATE_ARRAYS}{var}[$index]` bevorzugt, wenn vorhanden. Ein Beispiel findet sich in SL/DO.pm, welches über alle Positionen eines Lieferscheins in Schleife läuft:

```
for $i (1 .. $form->{rowcount}) {
  # ...
  push @{ $form->{TEMPLATE_ARRAYS}{runningnumber} },   $position;
  push @{ $form->{TEMPLATE_ARRAYS}{number} },          $form->{"partnumber_$i"};
  push @{ $form->{TEMPLATE_ARRAYS}{description} },     $form->{"description_$i"};
  # ...
}
```

4.1.3.2. %::myconfig

- Das einzige Hash unter den globalen Variablen
- Wird spätestens benötigt wenn auf die Datenbank zugegriffen wird
- Wird bei jedem Request neu erstellt.
- Enthält die Userdaten des aktuellen Logins
- Sollte nicht ohne Filterung irgendwo gedumpt werden oder extern serialisiert werden, weil da auch der Datenbankzugriff für diesen user drinsteht.
- Enthält unter anderem Listenbegrenzung vclimit, Datumsformat dateformat und Nummernformat numberformat
- Enthält Datenbankzugriffinformationen

`%::myconfig` ist im Moment der Ersatz für ein Userobjekt. Die meisten Funktionen, die etwas anhand des aktuellen Users entscheiden müssen, befragen `%::myconfig`. Innerhalb der Anwendungen sind dies überwiegend die Daten, die sich unter Programm -> Einstellungen befinden, bzw. die Informationen über den Benutzer die über die Administrator-Schnittstelle (admin.pl) eingegeben wurden.

4.1.3.3. $::locale

- Objekt der Klasse "Locale"
- Wird pro Request erstellt
- Muss auch für Tests und Scripte immer verfügbar sein.
- Cached intern über Requestgrenzen hinweg benutzte Locales

Lokalisierung für den aktuellen User. Alle Übersetzungen, Zahlen- und Datumsformatierungen laufen über dieses Objekt.

4.1.3.4. $::lxdebug

- Objekt der Klasse "LXDebug"
- Wird global gecached
- Muss immer verfügbar sein, in nahezu allen Funktionen

`$::lxdebug` stellt Debuggingfunktionen bereit, wie "`enter_sub`" und "`leave_sub`", mit denen in den alten Modulen ein brauchbares Tracing gebaut ist, "`log_time`", mit der man die Wallclockzeit seit Requeststart loggen kann, sowie "`message`" und "`dump`" mit denen man flott Informationen ins Log (tmp/kivitendo-debug.log) packen kann.

Beispielsweise so:

```
$main::lxdebug->message(0, 'Meine Konfig:' . Dumper (%::myconfig));
$main::lxdebug->message(0, 'Wer bin ich? Kunde oder Lieferant:' . $form->{vc});
```

4.1.3.5. $::auth

- Objekt der Klasse "SL::Auth"
- Wird global gecached
- Hat eine permanente DB Verbindung zur Authdatenbank
- Wird nach jedem Request resettet.

`$::auth` stellt Funktionen bereit um die Rechte des aktuellen Users abzufragen. Obwohl diese Informationen vom aktuellen User abhängen wird das Objekt aus Geschwindigkeitsgründen nur einmal angelegt und dann nach jedem Request kurz resettet.

4.1.3.6. $::lx_office_conf

- Objekt der Klasse "`SL::LxOfficeConf`"
- Global gecached
- Repräsentation der `config/kivitendo.conf[.default]`-Dateien

Globale Konfiguration. Configdateien werden zum Start gelesen und danach nicht mehr angefasst. Es ist derzeit nicht geplant, dass das Programm die Konfiguration ändern kann oder sollte.

Beispielsweise ist über den Konfigurationseintrag [debug] die Debug- und Trace-Log-Datei wie folgt konfiguriert und verfügbar:

```
[debug]
file = /tmp/kivitendo-debug.log
```

ist der Key `file` im Programm als `$::lx_office_conf->{debug}{file}` erreichbar.

Warnung

Zugriff auf die Konfiguration erfolgt im Moment über Hashkeys, sind also nicht gegen Tippfehler abgesichert.

4.1.3.7. $::instance_conf

- Objekt der Klasse "`SL::InstanceConfiguration`"
- wird pro Request neu erstellt

Funktioniert wie `$::lx_office_conf`, speichert aber Daten die von der Instanz abhängig sind. Eine Instanz ist hier eine Mandantendatenbank. Beispielsweise überprüft

```
$::instance_conf->get_inventory_system eq 'perpetual'
```

ob die berüchtigte Bestandsmethode zur Anwendung kommt.

4.1.3.8. $::dispatcher

- Objekt der Klasse "`SL::Dispatcher`"

- wird pro Serverprozess erstellt.
- enthält Informationen über die technische Verbindung zum Server

Der dritte Punkt ist auch der einzige Grund warum das Objekt global gespeichert wird. Wird vermutlich irgendwann in einem anderen Objekt untergebracht.

4.1.3.9. $::request

- Hashref (evtl später Objekt)
- Wird pro Request neu initialisiert.
- Keine Unterstruktur garantiert.

`$::request` ist ein generischer Platz um Daten "für den aktuellen Request" abzulegen. Sollte nicht für action at a distance benutzt werden, sondern um lokales memoizing zu ermöglichen, das garantiert am Ende des Requests zerstört wird.

Vieles von dem, was im moment in `$::form` liegt, sollte eigentlich hier liegen. Die groben Differentialkriterien sind:

- Kommt es vom User, und soll unverändert wieder an den User? Dann `$::form`, steht da eh schon
- Sind es Daten aus der Datenbank, die nur bis zum Ende des Requests gebraucht werden? Dann `$::request`
- Muss ich von anderen Teilen des Programms lesend drauf zugreifen? Dann `$::request`, aber Zugriff über Wrappermethode

4.1.4. Ehemalige globale Variablen

Die folgenden Variablen waren einmal im Programm, und wurden entfernt.

4.1.4.1. $::cgi

- war nötig, weil cookie Methoden nicht als Klassenfunktionen funktionieren
- Aufruf als Klasse erzeugt Dummyobjekt was im Klassennamespace gehalten wird und über Requestgrenzen leaked
- liegt jetzt unter `$::request->{cgi}`

4.1.4.2. $::all_units

- war nötig, weil einige Funktionen in Schleifen zum Teil ein paar hundert mal pro Request eine Liste der Einheiten brauchen, und de als Parameter durch einen Riesenstack von Funktionen geschleift werden müssten.
- Liegt jetzt unter `$::request->{cache}{all_units}`
- Wird nur in `AM->retrieve_all_units()` gesetzt oder gelesen.

4.1.4.3. %::called_subs

- wurde benutzt um callsub deep recursions abzufangen.
- Wurde entfernt, weil callsub nur einen Bruchteil der möglichen Rekursioenen darstellt, und da nie welche auftreten.
- komplette recursion protection wurde entfernt.

4.2. Entwicklung unter FastCGI

4.2.1. Allgemeines

Wenn Änderungen in der Konfiguration von kivitendo gemacht werden, muss der Webserver neu gestartet werden.

Bei der Entwicklung für FastCGI ist auf ein paar Fallstricke zu achten. Dadurch, dass das Programm in einer Endlosschleife läuft, müssen folgende Aspekte beachtet werden.

4.2.2. Programmende und Ausnahmen

Betrifft die Funktionen `warn`, `die`, `exit`, `carp` und `confess`.

Fehler, die dass Programm normalerweise sofort beenden (fatale Fehler), werden mit dem FastCGI Dispatcher abgefangen, um das Programm am Laufen zu halten. Man kann mit `die`, `confess` oder `carp` Fehler ausgeben, die dann vom Dispatcher angezeigt werden. Die kivitendo eigene `$::form->`error()> tut im Prinzip das Gleiche, mit ein paar Extraoptionen. `warn` und `exit` hingegen werden nicht abgefangen. `warn` wird direkt nach STDERR, also in Server Log eine Nachricht schreiben (sofern in der Konfiguration nicht die Warnungen in das kivitendo Log umgeleitet wurden), und `exit` wird die Ausführung beenden.

Prinzipiell ist es kein Beinbruch, wenn sich der Prozess beendet, fcgi wird ihn sofort neu starten. Allerdings sollte das die Ausnahme sein. Quintessenz: Bitte kein `exit` benutzen, alle anderen Exceptionmechanismen sind ok.

4.2.3. Globale Variablen

Um zu vermeiden, dass Informationen von einem Request in einen anderen gelangen, müssen alle globalen Variablen vor einem Request sauber initialisiert werden. Das ist besonders wichtig im `$::cgi` und `$::auth` Objekt, weil diese nicht gelöscht werden pro Instanz, sondern persistent gehalten werden.

In `SL::Dispatcher` gibt es einen sauber abgetrennten Block, der alle kanonischen globalen Variablen listet und erklärt. Bitte keine anderen einführen ohne das sauber zu dokumentieren.

Datenbankverbindungen wird noch ein Guide verfasst werden, wie man sicher geht, dass man die richtige erwischt.

4.2.4. Performance und Statistiken

Die kritischen Pfade des Programms sind die Belegmasken, und unter diesen ganz besonders die Verkaufsrechnungsmaske. Ein Aufruf der Rechnungsmaske in kivitendo 2.4.3 stable dauert auf einem Core2duo mit 4GB Arbeitsspeicher und Ubuntu 9.10 eine halbe Sekunde. In der 2.6.0 sind es je nach Menge der definierten Variablen 1-2s. Ab der Moose/Rose::DB Version sind es 5-6s.

Mit FastCGI ist die neuste Version auf 0,26 Sekunden selbst in den kritischen Pfaden, unter 0,15 sonst.

4.2.5. Bekannte Probleme

4.2.5.1. Encoding Awareness

UTF-8 kodierte Installationen sind sehr anfällig gegen fehlerhfate Encodings unter FCGI. latin9 Installationen behandeln falsch kodierte Zeichen eher unwissend, und geben sie einfach weiter. UTF-8 verweigert bei fehlerhaften Programmpfaden kurzerhand das Ausliefern. Es wird noch daran gearbeitet, alle Fehler da zu beseitigen.

4.3. SQL-Upgradedateien

4.3.1. Einführung

Der alte Mechanismus für SQL-Upgradescripte, der auf einer Versionsnummer beruht und dann in sql/Pg-upgrade nach einem Script für diese Versionsnummer sucht, schränkt sehr ein, z.B. was die parallele Entwicklung im stable- und unstable-Baum betrifft.

Dieser Mechanismus wurde für kivitendo 2.4.1 deutlich erweitert. Es werden weiterhin alle Scripte aus sql/Pg-upgrade ausgeführt. Zusätzlich gibt es aber ein zweites Verzeichnis, sql/Pg-upgrade2. In diesem Verzeichnis muss pro Datenbankupgrade eine Datei existieren, die neben den eigentlich auszuführenden SQL- oder Perl-Befehlen einige Kontrollinformationen enthält.

Neu sind die Kontrollinformationen, die Abhängigkeiten und Prioritäten definieren können werden, sodass Datenbankscripte zwar in einer sicheren Reihenfolge ausgeführt werden (z.B. darf ein "ALTER TABLE" erst ausgeführt werden, wenn die Tabelle mit "CREATE TABLE" angelegt wurde), diese Reihenfolge aber so flexibel ist, dass man keine Versionsnummern mehr braucht.

kivitendo merkt sich dabei, welches der Upgradescripte in sql/Pg-upgrade2 bereits durchgeführt wurde und führt diese nicht erneut aus. Dazu dient die Tabelle "schema_info", die bei der Anmeldung automatisch angelegt wird.

4.3.2. Format der Kontrollinformationen

Die Kontrollinformationen sollten sich am Anfang der jeweiligen Upgradedatei befinden. Jede Zeile, die Kontrollinformationen enthält, hat dabei das folgende Format:

Für SQL-Upgradedateien:

```
-- @key: value
```

Für Perl-Upgradedateien:

```
# @key: value
```

Leerzeichen vor "`value`" werden entfernt.

Die folgenden Schlüsselworte werden verarbeitet:

`tag`
: Wird zwingend benötigt. Dies ist der "Name" des Upgrades. Dieser "tag" kann von anderen Kontrolldateien in ihren Abhängigkeiten verwendet werden (Schlüsselwort "`depends`"). Der "tag" ist auch der Name, der in der Datenbank eingetragen wird.

  Normalerweise sollte die Kontrolldatei genau so heißen wie der "tag", nur mit der Endung ".sql" bzw. "pl".

  Ein Tag darf nur aus alphanumerischen Zeichen sowie den Zeichen _ - ( ) bestehen. Insbesondere sind Leerzeichen nicht erlaubt und sollten stattdessen mit Unterstrichen ersetzt werden.

`charset`
: Empfohlen. Gibt den Zeichensatz an, in dem das Script geschrieben wurde, z.B. "`UTF-8`". Aus Kompatibilitätsgründen mit alten Upgrade-Scripten wird bei Abwesenheit des Tags der Zeichensatz "`ISO-8859-15`" angenommen.

`description`
: Benötigt. Eine Beschreibung, was in diesem Update passiert. Diese wird dem Benutzer beim eigentlichen Datenbankupdate angezeigt. Während der Tag in englisch gehalten sein sollte, sollte die Beschreibung auf Deutsch erfolgen.

`depends`
: Optional. Eine mit Leerzeichen getrennte Liste von "tags", von denen dieses Upgradescript abhängt. kivitendo stellt sicher, dass die in dieser Liste aufgeführten Scripte bereits durchgeführt wurden, bevor dieses Script ausgeführt wird.

  Abhängigkeiten werden rekursiv betrachtet. Wenn also ein Script "b" existiert, das von Änderungen in "a" abhängt, und eine neue Kontrolldatei für "c" erstellt wird, die von Änderungen in "a" und "b" abhängt, so genügt es, in "c" nur den Tag "b" als Abhängigkeit zu definieren.

  Es ist nicht erlaubt, sich selbst referenzierende Abhängigkeiten zu definieren (z.B. "a" -> "b", "b" -> "c" und "c" -> "a").

`priority`
: Optional. Ein Zahlenwert, der die Reihenfolge bestimmt, in der Scripte ausgeführt werden, die die gleichen Abhängigkeitstiefen besitzen. Fehlt dieser Parameter, so wird der Wert 1000 benutzt.

  Dies ist reine Kosmetik. Für echte Reihenfolgen muss "depends" benutzt werden. kivitendo sortiert die auszuführenden Scripte zuerst nach der Abhängigkeitstiefe (wenn "z" von "y" abhängt und "y" von "x", so hat "z" eine Abhängigkeitstiefe von 2, "y" von 1 und "x" von 0. "x" würde hier zuerst ausgeführt, dann "y", dann "z"), dann nach der Priorität und bei gleicher Priorität alphabetisch nach dem "tag".

`ignore`
: Optional. Falls der Wert auf 1 (true) steht, wird das Skript bei der Anmeldung ignoriert und entsprechend nicht ausgeführt.

4.3.3. Hilfsscript dbupgrade2_tool.pl

Um die Arbeit mit den Abhängigkeiten etwas zu erleichtern, existiert ein Hilfsscript namens "`scripts/dbupgrade2_tool.pl`". Es muss aus dem kivitendo-ERP-Basisverzeichnis heraus aufgerufen werden. Dieses Tool liest alle Datenbankupgradescripte aus dem Verzeichnis `sql/Pg-upgrade2` aus. Es benutzt dafür die gleichen Methoden wie kivitendo selber, sodass alle Fehlersituationen von der Kommandozeile überprüft werden können.

Wird dem Script kein weiterer Parameter übergeben, so wird nur eine Überprüfung der Felder und Abhängigkeiten vorgenommen. Man kann sich aber auch Informationen auf verschiedene Art ausgeben lassen:

- Listenform: "**./scripts/dbupgrade2_tool.pl --list**"

  Gibt eine Liste aller Scripte aus. Die Liste ist in der Reihenfolge sortiert, in der kivitendo die Scripte ausführen würde. Es werden neben der Listenposition der Tag, die Abhängigkeitstiefe und die Priorität ausgegeben.

- Baumform: "**./scripts/dbupgrade2_tool.pl --tree**"

  Listet alle Tags in Baumform basierend auf den Abhängigkeiten auf. Die "Wurzelknoten" sind dabei die Scripte, von denen keine anderen abhängen. Die Unterknoten sind Scripte, die beim übergeordneten Script als Abhängigkeit eingetragen sind.

- Umgekehrte Baumform: "**./scripts/dbupgrade2_tool.pl --rtree**"

  Listet alle Tags in Baumform basierend auf den Abhängigkeiten auf. Die "Wurzelknoten" sind dabei die Scripte mit der geringsten Abhängigkeitstiefe. Die Unterknoten sind Scripte, die das übergeordnete Script als Abhängigkeit eingetragen haben.

- Baumform mit Postscriptausgabe: "**./scripts/dbupgrade2_tool.pl --graphviz**"

  Benötigt das Tool "**graphviz**", um mit seiner Hilfe die umgekehrte Baumform in eine Postscriptdatei namens "`db_dependencies.ps`" auszugeben. Dies ist vermutlich die übersichtlichste Form, weil hierbei jeder Knoten nur einmal ausgegeben wird. Bei den Textmodusbaumformen hingegen können Knoten und all ihre Abhängigkeiten mehrfach ausgegeben werden.

- Scripte, von denen kein anderes Script abhängt: "**./scripts/dbupgrade2_tool.pl --nodeps**"

  Listet die Tags aller Scripte auf, von denen keine anderen Scripte abhängen.

4.4. Translations and languages

4.4.1. Introduction

Anmerkung

Dieser Abschnitt ist in Englisch geschrieben, um internationalen Übersetzern die Arbeit zu erleichtern.

This section describes how localization packages in kivitendo are built. Currently the only language fully supported is German, and since most of the internal messages are held in English the English version is usable too.

A stub version of French is included but not functunal at this point.

4.4.2. File structure

The structure of locales in kivitendo is:

`kivitendo/locale/<langcode>/`

where <langcode> stands for an abbreviation of the language package. The builtin packages use two letter ISO 639-1[1] codes, but the actual name is not relevant for the program and can easily be extended to IETF language tags[2] (i.e. "en_GB"). In fact the original language packages from SQL Ledger are named in this way.

In such a language directory the following files are recognized:

LANGUAGE
: This file is mandatory.

  The `LANGUAGE` file contains the self descripted name of the language. It should contain a native representation first, and in parenthesis an english translation after that. Example:

  ```
  Deutsch (German)
  ```

charset
: This file should be present.

  The `charset` file describes which charset a language package is written in and applies to all other language files in the package. It is possible to write some language packages without an explicit charset, but it is still strongly recommended. You'll never know in what environment your language package will be used, and neither UTF-8 nor Latin1 are guaranteed.

  The whole content of this file is a string that can be recognized as a valid charset encoding. Example:

  ```
  UTF-8
  ```

all
: This file is mandatory.

  The central translation file. It is essentially an inline Perl script autogenerated by **locales.pl**. To generate it, generate the directory and the two files mentioned above, and execute the following command:

  ```
  scripts/locales.pl <langcode>
  ```

  Otherwise you can simply copy one of the other languages. You will be told how many are missing like this:

  ```
  $ scripts/locales.pl en
  English - 0.6% - 2015/2028 missing
  ```

  A file named "`missing`" will be generated and can be edited. You can also edit the "`all`" file directly. Edit everything you like to fit the target language and execute **locales.pl** again. See how the missing words get fewer.

Num2text
: Legacy code from SQL Ledger. It provides a means for numbers to be converted into natural language, like `1523 => one thousand five hundred twenty three`. If you want to provide it, it must be inlinable Perl code which provides a `num2text` sub. If an `init` sub exists it will be executed first.

  Only used in the check and receipt printing module.

special_chars
: kivitendo comes with a lot of interfaces to different formats, some of which are rather picky with their accepted charset. The `special_chars` file contains a listing of chars not suited for different file format and provides substitutions. It is written in "Simple Ini" style, containing a block for every file format.

  First entry should be the order of substitution for entries as a whitespace separated list. All entries are interpolated, so `\n`, `\x20` and `\\` all work.

  After that every entry is a special char that should be translated when writing text into such a file.

  Example:

[1] http://en.wikipedia.org/wiki/ISO_639-1
[2] http://en.wikipedia.org/wiki/IETF_language_tag

```
[Template/XML]
order=& < > \n
&=&amp;
<=&lt;
>=&gt;
\n=<br>
```

Note the importance of the order in this example. Substituting < and > befor & would lead to $gt; become >

For a list of valid formats, see the German `special_chars` entry. As of this writing the following are recognized:

```
HTML
URL@HTML
Template/HTML
Template/XML
Template/LaTeX
Template/OpenDocument
filenames
```

The last of which is very machine dependant. Remember that a lot of characters are forbidden by some filesystems, for exmaple MS Windows doesn't like ':' in its files where Linux doesn't mind that. If you want the files created with your language pack to be portable, find all chars that could cause trouble.

missing
: This file is not a part of the language package itself.

  This is a file generated by **scripts/locales.pl** while processing your locales. It's only to have the missing entries singled out and does not belong to a language package.

lost
: This file is not a part of the language package itself.

  Another file generated by **scripts/locales.pl**. If for any reason a translation does not appear anymore and can be deleted, it gets moved here. The last 50 or so entries deleted are saved here in case you made a typo, so that you don't have to translate everything again. If a tranlsation is missing, the lost file is checked first. If you maintain a language package, you might want to keep this safe somewhere.

4.5. Die kivitendo-Test-Suite

4.5.1. Einführung

kivitendo enthält eine Suite für automatisierte Tests. Sie basiert auf dem Standard-Perl-Modul `Test::More`.

Die grundlegenden Fakten sind:

- Alle Tests liegen im Unterverzeichnis `t/`.
- Ein Script (bzw. ein Test) in `f/` enthält einen oder mehrere Testfälle.
- Alle Dateinamen von Tests enden auf `.t`. Es sind selbstständig ausführbare Perl-Scripte.
- Die Test-Suite besteht aus der Gesamtheit aller Tests, sprich aller Scripte in `f/`, deren Dateiname auf `.t` endet.

4.5.2. Voraussetzungen

Für die Ausführung werden neben den für kivitendo eh schon benötigten Module noch weitere Perl-Module benötigt. Diese sind:

- `Test::Deep` (Debian-Paketname: `libtest-deep-perl`; Fedora Core: `perl-Test-Deep`; openSuSE: `perl-Test-Deep`)

4.5.3. Existierende Tests ausführen

Es gibt mehrere Möglichkeiten zum Ausführen der Tests: entweder, man lässt alle Tests auf einmal ausführen, oder man führt gezielt einzelne Scripte aus. Für beide Fälle gibt es das Helferscript `t/test.sh`.

Will man die komplette Test-Suite ausführen, so muss man einfach nur `t/test.sh` ohne weitere Parameter aus dem kivitendo-Basisverzeichnis heraus ausführen.

Um einzelne Test-Scripte auszuführen, übergibt man deren Namen an `t/test.sh`. Beispielsweise:

```
t/test.sh t/form/format_amount.t t/background_job/known_jobs.t
```

4.5.4. Bedeutung der verschiedenen Test-Scripte

Die Test-Suite umfasst Tests sowohl für Funktionen als auch für Programmierstil. Einige besonders zu erwähnende, weil auch während der Entwicklung nützliche Tests sind:

- `t/001compile.t` -- compiliert alle Quelldateien und bricht bei Fehlern sofort ab
- `t/002goodperl.t` -- überprüft alle Perl-Dateien auf Anwesenheit von '`use strict`'-Anweisungen
- `t/003safesys.t` -- überprüft Aufrufe von `system()` und `exec()` auf Gültigkeit
- `t/005no_tabs.t` -- überprüft, ob Dateien Tab-Zeichen enthalten
- `t/006spelling.t` -- sucht nach häufigen Rechtschreibfehlern
- `t/011pod.t` -- überprüft die Syntax von Dokumentation im POD-Format auf Gültigkeit

Weitere Test-Scripte überprüfen primär die Funktionsweise einzelner Funktionen und Module.

4.5.5. Neue Test-Scripte erstellen

Es wird sehr gern gesehen, wenn neue Funktionalität auch gleich mit einem Test-Script abgesichert wird. Auch bestehende Funktion darf und soll ausdrücklich nachträglich mit Test-Scripten abgesichert werden.

4.5.5.1. Ideen für neue Test-Scripte, die keine konkreten Funktionen testen

Ideen, die abgesehen von Funktions noch nicht umgesetzt wurden:

- Überprüfung auf fehlende symbolische Links
- Suche nach Nicht-ASCII-Zeichen in Perl-Code-Dateien (mit gewissen Einschränkungen wie das Erlauben von deutschen Umlauten)
- Test auf DOS-Zeilenenden (\r\n anstelle von nur \n)
- Überprüfung auf Leerzeichen am Ende von Zeilen
- Test, ob alle zu übersetzenden Strings in `locale/de/all` vorhanden sind
- Test, ob alle Webseiten-Templates in `templates/webpages` mit vom Perl-Modul `Template` compiliert werden können

4.5.5.2. Konvention für Verzeichnis- und Dateinamen

Es gibt momentan eine wenige Richtlinien, wie Test-Scripte zu benennen sind. Bitte die folgenden Punkte als Richtlinie betrachten und ihnen soweit es geht folgen:

- Die Dateiendung muss `.t` lauten.
- Namen sind englisch, komplett klein geschrieben und einzelne Wörter mit Unterstrichten getrennt (beispielsweise `bad_function_params.t`).
- Unterverzeichnisse sollten grob nach dem Themenbereich benannt sind, mit dem sich die Scripte darin befassen (beispielsweise `background_jobs` für Tests rund um Hintergrund-Jobs).
- Test-Scripte sollten einen überschaubaren Bereich von Funktionalität testen, der logisch zusammenhängend ist (z.B. nur Tests für eine einzelne Funktion in einem Modul). Lieber mehrere Test-Scripte schreiben.

4.5.5.3. Minimales Skelett für eigene Scripte

Der folgenden Programmcode enthält das kleinstmögliche Testscript und kann als Ausgangspunkt für eigene Tests verwendet werden:

```
use Test::More tests => 0;

use lib 't';

use Support::TestSetup;

Support::TestSetup::login();
```

Wird eine vollständig initialisierte kivitendo-Umgebung benötigt (Stichwort: alle globalen Variablen wie `$::auth`, `$::form` oder `$::lxdebug`), so muss in der Konfigurationsdatei `config/kivitendo.conf` im Abschnitt `testing.login` ein gültiger Login-Name eingetragen sein. Dieser wird für die Datenbankverbindung benötigt.

Wir keine vollständig initialisierte Umgebung benötigt, so kann die letzte Zeile `Support::TestSetup::login();` weggelassen werden, was die Ausführungszeit des Scripts leicht verringert.

4.6. Stil-Richtlinien

Die folgenden Regeln haben das Ziel, den Code möglichst gut les- und wartbar zu machen. Dazu gehört zum Einen, dass der Code einheitlich eingerückt ist, aber auch, dass Mehrdeutigkeit so weit es geht vermieden wird (Stichworte "Klammern" oder "Hash-Keys").

Diese Regeln sind keine Schikane sondern erleichtern allen das Leben!

Jeder, der einen Patch schickt, sollte seinen Code vorher überprüfen. Einige der Regeln lassen sich automatisch überprüfen, andere nicht.

1. Es werden keine echten Tabs sondern Leerzeichen verwendet.
2. Die Einrückung beträgt zwei Leerzeichen. Beispiel:

```
  foreach my $row (@data) {
    if ($flag) {
      # do something with $row
    }

    if ($use_modules) {
      $row->{modules} = MODULE->retrieve(
        id   => $row->{id},
        date => $use_now ? localtime() : $row->{time},
      );
    }
```

```
  $report->add($row);
}
```

3. Öffnende geschweifte Klammern befinden sich auf der gleichen Zeile wie der letzte Befehl. Beispiele:

```
sub debug {
  ...
}
```

oder

```
if ($form->{item_rows} > 0) {
  ...
}
```

4. Schließende geschweifte Klammern sind so weit eingerückt wie der Befehl / die öffnende schließende Klammer, die den Block gestartet hat, und nicht auf der Ebene des Inhalts. Die gleichen Beispiele wie bei 3. gelten.

5. Die Wörter "`else`", "`elsif`", "`while`" befinden sich auf der gleichen Zeile wie schließende geschweifte Klammern. Beispiele:

```
if ($form->{sum} > 1000) {
  ...
} elsif ($form->{sum} > 0) {
  ...
} else {
  ...
}

do {
  ...
} until ($a > 0);
```

6. Parameter von Funktionsaufrufen müssen mit runden Klammern versehen werden. Davon nicht betroffen sind interne Perl-Funktionen, und grep-ähnliche Operatoren. Beispiel:

```
$main::lxdebug->message("Could not find file.");
%options = map { $_ => 1 } grep { !/^#/ } @config_file;
```

7. Verschiedene Klammern, Ihre Ausdrücke und Leerzeichen:

   Generell gilt: Hashkeys und Arrayindices sollten nicht durch Leerzeichen abgesetzt werden. Logische Klammerungen ebensowenig, Blöcke schon. Beispiel:

```
if (($form->{debug} == 1) && ($form->{sum} - 100 < 0)) {
  ...
}

$array[$i + 1]             = 4;
$form->{sum}              += $form->{"row_$i"};
$form->{ $form->{index} } += 1;

map { $form->{sum} += $form->{"row_$_"} } 1..$rowcount;
```

8. Mehrzeilige Befehle

   a. Werden die Parameter eines Funktionsaufrufes auf mehrere Zeilen aufgeteilt, so sollten diese bis zu der Spalte eingerückt werden, in der die ersten Funktionsparameter in der ersten Zeile stehen. Beispiel:

```
  $sth = $dbh->prepare("SELECT * FROM some_table WHERE col = ?",
```

```
                        $form->{some_col_value});
```

b. Ein Spezialfall ist der ternäre Oprator "?:", der am besten in einer übersichtlichen Tabellenstruktur organisiert wird. Beispiel:

```
my $rowcount = $form->{"row_$i"} ? $i
             : $form->{oldcount} ? $form->{oldcount} + 1
             :                     $form->{rowcount} - $form->{rowbase};
```

9. Kommentare

a. Kommentare, die alleine in einer Zeile stehen, sollten soweit wie der Code eingerückt sein.

b. Seitliche hängende Kommentare sollten einheitlich formatiert werden.

c. Sämtliche Kommentare und Sonstiges im Quellcode ist bitte auf Englisch zu verfassen. So wie ich keine Lust habe, französischen Quelltext zu lesen, sollte auch der kivitendo Quelltext für nicht-Deutschsprachige lesbar sein. Beispiel:

```
my $found = 0;
while (1) {
  last if $found;

  # complicated check
  $found = 1 if //
}

$i  = 0        # initialize $i
$n  = $i;      # save $i
$i *= $const;  # do something crazy
$i  = $n;      # recover $i
```

10. Hashkeys sollten nur in Anführungszeichen stehen, wenn die Interpolation gewünscht ist. Beispiel:

```
$form->{sum}      = 0;
$form->{"row_$i"} = $form->{"row_$i"} - 5;
$some_hash{42}    = 54;
```

11. Die maximale Zeilenlänge ist nicht beschränkt. Zeilenlängen unterhalb von 79 Zeichen helfen unter bestimmten Bedingungen, aber wenn die Lesbarkeit unter kurzen Zeilen leidet (wie zum Biespiel in grossen Tabellen), dann ist Lesbarkeit vorzuziehen.

Als Beispiel sei die Funktion `print_options` aus `bin/mozilla/io.pl` angeführt.

12. Trailing Whitespace, d.h. Leerzeichen am Ende von Zeilen sind unerwünscht. Sie führen zu unnötigen Whitespaceänderungen, die diffs verfälschen.

Emacs und vim haben beide recht einfache Methoden zur Entfernung von trailing whitespace. Emacs kennt das Kommande **nuke-trailing-whitespace**, vim macht das gleiche manuell über `:%s/\s\+$//e` Mit `:au BufWritePre * :%s/\s\+$//e` wird das an Speichern gebunden.

13. Es wird kein **perltidy** verwendet.

In der Vergangenheit wurde versucht, **perltidy** zu verwenden, um einen einheitlichen Stil zu erlangen. Es hat sich aber gezeigt, dass **perltidy**s sehr eigenwilliges Verhalten, was Zeilenumbrüche angeht, oftmals gut formatierten Code zerstört. Für den Interessierten sind hier die **perltidy**-Optionen, die grob den beschriebenen Richtlinien entsprechen:

```
-syn -i=2 -nt -pt=2 -sbt=2 -ci=2 -ibc -hsc -noll -nsts -nsfs -asc -dsm
-aws -bbc -bbs -bbb -mbl=1 -nsob -ce -nbl -nsbl -cti=0 -bbt=0 -bar -l=79
-lp -vt=1 -vtc=1
```

14. `STDERR` ist tabu. Unkonditionale Debugmeldungen auch.

kivitendo bietet mit dem Modul `LXDebug` einen brauchbaren Trace-/Debug-Mechanismus. Es gibt also keinen Grund, nach `STDERR` zu schreiben.

Die `LXDebug`-Methode "`message`" nimmt als ersten Paramter außerdem eine Flagmaske, für die die Meldung angezeigt wird, wobei "0" immer angezeigt wird. Solche Meldungen sollten nicht eingecheckt werden und werden in den meisten Fällen auch vom Repository zurückgewiesen.

15.Alle neuen Module müssen use strict verwenden.

`$form`, `$auth`, `$locale`, `$lxdebug` und `%myconfig` werden derzeit aus dem main package importiert (siehe Globale Variablen [41]. Alle anderen Konstrukte sollten lexikalisch lokal gehalten werden.

4.7. Dokumentation erstellen

4.7.1. Einführung

Diese Dokumentation ist in DocBook™ XML geschrieben. Zum Bearbeiten reicht grundsätzlich ein Text-Editor. Mehr Komfort bekommt man, wenn man einen dedizierten XML-fähigen Editor nutzt, der spezielle Unterstützung für DocBook™ mitbringt. Wir empfehlen dafür den XMLmind XML Editor[3], der bei nicht kommerzieller Nutzung kostenlos ist.

4.7.2. Benötigte Software

Bei DocBook™ ist Prinzip, dass ausschließlich die XML-Quelldatei bearbeitet wird. Aus dieser werden dann mit entsprechenden Stylesheets andere Formate wie PDF oder HTML erzeugt. Bei kivitendo übernimmt diese Aufgabe das Shell-Script **scripts/build_doc.sh**.

Das Script benötigt zur Konvertierung verschiedene Softwarekomponenten, die im normalen kivitendo-Betrieb nicht benötigt werden:

- Java[4] in einer halbwegs aktuellen Version
- Das Java-Build-System Apache Ant[5]
- Das Dokumentations-System Dobudish für DocBook™ 4.5, eine Zusammenstellung diverser Stylesheets und Grafiken zur Konvertierung von DocBook™ XML in andere Formate. Das Paket, das benötigt wird, ist zum Zeitpunkt der Dokumentationserstellung `dobudish-nojre-1.1.4.zip`, aus auf code.google.com[6] bereitsteht.

Apache Ant sowie ein dazu passendes Java Runtime Environment sind auf allen gängigen Plattformen verfügbar. Beispiel für Debian/Ubuntu:

```
apt-get install ant openjdk-7-jre
```

Nach dem Download von Dobudish muss Dobudish im Unterverzeichnis `doc/build` entpackt werden. Beispiel unter der Annahme, das Dobudish™ in `$HOME/Downloads` heruntergeladen wurde:

```
cd doc/build
unzip $HOME/Downloads/dobudish-nojre-1.1.4.zip
```

4.7.3. PDFs und HTML-Seiten erstellen

Die eigentliche Konvertierung erfolgt nach Installation der benötigten Software mit einem einfachen Aufruf direkt aus dem kivitendo-Installationsverzeichnis heraus:

[3] http://www.xmlmind.com/xmleditor/
[4] http://www.oracle.com/technetwork/java/index.html
[5] http://ant.apache.org/
[6] http://code.google.com/p/dobudish/downloads/list

```
./scripts/build_doc.sh
```

4.7.4. Einchecken in das Git-Repository

Sowohl die XML-Datei als auch die erzeugten PDF- und HTML-Dateien sind Bestandteil des Git-Repositories. Daraus folgt, dass nach Änderungen am XML die PDF- und HTML-Dokumente ebenfalls gebaut und alles zusammen in einem Commit eingecheckt werden sollten.

Die "`dobudish`"-Verzeichnisse bzw. symbolischen Links gehören hingegen nicht in das Repository.