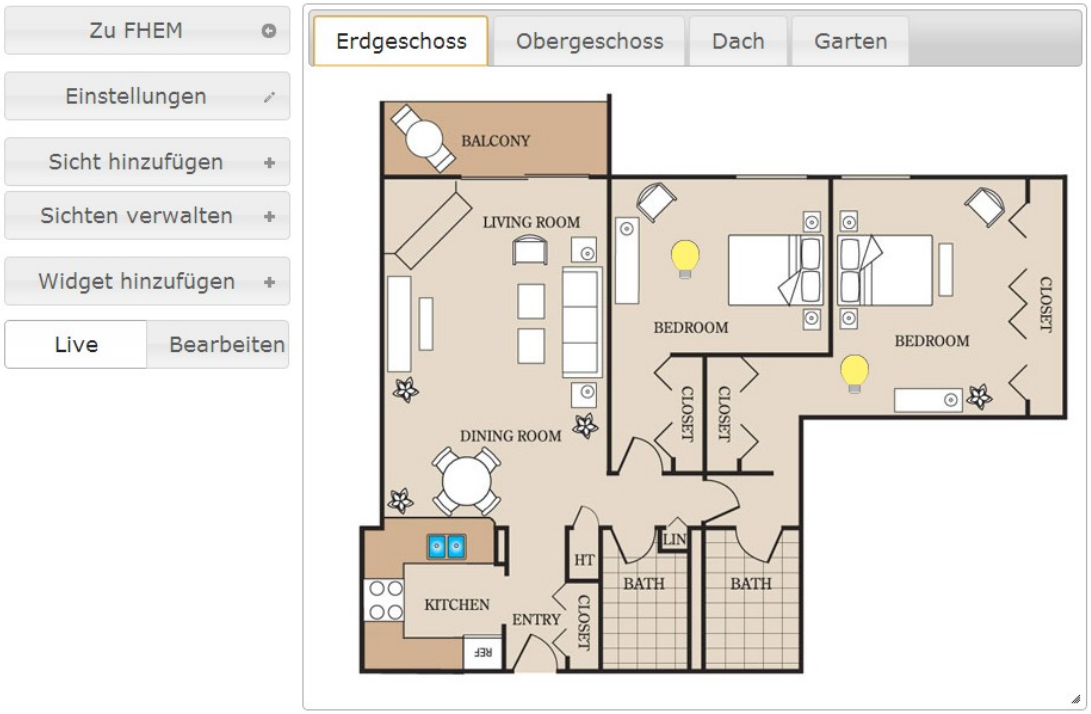
Projektarbeit
Wintersemester 2012 / 2013

Thema:
Entwicklung eines Web-Frontends für einen Hausautomatisierungsserver



Projektteam:
Markus Mangei (mangei.markus@googlemail.com)
Daniel Weisensee (daniel.weisensee@googlemail.com)

Betreuender Professor:
Prof. Dr. rer. nat. Peter A. Henning

Hochschule Karlsruhe - Technik und Wirtschaft
Studiengang Informatik (Bachelor)

Datum: 10. März 2013

Inhaltsverzeichnis

1 - Einführung
- 1.1 Was ist FHEM
- 1.2 Projekt - YAF
  - 1.2.1 Ablauf des Projekts

2 - Architektur - YAF
- 2.1 Ordner und Dateistruktur

3 - Installation von FHEM und YAF
- 3.1 Vorbereitungen für FHEM
- 3.2 FHEM installieren
- 3.3 FHEM starten und stoppen
- 3.4 Vorbereitungen für YAF
  - 3.4.1 Ausgangslage ermitteln
  - 3.4.2 Modul XML::LibXML aus Paketverwaltung installieren
  - 3.4.3 Module aus dem CPAN Repository installieren
  - 3.4.4 Empfohlene Installationsschritte
- 3.5 YAF installieren
  - 3.5.1 Installationsvorgang
  - 3.5.2 Konfiguration anpassen

4 - Verwendung von YAF
- 4.1 YAF GUI
- 4.2 Einstellungen
- 4.3 Sichten
  - 4.3.1 Sicht hinzufügen
- 4.4 Modi von YAF
  - 4.4.1 Live-Modus
  - 4.4.2 Bearbeiten-Modus
- 4.5 Widgets
  - 4.5.1 Widgets einfügen
  - 4.5.2 Widgets bearbeiten / löschen
  - 4.5.3 Widgets aktualisieren

5 - Eigenes Widget erstellen
- 5.1 Datei- und Verzeichnisstruktur
- 5.2 Funktionen der fs20easylamp.pm
- 5.3 Auslieferung eines Widgets

6 - Ideen für die Zukunft von YAF

7 - Fazit

A - Perl Module manuell installieren
- A.1 XML::LibXML aus CPAN installieren
- A.2 Perl Module manuell installieren

1 - Einführung

1.1 Was ist FHEM

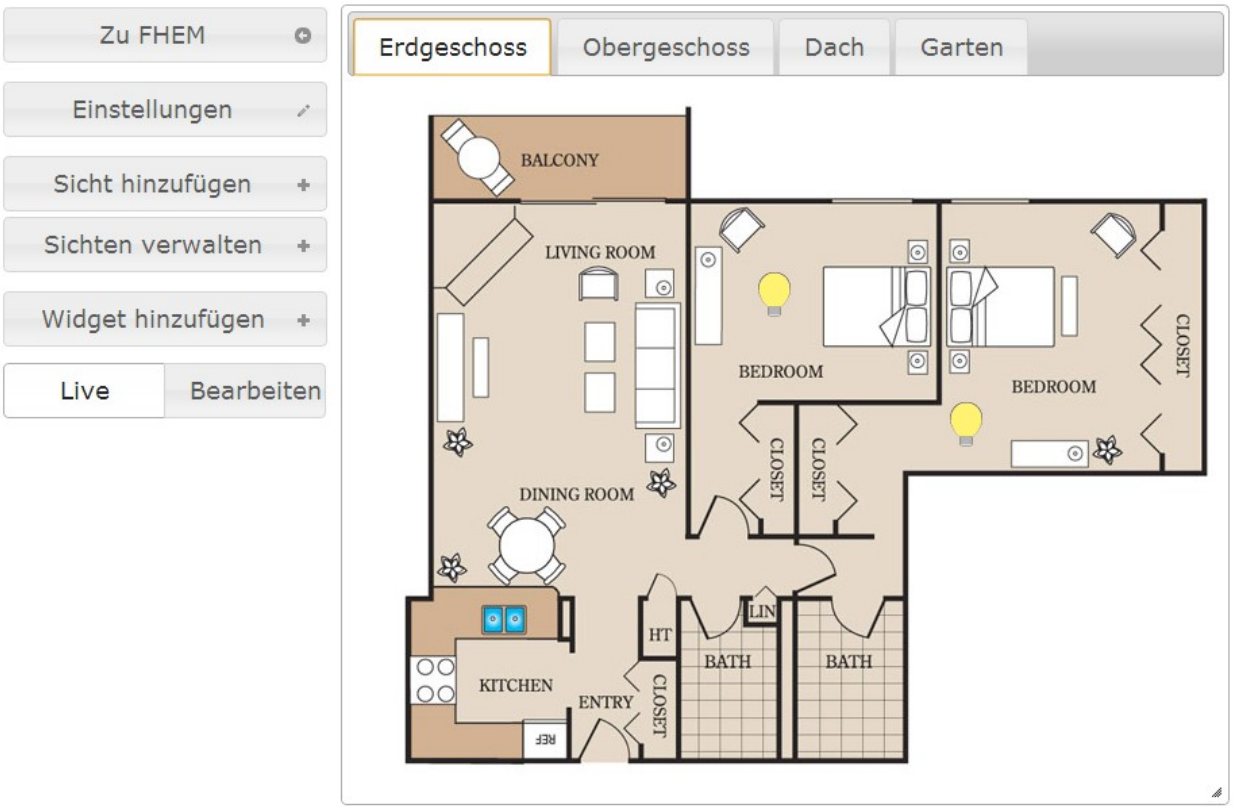
FHEM steht für “Freundliche Hausautomatisierung und Energie-Messung” und ist ein auf Perl basierender Server für die Hausautomatisierung. Ansteuern von Lampen, Schaltern, Heizung usw. (Aktoren) und Aufzeichnen von Temperatur, Feuchtigkeit, Stromverbrauch usw. (Sensoren) sind nur eine mögliche Auswahl an Funktionen, die mit FHEM umgesetzt werden können. Eine Besonderheit an FHEM ist, dass dieser Server durch die Verwendung von Perl auf vielen Geräten läuft. Spezielle Unterstützung findet man z.B. für die FRITZ!Box sowie den Raspberry Pi. Der Server lässt sich mittels Web-Frontends, Telnet, Command Line oder TCP/IP steuern und bietet somit auch hier einige Alternativen an. Da der Server unter der GPL Lizenz veröffentlicht wurde, wächst die FHEM Community stetig an.

http://fhem.de/fhem.html
http://forum.fhem.de/
http://fhem.de/Heimautomatisierung-mit-fhem-V2.pdf
http://www.fhemwiki.de/wiki/FHEM
https://groups.google.com/forum/#!forum/fhem-developers
https://groups.google.com/forum/#!forum/fhem-users

1.2 Projekt - YAF

YAF entstand als Projektarbeit von Daniel Weisensee und Markus Mangei an der Hochschule Karlsruhe - Technik und Wirtschaft. Es steht für “Yet Another Floorplan” und soll eine Alternative zum bisher vorhandenen Floorplan bieten. YAF basiert auf Client-Seite aus den JavaScript Frameworks JQuery und JQuery UI, serverseitig werden die CPAN Module XML::LibXML (bindet libxml2 an Perl an) und JSON::XS verwendet, um die Konfiguration zu persisitieren und um Daten zwischen der Oberfläche und dem Server austauschen zu können.
Durch die Erweiterbarkeit von Widgets soll YAF flexibel gehalten werden. Mit Hilfe dieser Schnittstelle können problemlos Widgets von verschiedenen Entwicklern veröffentlicht werden, ohne dass sich diese über gewünschte Änderungen am YAF Code mit der Community abstimmen müssen. Es soll ähnlich dem Prinzip der Widgets unter Android oder Windows funktionieren. Widgets sollen speziell für FHEM Plugins geschrieben werden, um somit möglichst komfortable Oberflächen bieten zu können.

Das Projekt ist freie Software unter der GNU General Public License und befindet sich auf einem Subversion Server von Google:
http://code.google.com/p/yaf2012/



1.2.1 Ablauf des Projekts

Da das Projekt während des regulären Semesters als Teil einer Prüfungsleistung stattfand, war der Rahmen zeitlich begrenzt und die Ziele wurden dementsprechend gesetzt. Vor diesem Projekt hatte allerdings noch keiner der Entwickler Erfahrungen mit Hausautomatisierung und FHEM gesammelt und auch Perl musste erst erlernt werden. Anfangs war es somit schwer abzuschätzen, wie viel in dieser Zeit erreicht werden kann. Um Erfahrungen zu sammeln war ein erstes Ziel, FHEM mit Floorplan zu installieren und den Aufbau zu analysieren. Mit diesem Wissen konnten im nächsten Schritt Ziele definiert werden. Unter anderem wurde definiert, was ein alternativer Floorplan alles unterstützen muss und wie viel man in der für das Projekt verfügbaren Zeit erreichen kann. Als nächstes Ziel war bereits eine Oberfläche geplant, die mit Hilfe einer modernen Oberfläche verschiedene Sichten (für z.B. verschiedene Stockwerke) anzeigen / erstellen / löschen und bearbeiten kann. Die Konfiguration hierfür soll in einer XML Datei persistiert werden. Die Daten, welche von der Oberfläche zum Server und vom Server zur Oberfläche gesendet werden, sollen über JSON codierte Nachrichten übertragen werden. Als nächstes wurde überlegt, wie man die Oberfläche möglichst flexibel halten kann, ohne immer wieder etwas am Grundgerüst von YAF ändern zu müssen. Hier hatte sich der Lösungsansatz mit Widgets als bester Weg herausgestelt. Es soll also die Entwicklung von Widgets von der Entwicklung von YAF getrennt werden. Um diesen in der Theorie entwickelten Ansatz besser zu veranschaulichen wird ein Beispiel Widget (fs20easylamp) entwickelt, um daran die Funktionsweise der Widgets in Absatz 5 zu erklären. Als Abschluss wird eine Dokumentation verfasst, welche es Interessierten und späteren Projektgruppen ermöglichen soll einen besseren Einstieg in das Projekt zu finden.

2 - Architektur - YAF

2.1 Ordner und Dateistruktur

YAF wird als Dateiarchiv ausgeliefert, in dem sich Dateien und Ordner in der unten gezeigten Struktur befinden. Dieses Archiv muss einfach in das FHEM Verzeichnis entpackt werden. Es wird außerdem ein Makefile mitgeliefert, welches den Kopiervorgang übernimmt.

FHEM/YAF

- *YAFConfig.pm*
- *YAFWidgets.pm*
- widgets
  - widget1
    - www
      - [...]
    - *widget1.pm*
- www
  - css
    - *yaf.css*
    - [...]
  - img
    - [...]
  - js
    - *yaf-dialogs.js*
    - [...]
  - xml
    - *yafConfig.xml*
    - *xmlSchema.xsd*
  - *yaf.htm*

FHEM/

- *01_YAF.pm*

3 - Installation von FHEM und YAF

Im folgenden wird der vollständige Installationsvorgang auf einem Linux System beschrieben, was FHEM, YAF und notwendige Abhängigkeiten beinhaltet. Getestet wurde dieses Vorgehen auf “Debian Squeeze 64 Bit”, “Ubuntu 12.04 LTS 32Bit”, “Ubuntu Server Edition 12.10 32Bit” und “Raspbian Wheezy 2012-10-28”. Auf weitere Linuxdistributionen ist diese Vorgehensweise übertragbar.

3.1 Vorbereitungen für FHEM

Sollte FHEM auf dem Zielsystem bereits erfolgreich im Einsatz sein, so können die Abschnitte 3.1, 3.2 und 3.3 übersprungen werden.
Um FHEM installieren und verwenden zu können, werden zunächst einige Tools benötigt. Diese sind perl, gcc, make und telnet, die unter Umständen bereits vorinstalliert sind. Ist das nicht der Fall und die apt (Advanced Packaging Tool) Paketverwaltung ist im Einsatz, so lassen sich die notwendigen Pakete komfortabel mit den folgenden Befehlen installieren:

```
$ sudo apt-get install perl
$ sudo apt-get install gcc (oder vergleichbarer C Compiler)
$ sudo apt-get install make
$ sudo apt-get install telnet
```

3.2 FHEM installieren

Es gibt mehrere Möglichkeiten FHEM zu installieren. In den offiziellen Installationsanleitungen finden sich nähere Erläuterungen dazu.

Möglichkeit 1

```
$ wget http://fhem.de/fhem-5.3.deb
$ sudo dpkg --install fhem-5.3.deb
$ sudo apt-get -f install
```

FHEM wird in den Pfad /opt/fhem installiert. Dieser Ordner hat folgende Rechte:

```
drwxr-xr-x 7 fhem root 4096 Mar 10 16:30 fhem
```

Bei der Installation über den Paketmanager wird möglicherweise ein Startskript abgelegt, das dafür sorgt, dass FHEM bei Systemstart und nach dem Beenden des Servers automatisch wieder gestartet wird.

Möglichkeit 2:

```
$ wget http://fhem.de/fhem-5.3.tar.gz
$ tar -zxf fhem-5.3.tar.gz
$ cd fhem-5.3
$ fhem-5.3$ sudo make install
```

FHEM wird nach /opt/fhem installiert. Dieser Ordner hat folgende Rechte:

```
drwxr-xr-x  7 root root 4096 Mar 10 16:37 fhem
```

Experimentell: Hier ist es nur möglich, FHEM mittels root Rechten zu starten. Will man das vermeiden, so hilft folgender Workaround:

```
$ sudo chown user:group fhem.cfg
$ sudo chown user:group fhem.pl
$ sudo chown -R user:group log/
```

3.3 FHEM starten und stoppen

FHEM starten

```
/opt/fhem$ perl fhem.pl fhem.cfg
```

Je nach Ordner- und Dateirechten eventuell nur mit root Rechten (`sudo`) ausführbar.

FHEM stoppen

```
/opt/fhem$ fhem.pl 7072 “shutdown”
```

Falls dieser direkte Aufruf nicht funktioniert, kann man alternativ zunächst eine Telnet Verbindung aufbauen und dort den shutdown Befehl absenden:

```
$ telnet localhost 7072
shutdown
```

Nach dem Start von FHEM ist die Oberfläche erreichbar über http://localhost:8083/fhem.

Damit wurde FHEM erfolgreich installiert und in den folgenden Abschnitten wird nun die Installation unseres YAF Projekts beschrieben.

3.4 Vorbereitungen für YAF

Zunächst müssen die notwendigen Voraussetzungen für den Betrieb von YAF geschaffen werden. Da im Backend Abhängigkeiten zu bestimmten Perl Modulen sowie einer XML Bibliothek bestehen und diese nicht mit ausgeliefert werden können, müssen diese zunächst installiert werden. Dafür bestehen mehrere Möglichkeiten. Aus der Erfahrung heraus zeigt sich, dass es an dieser Stelle durchaus Probleme geben kann, sodass es sehr hilfreich ist, alle Möglichkeiten zu kennen, um bei Schwierigkeiten alternative Wege einschlagen zu können. Konkret werden die Module XML::LibXML (Binding für die libxml2 Bibliothek), XML::LibXML::PrettyPrint und JSON:XS sowie deren jeweilige Abhängigkeiten (was auch die libxml2 umfasst) benötigt.

3.4.1 Ausgangslage ermitteln

Das Modul XML::LibXML inklusive der libxml2 Bibliothek wird für einige Linuxdistributionen als Paket zur Installation über einen Paketmanager beispielsweise unter dem Namen libxml-libxml-perl angeboten. Im ersten Schritt sollte also überprüft werden, ob dieses Paket bereits installiert ist:

```
$ sudo apt-get install libxml-libxml-perl
```

Der Aufruf ermittelt, ob das gewünschte Paket verfügbar ist und falls ja, ob es bereits installiert wurde. Falls es noch nicht installiert wurde, erhält man eine Abfrage, die man an dieser Stelle am besten zunächst mit “n” für no beantwortet, um das Paket nicht zu installieren.

Eine übliche Bezugsquelle für Perl Module ist das CPAN (Comprehensive Perl Archive Network), ein Online-Repository, das sehr viele Perl Module anbietet und deren Installation und Verwaltung vereinfacht. Alle für YAF benötigten Module sind dort zu finden. Es sollte zunächst mit Hilfe des Befehls `instmodsh` und folgendem `l` (list) überprüft werden, welche Module bereits auf dem Zielrechner installiert sind:

```
$ instmodsh
cmd? l
      Perl
      …

cmd? quit
```

Erscheinen in dieser Liste XML::LibXML::PrettyPrint, JSON::XS und XML::LibXML bzw. wurde XML::LibXML alternativ als Paket installiert, so sind alle Voraussetzungen erfüllt und man kann sich der Installation von YAF widmen. Ist dies nicht der Fall, so müssen die fehlenden Module anhand der folgenden Abschnitte installiert werden.
Achtung: Wurde XML::LibXML über das Linuxpaket installiert, so wird es an dieser Stelle nicht aufgelistet!

3.4.2 Modul XML::LibXML aus Paketverwaltung installieren

Einige Distributionen bieten ein passendes Paket über den Paketmanager an, wodurch die Installation vereinfacht wird. Oftmals ist das unter dem Namen libxml-libxml-perl der Fall und das Paket kann wie folgt einfach installiert werden:

```
$ sudo apt-get install libxml-libxml-perl
```

Achtung: XML::LibXML ist ein Perl Binding für libxml2, setzt also das Vorhandensein der nötigen Bibliotheken voraus. Die Installation über die Paketverwaltung installiert alle notwendigen Bibliotheken mit, was den Vorgang deutlich vereinfacht.

3.4.3 Module aus dem CPAN Repository installieren

Die Installation der Module aus dem CPAN ist der gängigste Weg, um notwendige Perl Module zu installieren.

Die CPAN Konsole lässt sich wie folgt aufrufen (root Rechte sind sinnvoll):
`$ sudo perl -MCPAN -e shell`

Beim ersten Aufruf muss diese zunächst konfiguriert werden. Auf die Frage "`Would you like to configure as much as possible automatically?`" kann mit "`yes`" geantwortet werden. In den eckigen Klammern wird eine Standardantwort vorgeschlagen, die mit Druck auf Enter ausgewählt wird. `[local::lib]` und auch die weiteren Abfragen können also mit Enter bestätigt werden. Die folgende automatische Ausführung der Konfiguration kann einen Moment dauern.

Erfahrungsgemäß funktioniert die automatische Konfiguration nicht immer auf Anhieb. Empfehlenswert ist auf jeden Fall eine Aktualisierung von CPAN:

```
$ sudo perl -MCPAN -e shell
cpan> install CPAN
cpan> reload cpan
cpan> quit
```

Eine erneute Konfiguration kann wie folgt vorgenommen werden:

```
$ sudo perl -MCPAN -e shell
cpan> o conf init
```

Bei Problemen kann ein erneuter Aufruf von `install CPAN` helfen.
Anschließend können die entsprechenden Module installiert werden. Dies geschieht wie folgt:

```
$ sudo perl -MCPAN -e shell
cpan> install modulname
```

3.4.4 Empfohlene Installationsschritte

Die soeben in 3.4.2. und 3.4.3. beschriebenen Schritte führen zur empfohlenen Installation der YAF Abhängigkeiten. Im Idealfall wird aus der Paketverwaltung libxml-libxml-perl installiert, das libxml2 und XML::LibXML beinhaltet. Die beiden anderen Perl Module XML::LibXML::PrettyPrint und JSON::XS werden inklusive ihrer Abhängigkeiten aus dem CPAN Repository installiert. Damit reduzieren sich die notwendigen Schritte im Kern auf die folgenden:

```
$ sudo apt-get install libxml-libxml-perl
$ sudo perl -MCPAN -e shell
cpan> install XML::LibXML::PrettyPrint
cpan> install JSON::XS
```

Gibt man anschließend die Liste der installierten Perl Module mit `instmodsh` und `l` aus, so ergibt sich mindestens die folgende Liste:

```
$ instmodsh
cmd? l
      CPAN
      Carp
      ExtUtils::MakeMaker
      JSON::XS
      Perl
      Pragmatic
      Sub::Uplevel
      Test::Warn
      Tree::DAG_Node
      XML::LibXML::PrettyPrint
      YAML
      common::sense
```

Damit wären alle notwendigen Vorbereitungen getroffen, um im nächsten Schritt YAF zu installieren. Führen die bisher beschriebenen Schritte allerdings nicht zum Erfolg, so finden sich im Anhang alternative Schritte, um die Voraussetzungen für YAF zu schaffen. Diese sind allerdings etwas aufwändiger und konnten bisher nicht alle im Detail von uns getestet werden.

3.5 YAF installieren

Es muss zunächst sichergestellt sein, dass FHEM sowie die Module XML::LibXML, XML::LibXML::PrettyPrint und JSON::XS, wie in den Abschnitten 3.2 respektive 3.4 beschrieben, installiert sind. Ist dies der Fall, kann die eigentliche Installation von YAF erfolgen.

YAF kann bezogen werden aus dem Subversion Repository des Projekts. Unter Downloads finden sich die aktuellen Versionsreleases: http://code.google.com/p/yaf2012/downloads/list
Alternativ kann auch der aktuelle trunk ausgecheckt werden.

3.5.1 Installationsvorgang

Nach dem Download eines Versionsreleases sind die folgenden, wenigen Schritte notwendig, um YAF zu installieren:

1. Entpacken des YAF Archiv
   `$ tar -zxf yaf-0.1.tar.gz`
2. In das Verzeichnis mit den entpackten Dateien wechseln
   `$ cd yaf-0.1`
3. Installation durchführen
   `$ make install`

Es wird also ein Makefile mitgeliefert, welches unter der Annahme funktioniert, dass FHEM im Ordner `/fhem/opt` installiert ist. Ein entsprechender Hinweis teilt mit, ob die Installation erfolgreich verlaufen ist.

3.5.2 Konfiguration anpassen

Im zweiten Schritt muss nun noch die fhem.cfg der FHEM Installation angepasst werden. YAF wird integriert, indem ein entsprechendes Attribut definiert wird. Dies kann an beliebiger Stelle innerhalb der fhem.cfg geschehen:

```
define yaf YAF
```

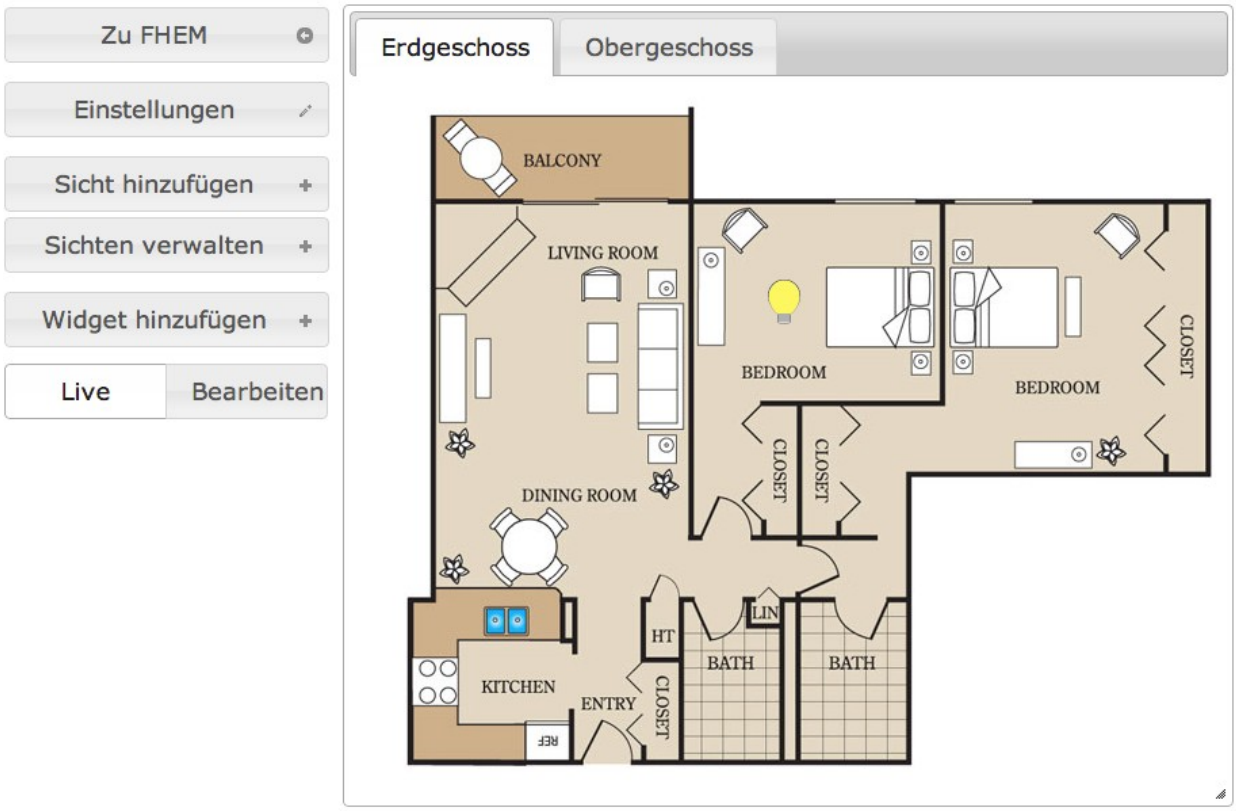
Damit ist die Installation beendet.

4 - Verwendung von YAF

Falls alle Schritte der Installation erfolgreich durchgeführt wurden, ist YAF nach dem Start (bzw. Neustart) des FHEM Servers integriert und auf der Oberfläche von FHEM befindet sich ein Link zu YAF (siehe Abbildung auf Seite 4), welcher zur GUI führt.

4.1 YAF GUI

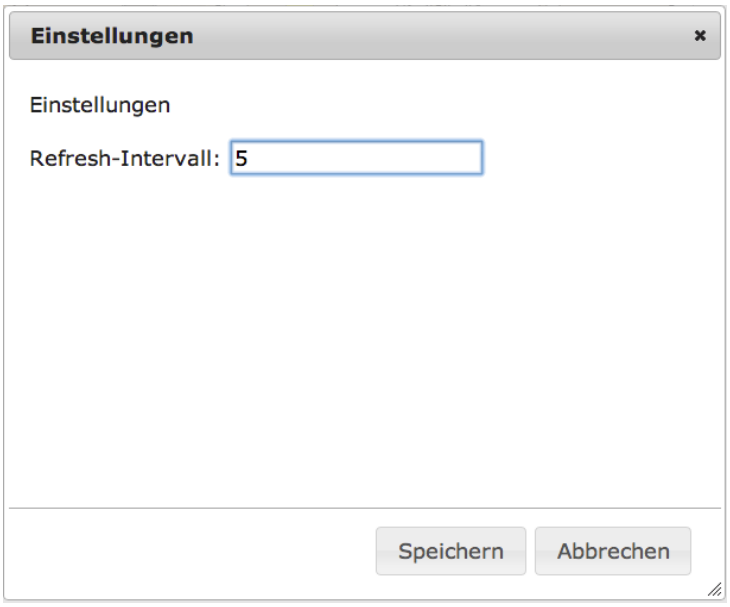
Die folgende Abbildung zeigt das Hauptfenster von YAF.



Auf der linken Seite sind die Buttons vereint, die wichtige Steuermöglichkeiten bieten. Auf der rechten Seite ist ein Container mit den Sichten sichtbar. Ein Klick auf “Zu FHEM” führt zurück auf die bekannte FHEM Oberfläche.

4.2 Einstellungen

Ein Klick auf “Einstellungen” öffnet den Einstellungsdialog, in dem sich aktuell nur das Refresh-Intervall einstellen lässt. Das Refresh-Intervall definiert die Anzahl Sekunden, die gewartet wird, bis sich die Widgets aktualisieren. Beispielsweise liest das fs20easylamp Widget dann nach Ablauf des Zeitintervalls seinen aktuellen Zustand aus (on / off). Der Wert muss ein positiver Integerwert sein, um gespeichert werden zu können.

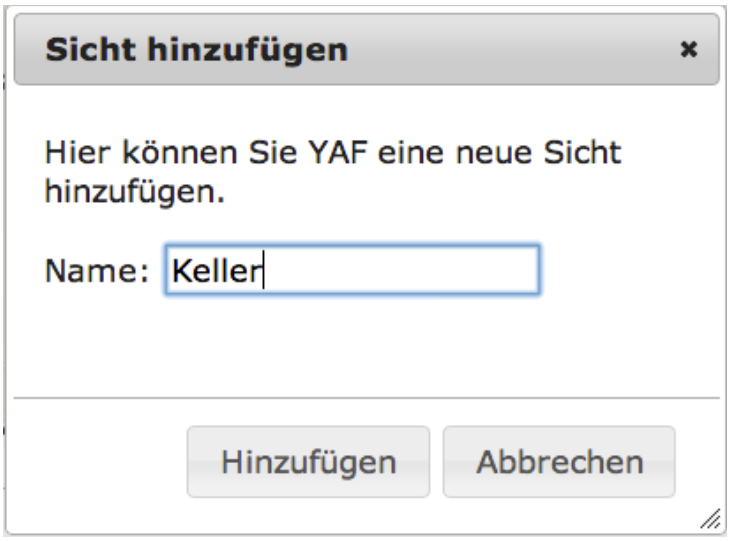


4.3 Sichten

Auf oberster Ebene verwendet YAF sogenannte Sichten, die in Form von Tabs dargstellt werden. Mittels einem Klick auf das Tab mit dem jeweiligen Namen der Sicht, kann zwischen den einzelnen Sichten umgeschaltet werden. Sie sind dafür geeignet, beispielsweise verschiedene Stockwerke darzustellen, können aber auch einfach nur eine Sammlung an Widgets anzeigen. Weitere Einsatzmöglichkeiten sind denkbar. Über das Menü auf der linken Seite können die Sichten verwaltet werden.

4.3.1 Sicht hinzufügen

Ein Klick auf “Sicht hinzufügen” öffnet einen Dialog, der nach dem Namen der neuen Sicht fragt. Mit Klick auf “Hinzufügen” wird das Erzeugen der neuen Sicht bestätigt.

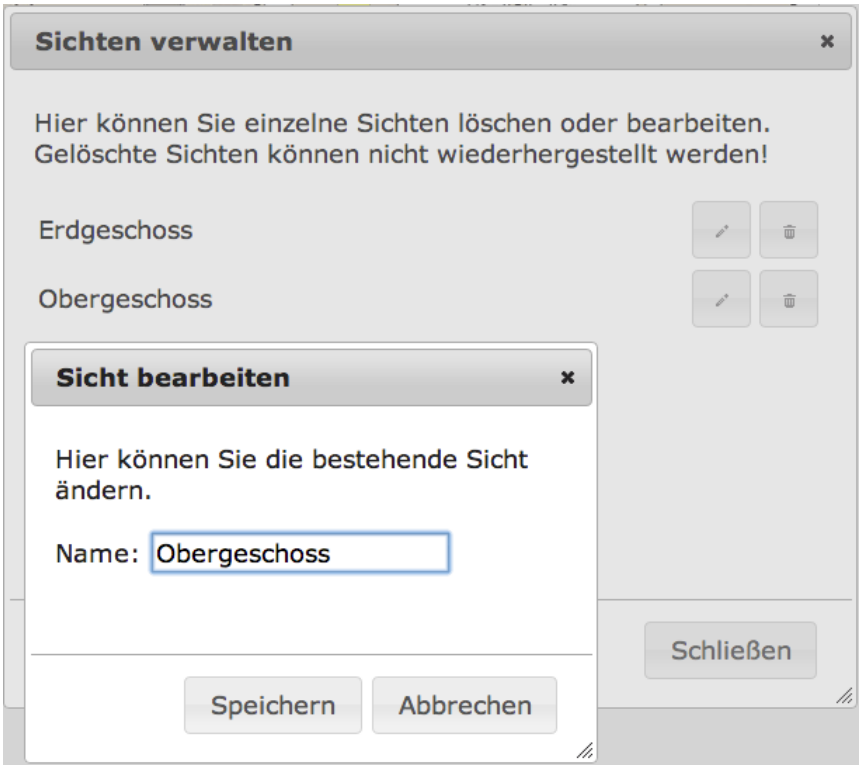


Die neue Sicht wird in der XML Konfiguration gespeichert und erhält einen Standardhintergrund, der leider momentan nur über eine Änderung des entsprechenden Pfades in der yafConfig.xml änderbar ist:

```
<view id="1" name="Keller">
      <backgrounds>
            <background img_url="./img/eg.jpeg" x_pos="1" y_pos="1"/>
      </backgrounds>
      [...]
</view>
```

4.3.2 Sichten verwalten

Der Dialog “Sichten verwalten” bietet die Möglichkeit, die bestehenden Sichten umzubennen und zu löschen.



4.4 Modi von YAF

Die YAF Oberfläche bietet zwei verschiedene Modi. Den Live- sowie den Bearbeiten-Modus. Durch Umschalten der Modi bietet die Oberfläche unterschiedliche Funktionen. Der aktive Modus wird weiß dargestellt.

4.4.1 Live-Modus

Im Live-Modus werden Klicks auf ein Widget ausgeführt und an das entsprechende Widget weitergeleitet. Die Funktion kann von Widget zu Widget verschieden sein. Es kann auch der Fall sein, dass ein Widget keine Funktion zur Verfügung stellt.

4.4.2 Bearbeiten-Modus

Im Bearbeiten-Modus können Widgets entweder via Drag&Drop verschoben werden oder über dessen Menü bearbeitet oder gelöscht werden. Die Funktionen Bearbeiten und Löschen können über das Menü erreicht werden, welches sich beim Klick öffnet.

4.5 Widgets

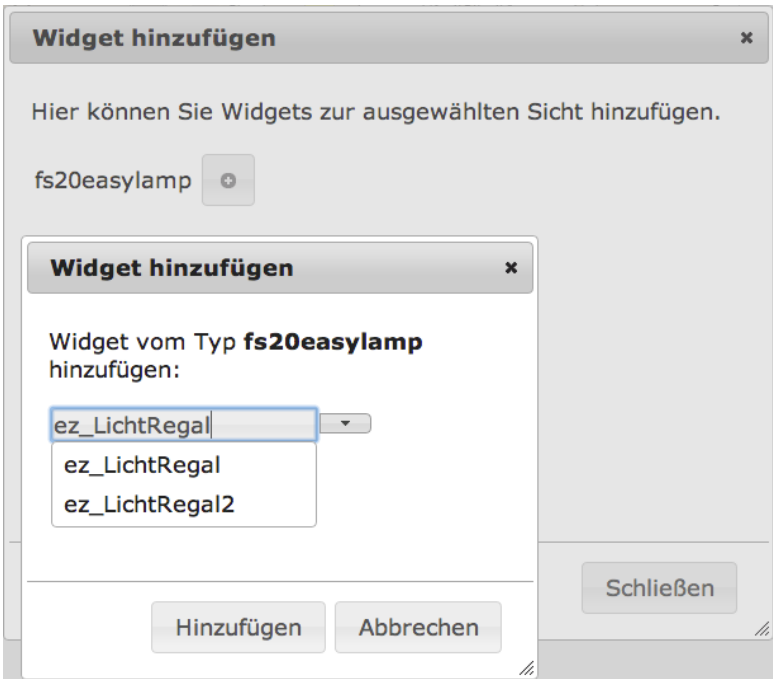
Die Widgets selbst sind sehr flexibel und interaktiv. Sie können je nach Modus und Widget Implementierung verschiedene Funktionen bieten, die direkt durch Klick auf das Widget im Live- oder Bearbeiten-Modus ausgelöst werden. Außerdem gibt es einen Menüeintrag “Widget hinzufügen”, um Widgets einzufügen.

4.5.1 Widgets einfügen

Links am Rand befindet sich der Button “Widget hinzufügen”, mit dem man vorhandene Widgets auf der aktiven Sicht einfügen kann. Es öffnet sich ein Dialog, in dem die geladenen Widgets angezeigt werden. Durch einen Klick auf den “+”-Button öffnet sich das Konfigurationsmenü des Widgets. Wenn das Widget erfolgreich erzeugt wurde, dann wird es im linken oberen Eck der Oberfläche angezeigt und kann anschließend im Bearbeiten-Modus nach Belieben verschoben werden.

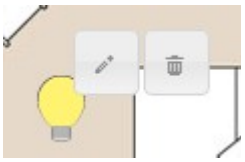
In Fall der folgenden Abbildung handelt es sich um das Widget fs20easylamp. Die Combobox enthält die in der fhem.cfg definierten FS20 Geräte:

```
define ez_LichtRegal FS20 6969 01
define ez_LichtRegal2 FS20 6969 02
```



4.5.2 Widgets bearbeiten / löschen

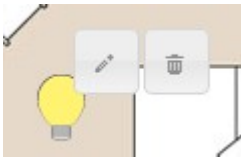
Um ein Widget zu bearbeiten oder zu löschen, kann man es im Bearbeiten-Modus anklicken. Beim Klick öffnet sich ein Menü, in dem man entweder den Button für Löschen oder zum Bearbeiten klicken kann.



4.5.3 Widgets aktualisieren

Die Widgets aktualisieren sich alle x Sekunden selbst, folgend dem Refresh-Interval (siehe Einstellungsdialog 4.2).

5 - Eigenes Widget erstellen



Diese Anleitung orientiert sich am vorhandenen Widget fs20easylamp. Fs20easylamp dient dazu, den Status eines FS20 Schalter darzustellen, sowie ihn schalten (ON / OFF) zu können. Anhand dieses Beispiels sollen sich weitere Widgets erstellen lassen. Um ein eigenes Widget zu erstellen muss sich der Entwickler lediglich an die Dateistruktur halten, sowie einige Funktionen implementieren, welche von YAF aufgerufen werden. Wenn sich der Entwickler an diese Vorgaben hält, dann lassen sich die neuen Widgets genau wie das Beispiel Widget verschieben, bearbeiten und löschen.

5.1 Datei- und Verzeichnisstruktur

Der Ordner, in dem das Widget gepackt ist, muss immer einen eindeutigen Namen haben. Zwei unterschiedliche Widgets dürfen niemals den selben Namen haben, da dieser ein Widget eindeutig identifiziert. Das Widget muss immer eine Perl Datei haben, die den Namen des Widgets trägt. Wenn das Widget wie im Beispiel fs20easylamp heißt, dann muss auch die Perl Datei fs20easylamp.pm heißen.

Die Struktur sollte dann folgendermaßen aussehen:

FHEM/YAF/widgets/fs20easylamp/
fs20easylamp.pm
/www/
[...]

5.2 Funktionen der fs20easylamp.pm

fs20easylamp_get_widgetcss() {}

Hier kann CSS Code definiert werden, welcher in YAF geladen werden soll. Die Funktion wird nur einmal aufgerufen, wenn die gesamte YAF Seite neu angezeigt wird.

fs20easylamp_get_widgethtml() {}

Hier wird HTML Code definiert, der im Widget DIV in YAF angezeigt wird, wenn das Widget eingebunden wird. Da bei fs20easylamp das Bild der Lampe über eine CSS Klasse definiert wird, besitzt diese Funktion dort einen leeren String als Rückgabewert. Diese Funktion wird pro angezeigtem Widget aufgerufen. Wenn also drei Lampen mit fs20easylamp angezeigt werden, dann wird auch die funktion fs20easylamp_get_widgethtml() dreimal aufgerufen.

fs20easylamp_get_widgetjs() {}

In dieser Funktion wird Javascript Code definiert, welcher von YAF geladen wird. Hier muss zum Beispiel der Event-Handler implementiert werden, der aufgerufen wird, wenn ein Widget angeklickt wird. Im Beispiel von fs20easylamp ist dies die Funktion fs20easylamp_on_click(*view_id*, *widget_id*). Die Parameter *view_id* und *widget_id* dienen dazu, das angeklickte Widget in der Konfiguration zu finden und es somit eindeutig zu identifizieren. Die view_id in Verbindung mit der widget_id sind eindeutig.

Beispiel Click-Handler fs20easylamp:

Bei einem Klick auf eine Lampe des fs20easylamp Widgets, wird die Funktion fs20easylamp_on_click(view_id, widget_id) aufgerufen. Diese Funktion ruft über Ajax die Adresse /fhem/YAF/ajax/widget/fs20easylamp/set_lamp_status?view_id=1&widget_id=1&status=off. Hier werden also die Informationen *view_id*, *widget_id* und der Status, wie die Lampe geschaltet werden soll, übergeben. Anhand der übergebenen Informationen kann die Serverseite von fs20easylamp die Informationen aus der Config auslesen, welche gebraucht werden um letztendlich wirklich die Lampe zu schalten. Es wird also eine Verknüpfung zum wirklichen FS20 Attribut in der fhem.cfg hergestellt.

```
function fs20easylamp_on_click(view_id, widget_id) {
	var widget = $("#widget_"+view_id+"_"+widget_id);
	var newState;
	if (widget.hasClass("widget_fs20easylamp_on")) {
		newState = "off";
	} else{
		newState = "on";
	}
	$.ajax({
		type: "GET",
		async: true,
		url: "../../ajax/widget/fs20easylamp/set_lamp_status",
		data: "view_id="+view_id+"&widget_id="+widget_id+"&status="+newState,
		context: document.body,
		success: function(){
				fs20easylamp_update_widget(view_id, widget_id);
		}
	});
}
```

Beispiel Update-Handler fs20easylamp:
Ein Widget kann selbst entscheiden, was bei einem Update alles ausgeführt werden soll. Am Beispiel von fs20easylamp wird vom Server der Status einer bestimmten Lampe abgerufen.

```
function fs20easylamp_update_widget(view_id, widget_id) {
	$.ajax({
			type: "GET",
			async: true,
			url: "../../ajax/widget/fs20easylamp/get_lamp_status",
			data: "view_id="+view_id+"&widget_id="+widget_id,
			context: document.body,
			success: function(lamp_status) {
				var widget = $("#widget_"+view_id+"_"+widget_id);
				if (lamp_status == "off") {
					if (widget.hasClass("widget_fs20easylamp_on")) {
						widget.removeClass("widget_fs20easylamp_on");
					}
					if (!widget.hasClass("widget_fs20easylamp_off")) {
						widget.addClass("widget_fs20easylamp_off");
					}
				}
				else if (lamp_status == "on") {
					if (!widget.hasClass("widget_fs20easylamp_on")) {
						widget.addClass("widget_fs20easylamp_on");
					}
					if (widget.hasClass("widget_fs20easylamp_off")) {
						widget.removeClass("widget_fs20easylamp_off");
					}
				}
			}
	});
}
```

5.3 Auslieferung eines Widgets

Es ist gedacht, dass Widgets später als Dateiarchiv ausgeliefert werden, dass dann nur noch in FHEM/YAF/widgets/ entpackt werden muss. Nachdem ein neues Widget eingefügt wurde, muss FHEM neu gestartet werden, um das Widget zu aktivieren.

6 - Ideen für die Zukunft von YAF

Während der Entwicklung sind einige Ideen zu Veränderungen, Weiterentwicklungen und Verbesserungen der bereits vorhandenen Implementierung gekommen. Die Punkte sind nach ihrer Wichtigkeit geordnet.

Kontextmenü der Widgets
Das Kontextmenü zeigt noch nicht den Namen des jeweiligen Widgets und keine Bearbeitungsmöglichkeiten an.

Hintergrundbilder von Sichten über Einstellungen
Beim Erstellen einer neuen Sicht muss der Hintergrund bisher manuell über die Konfigurationsdatei eingetragen werden. Dies soll in Zukunft über den Dialog “Sicht erstellen” funktionieren.

Fehlermeldungen
Fehlermeldungen müssen einheitlich und verständlich ausgegeben werden. Bisher werden Fehler auf der Konsole ausgegeben. Diese sollen in der Zukunft in Logfiles geschrieben werden.

Code Refactoring
Bei der Entwicklung ist Code entstanden, welcher von Grund auf überarbeitet werden sollte. Vereinheitlichte Struktur, Variablennamen, Funktionsnamen, usw. sind wichtig für die Lesbarkeit und Wartbarkeit des Codes. Dies ist wichtig, um eine Weiterentwicklung zu ermöglichen.

Die Update Funktion zusammenfassen
Bisher ruft jedes Widget separat seine Update Funktion am Server auf. Deshalb kommt es je nach Einstellung des Refresh-Intervalls zu sehr vielen Anfragen in kurzer Zeit. Es wäre gut, wenn es hierfür nur einen Aufruf geben würde, der die Antworten zusammenfasst. Somit könnte man einiges an Traffic sparen und die Auslastung des Servers verringern.

Ajax Aufrufe sollen Wert enthalten, ob der Aufruf erfolgreich war
Aktuell wird bei einem Ajax Aufruf ein einzelner Wert zurückgegeben. An diesem Wert kann man leider nicht eindeutig erkennen, ob es beim Aufruf zu einem Fehler kam. Hierfür ist ein einheitlicher Response wünschenswert. Beispielsweise könnte anstatt einer Variable ein Array (mit JSON codiert) zurückgegeben werden. Dieses könnte neben den Daten noch eine Variable “success” enthalten, welche angibt, ob ein Aufruf erfolgreich war.

Größe anpassbar machen (aufwändig)
Durch verschiedene Auflösungen unterschiedlicher Geräte muss die Größe einstellbar sein.

Versuchen die benötigten CPAN Module einfacher zu installieren (möglich?)
Da es beim Installieren der CPAN Module häufig zu Problemen kommt, wäre es schön, wenn sich dies vereinfachen ließe. Vielleicht mit Hilfe des makefiles, das es bereits gibt.

7 - Fazit

Im Laufe des Semesters haben wir sehr viel über Hausautomatisierung und speziell FHEM lernen können, was bei uns definitiv ein Interesse an diesem Themengebiet geweckt hat. Speziell zum Projektablauf können wir sagen, dass es einiges an Zeit kostet, sich in FHEM einzuarbeiten, um anschließend eigene Erweiterungen entwickeln zu können. Mit der Zeit hatten wir öfters Probleme mit verschiedenen FHEM Versionen, welche in diesem Zeitraum veröffentlicht wurden. Dieser Umstand hat sich am Ende wesentlich verbessert. Was unserer Meinung nach bei der Entwicklung etwas zu kurz kam, war das Refactoring unserer Implementierung und das Entwickeln einiger kleinerer Funktionen. Dies war in der Zeit leider nicht mehr möglich. Beim Projektende handelt es sich allerdings um eine Frist für die Abgabe des Projekts an der Hochschule Karlsruhe - Technik und Wirtschaft. Wir sind durchaus in der Überlegung, an YAF weiterzuarbeiten und sind auf Feedback der Community gespannt.

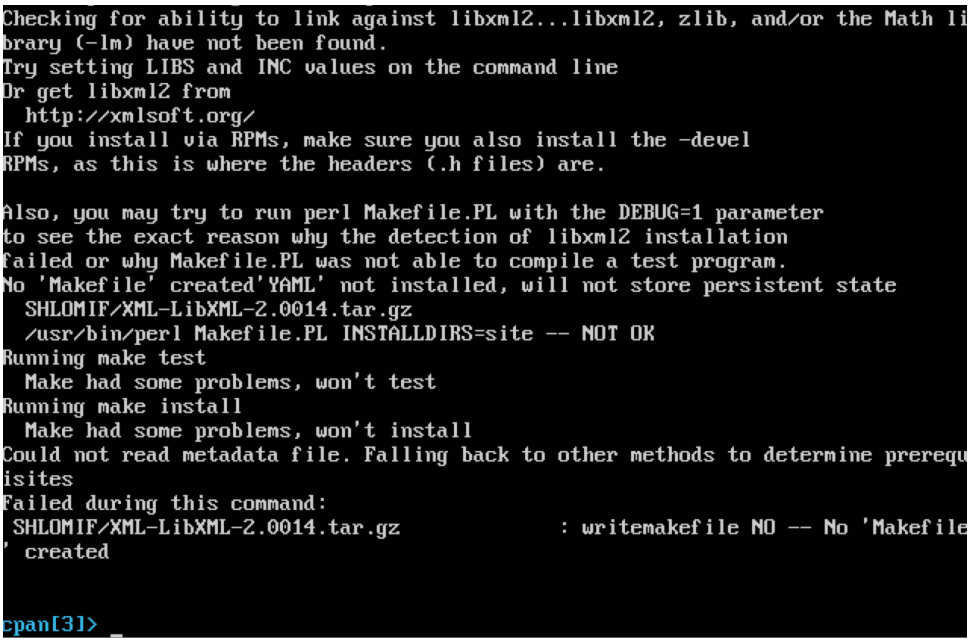
A - Perl Module manuell installieren

In Abschnitt 3.4 werden die notwendigen Voraussetzungen für den Einsatz von YAF geschaffen. Gibt es bei diesen Schritten Probleme, bieten sich die alternative Möglichkeiten an, um die Perl Module zu installieren.

A.1 XML::LibXML aus CPAN installieren

Die Installation über den Paketmanager in 3.4.2 ist stark zu empfehlen. Sollte das nicht möglich sein, kann das Modul auch so installiert werden.

```
$ sudo perl -MCPAN -e shell
cpan> install XML::LibXML
```



Der Vorgang wird wahrscheinlich mit einer Fehlermeldung abbrechen. Es ist nun notwendig, libxml2 manuell zu installieren:

```
$ wget ftp://xmlsoft.org/libxml2/libxml2-2.9.0.tar.gz
$ tar -zxf libxml2-2.9.0.tar.gz
$ cd libxml2-2.9.0
$ ./configure
$ make
$ sudo make install
```

Nun muss dem Makefile die libxml2 bekannt gemacht werden:

```
$ cd /home/user/.cpan/build/XML-LibXML-2.0014-xy/
```

Hinzufügen einer Zeile in die Makefile.PL

```
$ sudo vi Makefile.PL
      $config{LIBS} = '-L/usr/local/lib -L/usr/lib -lxml2 -lm';
$ perl Makefile.PL
$ make
$ sudo make install
```

Diese Vorgehensweise ist entnommen aus: http://williamjxj.wordpress.com/2011/05/28/perl-install-xmllibxml/

Außerdem sind noch Schwierigkeiten mit dem automatischen Auflösen der Abhängigkeiten beobachtet worden, sodass das Modul YAML nicht installiert werden konnte. Abhilfe schafft dieser Aufruf:

```
cpan> install YAML
```

A.2 Perl Module manuell installieren

Es ist außerdem möglich, die Module komplett manuell zu installieren. Diese Herangehensweise ist nur für den Fall gedacht, dass die Installation über CPAN nicht funktioniert.
Hierbei müssen die Module und v.a. ihre einzelnen Abhängigkeiten von Hand installiert werden, was ein sehr mühsamer Prozess ist.

Auf http://www.cpan.org/ kann man oben rechts in dem Suchfeld nach den Modulen suchen. Sucht man nach “JSON::XS”, so führt der erste Treffer auf die Seite http://search.cpan.org/~mlehmann/JSON-XS-2.33/XS.pm

Oben rechts findet sich dann der Downloadlink der .tar.gz des Moduls, in diesem Fall JSON-XS-2.33.tar.gz

Der Installationsvorgang für die Module ist immer gleich:

1. Entpacken des Modularchivs
   ```
   $ tar -zxf JSON-XS-2.33.tar.gz
   ```
2. In das Verzeichnis mit den entpackten Dateien wechseln
   ```
   $ cd JSON-XS-2.33
   ```
3. Datei mit Installationsinformationen erzeugen
   ```
   $ perl Makefile.PL
   ```
4. Dateien kompilieren
   ```
   $ make
   ```
5. Installationsvorbereitung testen
   ```
   $ make test
   ```
6. Modulinstallation durchführen
   ```
   $ sudo make install
   ```

Der Vorgang wird vermutlich nicht erfolgreich abschließen, da JSON::XS eine Modulabhängigkeit zu common-sense hat, was nicht automatisiert installiert werden kann. Die beschriebene Vorgehensweise beginnt also von vorne mit der Suche im Repository nach dem Modul common-sense, dem Download und Entpacken des Modularchivs, und der anschließenden Installation von common-sense. Anschließend kann erneut versucht werden, JSON::XS zu installieren, was erfolgreich klappen sollte.

Werden die Module von Hand installiert, so empfiehlt sich die folgende Reihenfolge:

1. `XML::LibXML` (siehe 3.4.2 oder 6.1.1)
2. `XML::LibXML::PrettyPrint`
   zwingende Abhängigkeiten
   - `Sub::Uplevel`
   - `Tree::DAG_Node`
   - `Test::Warn`
   - `Pragmatic`
3. `JSON::XS`
   zwingende Abhängigkeiten
   - `common::sense`