

# **Origo - social client**

## **Dokumentation**

**Mario Volke**

---

# **Origo - social client: Dokumentation**

Mario Volke

Copyright © 2008-2009 Mario Volke

---

---

---

# Inhaltsverzeichnis

Vorwort .....	v
1. Motivation & Vision .....	1
Motivation .....	1
Vision .....	1
2. Konzept .....	2
3. Das verteilte soziale Netzwerk .....	3
4. Der persönliche URI .....	5
5. Technische Umsetzung .....	7
6. Installation & Konfiguration .....	8
Vorraussetzungen .....	8
Installation .....	8
Konfiguration .....	8
Literaturverzeichnis .....	9
A. REST API Dokumentation .....	10
Authentifizierung .....	10
Fehlermeldungen .....	10
Allgemeine Methoden .....	10
Editor Methoden .....	10
Browser Methoden .....	12

---

# Vorwort



Diese Dokumentation entstand parallel zur Konzeptions- und Entwicklungsphase. Ziel dieses Dokuments ist es, dem Leser zum einen aufzuzeigen, welche Problemstellungen mit *Origo* gelöst werden können und zum anderen werden die Konzepte und Module von *Origo* vorgestellt, um die angesprochenen Probleme zu lösen.

---

# Kapitel 1. Motivation & Vision

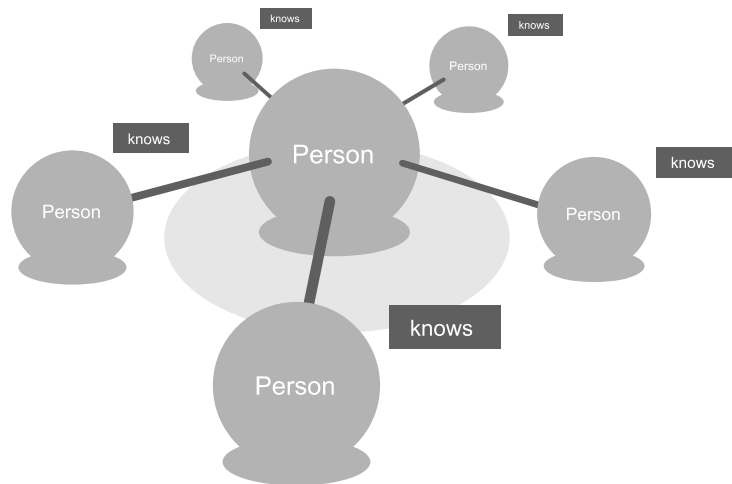
## Motivation

Seit den Anfängen des World Wide Web, über den Boom von Web 2.0, bis heute sind unzählige soziale Netzwerke mit unterschiedlich starken Nutzerzahlen entstanden und auch wieder verschwunden. Diese Netzwerke sind bis heute kaum untereinander verbunden. Ein User in Netzwerk A kann in der Regel keinen Kontakt zu einem User in Netzwerk B aufnehmen ohne sich mit einem neuen Account dort zu registrieren. Dadurch besteht die eigene Identität im Web meistens aus einer ganzen Fülle an unterschiedlichen Netzwerken, Benutzernamen, Passwörtern, etc.

## Vision

Das sog. *Semantic Web*, welches durch mehrere Sprachen und Standards vom W3C [<http://www.w3c.org>] definiert wird, stellt den nötigen Rahmen bereit, um offene Schnittstellen über Ontologien zu erstellen und damit die Datensilos heutiger sozialer Netzwerke aufzubrechen. Durch die *Friend of a Friend* (FOAF) Ontologie ist eine solche Schnittstelle um Personen im Semantic Web zu beschreiben bereits vorhanden. Mit dieser Ontologie ist es nicht nur möglich sich selbst über verschiedenste Metadaten zu beschreiben, zusätzlich lassen sich Verbindungen und Beziehungen zwischen Personen darstellen.

Die Daten werden in der Regel in RDF/XML serialisiert. Solche RDF-Dokumente, die FOAF-Daten enthalten, können von den verschiedenen sozialen Netzwerken veröffentlicht werden. Personen werden über URIs identifizierbar. Damit wird es möglich Personen über die bisherigen Grenzen sozialer Netzwerke hinweg miteinander zu verbinden. Es entsteht ein **semantisches, verteiltes, soziales Netzwerk**.



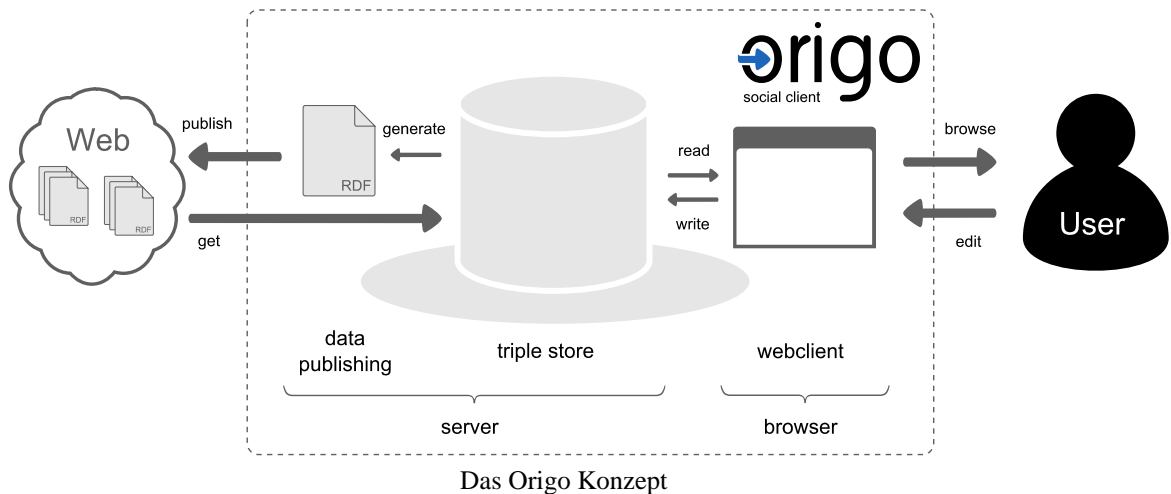
Ein verteiltes soziales Netzwerk

Damit ein User die vollständige Kontrolle über seinen persönlichen URI und seine veröffentlichten Daten behält, sollte er die RDF-Daten auf seinen eigenen Webspace oder Webserven laden. Die Verwaltung dieser Daten wird aber schnell unübersichtlich und die XML-Dokumente selbst zu schreiben kostet Zeit. *Orgio* kann auf dem eigenen Webserven selbst installiert und eingerichtet werden. Der *Orgio* Webclient schafft dann eine einfach und intuitiv zu bedienende Schnittstelle zum User. Dieser kann seine Daten dadurch auch ohne direkte Kenntnisse über die Technologien des Semantic Web selbst verwalten. Er macht sich nicht mehr wie bisher von einem Dienst abhängig. Auf bereits bestehende Profile in anderen sozialen Netzwerken kann mit Hilfe von *Orgio* verlinkt werden, sofern die Nutzerdaten auch semantisch verfügbar sind.

---

# Kapitel 2. Konzept

*Origo* soll dem Nutzer auf komfortable und flexible Weise erlauben seine Identität und sein soziales Netzwerk im semantischen Web zu überschauen und zu verwalten. Das Wort *Origo* kommt aus dem Lateinischen und bedeutet „*Ursprung*“. Der URI des Nutzers wird damit quasi zum Ursprung seines persönlichen Netzwerks im *Semantic Web*.



Origo erfüllt im wesentlichen drei Aufgaben. Das Profil des Nutzers zu veröffentlichen, dem Nutzer zu erlauben sein Profil zu bearbeiten und zuletzt sein soziales Netzwerk über den Browser zu betrachten.

Die Bereitstellung der semantischen Daten geschieht nach dem Ansatz *Linked Data*, wie vom W3C vorgeschlagen wird hierbei *Content Negotiation* eingesetzt, um je nach Anfrage auf verschiedene Datenformate weiterzuleiten [W3C08]. Origo unterstützt hierbei folgende Mime-Types:

- *text/html* Kann von Origo direkt generiert werden oder auf eine andere Website weiterleiten.
- *application/rdf+xml* Die Standard-Serialisierung von RDF-Daten.
- *application/turtle*
- *application/json*

Die ausgelieferten Daten werden gecached, dadurch können schnelle Zugriffszeiten realisiert werden.

Der Triple Store speichert sämtliche Profile, sowohl das des Nutzers selbst, als auch Profile, die vom Origo Browser geladen werden. Der Zugriff auf den Triple Store erfolgt mit Hilfe von SPARQL. Für die vom Browser geladenen Daten werden zusätzlich einfache Schlussfolgerungsmechanismen eingesetzt (*owl:inverseOf-Inferencing*, *rdfs:subPropertyOf-Inferencing*). Dadurch können implizit gegebene Informationen über den Browser explizit dargestellt werden.

Der Origo Webclient ist die Verbindung des Systems zum Nutzer. Er besteht aus einem Editor und einem Browser. Mit dem Editor kann der Nutzer seine persönlichen Daten verwalten, seinen persönlichen URI mit weiteren Profilen verknüpfen und Beziehungen zu anderen Personen erstellen. Der Browser erlaubt es dem Nutzer sein soziales Netzwerk auf intuitive Art und Weise zu durchstöbern.

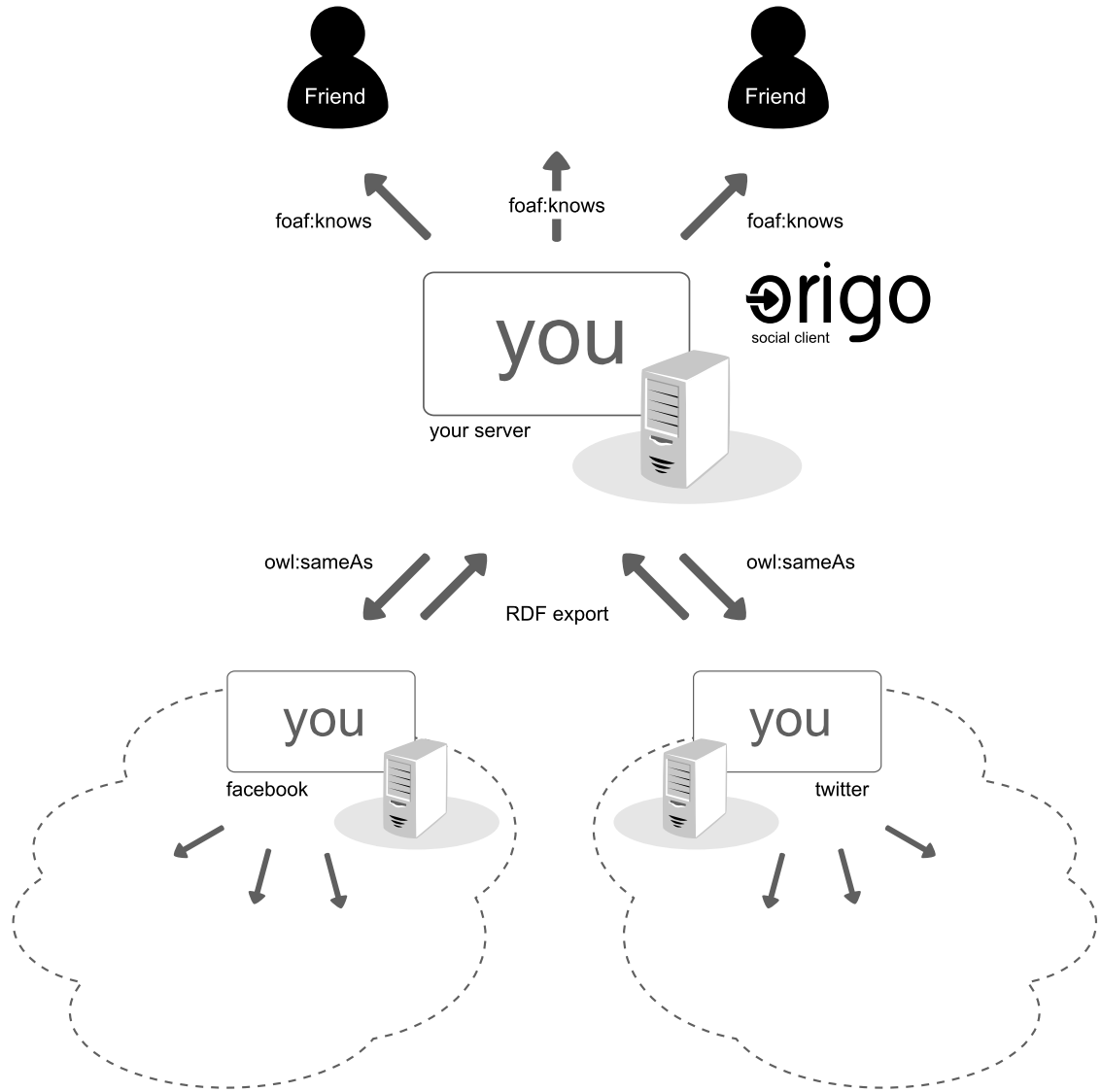
---

# Kapitel 3. Das verteilte soziale Netzwerk

Das Origo Packet wird auf dem eigenen Webserver oder Webservice installiert und verwaltet von da an die eigene Identität im Semantic Web. Über den Origo Client lassen sich Personen, die ebenfalls eine semantische Identität haben, als Freunde hinzufügen. Intern wird über *foaf:knows*, einer Eigenschaft aus der FOAF-Ontologie auf den URI dieser Person verlinkt. Eine solche Verlinkung lässt sich unabhängig davon erstellen, ob diese Person ebenfalls Origo einsetzt oder nicht.

Mit Hilfe der RELATIONSHIP-Ontologie ist es zudem möglich die Beziehungen zwischen zwei Personen genauer zu spezifizieren. In dieser Ontologie sind beispielsweise beziehungungen wie *rel:childOf* oder *rel:friendOf* definiert.

Häufig existieren bereits Profile in anderen in sich geschlossenen sozialen Netzwerken. Der Zugriff auf diese Netzwerke ist möglich, wenn ein RDF-Export der dort vorliegenden Daten möglich ist. Ein solcher Export kann sowohl von dem Betreiber dieses Netzwerkes selbst, als auch von einem externen Dienst, der über eine API mit dem Netzwerk kommuniziert, bereitgestellt werden. Wenn ein solcher Export vorhanden ist, dann kann Origo über die Eigenschaft *owl:sameAs* auf das externe Profil verlinken.



Origo im Zentrum des sozialen Netzwerks

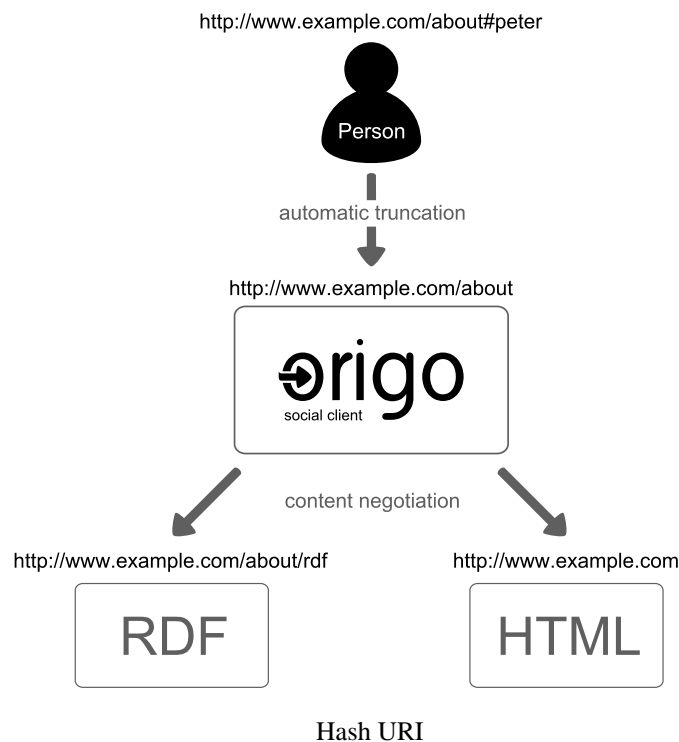
---

# Kapitel 4. Der persönliche URI

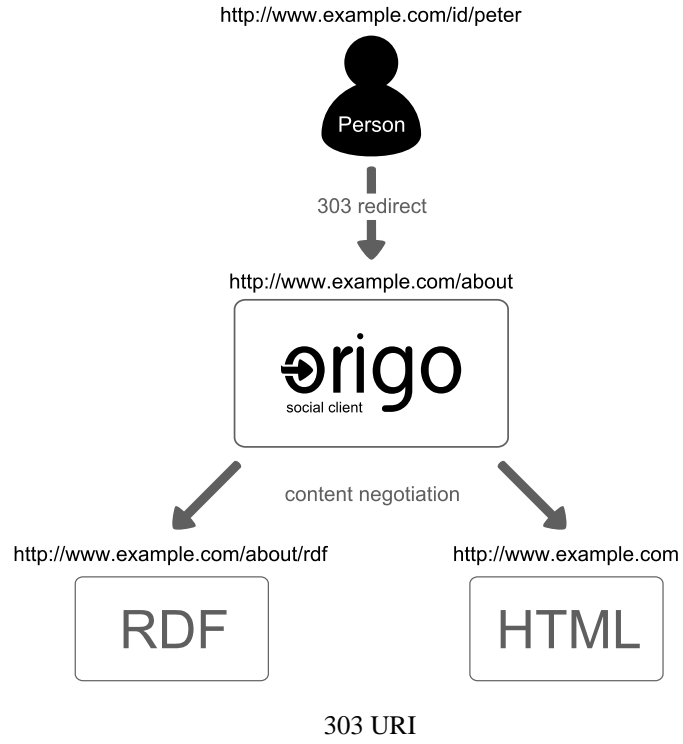
Der persönliche URI identifiziert den Nutzer von Origo und wird dadurch zum zentralen Element im Semantic Web. Origo bietet zwei verschiedene Möglichkeiten diesen URI zu konfigurieren.

Der sog. *Hash URI* wird vom Installer unterstützt und steht damit schnell und einfach zur Verfügung. Der vordere Teil der URI ergibt sich direkt aus dem Ort an dem Origo installiert wurde. Zusätzlich wird ein Hash-Wert angehängt, um den Nutzer zu identifizieren. Der Hash-Wert kann frei gewählt werden. Es empfiehlt sich aber die eigenen Initialen oder den Vornamen zu verwenden.

Beim Aufruf der URI im Browser wird der Hash-Wert automatisch abgeschnitten. Der Browser ruft also den Ort der Origo Installation auf. Dort greift der Content Negotiator ein und leitet die Anfrage entsprechend des angeforderten Inhalts entweder auf die RDF-Daten oder auf ein vorher konfiguriertes HTML-Dokument weiter.



Fortgeschrittenen Nutzern steht eine weitere Möglichkeit der Konfiguration des eigenen URI zur Verfügung. An einem beliebigen Ort, der sich auch unter einer anderen Domain befinden kann, wird eine 303-HTTP-Weiterleitung eingerichtet, die eine Anfrage auf den Ort der Origo Installation leitet. Dadurch kann der persönliche URI relativ beliebig gewählt werden. Nach der 303-Weiterleitung finden dann die selben Prozesse statt, wie beim Hash-URI.



---

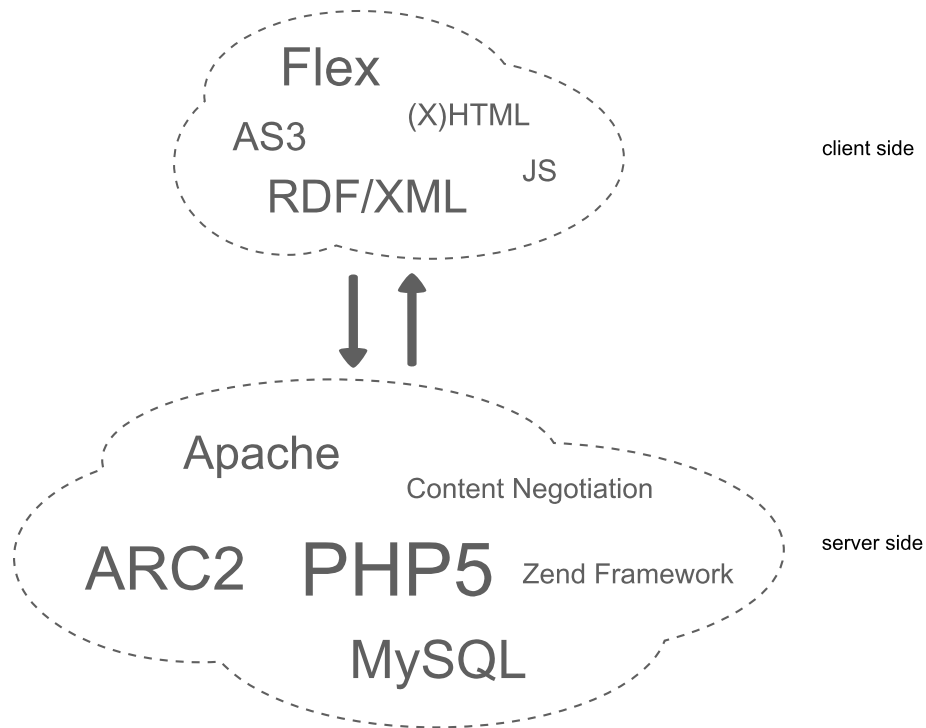
# Kapitel 5. Technische Umsetzung

Origo verwendet bevorzugt weit verbreitete Webtechnologien, um auch auf einfachen Webhosting-Paketten eingesetzt werden zu können. Origo ist vollständig web- und browserbasiert, d.h. es muss keine Software auf dem eigenen Computer installiert werden. Origo ist damit auf jedem Rechner sofort nutzbar. Das gesamte Packet besteht aus serverseitigen und clientseitigen Teilen.

Als serverseitiges Fundament dient ein typischer LAMP (Linux, Apache, MySQL, PHP) oder MAMP (Mac OS, Apache MySQL, PHP) Stack. ARC2 [<http://arc.semsol.org/>] ist eine Library für PHP5 und stellt die wesentlichen Features für die Arbeit mit semantischen Daten bereit. Dazu gehören unter anderem ein Triple Store und eine SPARQL-Schnittstelle. Eine *Content Negotiation*-Implementierung in PHP ist zentraler Bestandteil für die korrekte Weiterleitung des persönlichen URI anhand des HTTP-Headers. Zudem wird Zend Framework als Basis für die serverseitigen Aufgaben eingesetzt.

Auf Clientseite kommt neben den Standard-Sprachen *(X)HTML*, *JS*, *RDF/XML* eine *Flash/Flex*-Anwendung zum Einsatz. Dies schafft eine intuitive Bedienung des Origo Clients, wie es der Nutzer von Desktopanwendungen gewohnt ist. Da das Flash-Plugin bei über 95% aller Browser bereits installiert ist, sollte Origo in der Regel sofort einsetzbar sein.

Server und Webclient sind über eine REST-API miteinander verbunden, die es ermöglicht schreibend und lesend auf den Triple Store zuzugreifen. Die einzelnen Methoden der API sind im Anhang A beschrieben. Durch diese strikte Trennung von Server und Webclient ist es denkbar, dass auch andere Webclients (z.B. auf Basis von HTML/JS) für Origo entwickelt werden.



Technologien, Sprachen und Libraries in Origo

---

# Kapitel 6. Installation & Konfiguration

## Vorraussetzungen

- Apache + mod\_rewrite
- PHP >= 5.2.4
- MySQL

## Installation

- Den Ordner *src* löschen. Dieser wird für den Produktiveinsatz von Origo nicht benötigt.
- Die neueste Version des Zend Framework unter [framework.zend.com](http://framework.zend.com) [<http://framework.zend.com>] herunterladen.
- Den Ordner *library/Zend* aus dem Zend Framework nach *library/Zend* im Origo Verzeichnis kopieren.
- Dem *cache*-Ordner Schreibrechte geben.
- Eine MySQL-Datenbank erstellen.
- Die Datei *config/config.default.ini* nach *config/config.ini* kopieren.

## Konfiguration

Zuletzt muss die Konfigurationsdatei unter *config/config.ini* angepasst werden. Die zuvor erstellte MySQL-Datenbank muss hier eingetragen werden. In der Regel genügt es *username*, *password* und *dbname* anzugeben. Als nächstes muss der URI des Origo-Profiles angegeben werden. Unter *profile.location* sollte deshalb der URL zu Origo + *"/profile"* angegeben werden (z.B. *http://www.example.com/origo/profile*).

Wie im Kapitel zum persönlichen URI beschrieben, gibt es nun zwei Möglichkeiten diesen URI zu konstruieren. Den Hash-URI kann man verwenden, indem man unter *profile.identifizier* einen Hash einträgt (z.B. *#me*). Der persönliche URI ist dann *profile.location* + *profile.identifizier*. Die zweite Möglichkeit besteht darin unter einer beliebigen anderen Adresse eine 303-Weiterleitung zu *profile.location* einzurichten. Diese Adresse wird dann unter *profile.identifizier* eingetragen und wird damit zum persönlichen URI.

Unter *api.auth.username* und *api.auth.password* sollte für die Authentifizierung im Origo Webclient ein beliebiger Benutzername mit Passwort eingetragen werden.

Unter *negotiation.html* ist es möglich ein HTML-Profil (z.B. die eigene Website) anzugeben, auf die bei einer HTML-Anfrage des persönlichen URI automatisch weitergeleitet wird.

Die restlichen Konfigurationsmöglichkeiten sollten im Allgemeinen nicht verändert werden.

---

# Literaturverzeichnis

[AP08] *One FOAF fits all* [<http://apassant.net/blog/2008/01/12/one-foaf-fits-all/>]. Alexandre Passant. 12.01.2008.

[W3C08] *Cool URIs for the Semantic Web* [<http://www.w3.org/TR/2008/NOTE-cooluris-20080331/>]. W3C Interest Group. Leo Sauermann und Richard Cyganiak. 31.03.2008.

---

# Anhang A. REST API Dokumentation

## Authentifizierung

Alle Methoden, die Authentifizierung benötigen erwarten einen POST Parameter key mit dem Wert md5([username]:[password]).

## Fehlermeldungen

Falls in einer Methode ein Fehler auftritt, so wird eine Fehlermeldung mit folgender Formatierung ausgegeben:

```
<error><error_code>[maschinenlesbarer Fehlercode]</error_code><error_message>[Fehlernachricht]</error_message></error>
```

## Allgemeine Methoden

- **/api/test**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: -

Beschreibung: Hilfsmethode zum Testen der Login-Daten.

Return: `<request><code>200</code><message>Api status: OK</message></request>` im Erfolgsfall.

## Editor Methoden

- **/api/editor/get**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: -

Beschreibung: Liefert das Profil mit allen einfach vorkommenden Eigenschaften (title, nick, homepage, mbox, mbox\_shalsum, img, depiction, family\_name, givenname, name, weblog, workinfohomepage, workplacehomepage, schoolhomepage, plan, geekcode, gender, myersbriggs, openid, icq, msn, aim, yahoo, jabber)

Return: Das aktuell gespeicherte Profil mit allen einfach vorkommenden Eigenschaften.

- **/api/editor/update**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *title, nick, homepage, mbox, mbox\_sha1sum, img, depiction, family\_name, givenname, name, weblog, workinfohomepage, workplacehomepage, schoolhomepage, plan, geekcode, gender, myersbriggs, openid, icq, msn, aim, yahoo, jabber* (Alle optional)

Beschreibung: Aktualisiere oder erzeuge einfach vorkommende Eigenschaften.

Return: Das aktualisierte Profil mit allen einfach vorkommenden Eigenschaften.

- **/api/editor/delete**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *title, nick, homepage, mbox, mbox\_sha1sum, img, depiction, family\_name, givenname, name, weblog, workinfohomepage, workplacehomepage, schoolhomepage, plan, geekcode, gender, myersbriggs, openid, icq, msn, aim, yahoo, jabber* (Alle optional)

Beschreibung: Lösche die über POST Parameter gegebenen Eigenschaften.

Return: Das aktualisierte Profil mit allen einfach vorkommenden Eigenschaften.

- **/api/editor/clean**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: -

Beschreibung: Löscht das gesamte Profil mit allen Eigenschaften und Beziehungen.

Return: *<result>1</result>* im Erfolgsfall.

- **/api/editor/profiles/get**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: -

Beschreibung: Liefert die Verknüpfungen zu externen Profilen.

Return: Alle vorhandenen Verknüpfungen zu externen Profilen.

- **/api/editor/profiles/update**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *sameas, seealso* (Benötigt) *label* (Optional)

Beschreibung: Verknüpft das Profil mit einem externen Profil. Bei gleichbleibendem *sameas* kann die Verknüpfung auch aktualisiert werden.

Return: *Alle vorhandenen Verknüpfungen zu externen Profilen.*

- **/api/editor/profiles/delete**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *sameas* (Benötigt)

Beschreibung: Löscht die Verknüpfung zu einem externen Profil.

Return: *Alle vorhandenen Verknüpfungen zu externen Profilen.*

- **/api/editor/relationships/get**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: -

Beschreibung: Liefert die Beziehungen zu anderen Personen.

Return: *Alle vorhandenen Beziehungen zu anderen Personen.*

- **/api/editor/relationships/update**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *to* (Benötigt) *acquaintanceof, ambivalentof, ancestorof, antagonistof, apprenticeto, childof, closefriendof, collaborateswith, colleagueof, descendantof, employedby, employerof, enemyof, engagedto, friendof, grandchildof, grandparentof, hasmet, knowsbyreputation, knowsinpassing, knowsof, lifepartnerof, liveswith, lostcontactwith, mentorof, neighborof, parentof, participant, participantin, siblingof, spouseof, workswith, wouldliketoknow* (Optional)

Beschreibung: Fügt eine neue Beziehung zu *to* hinzu. Mit den zusätzlichen Parametern kann die Beziehung näher spezifiziert werden. *to* kann entweder die dereferenzierbare URI der Person sein oder die URI des foaf:PersonalProfileDocument mit vorhandenem foaf:primaryTopic. Die Methode lädt zudem grundlegende Eigenschaften wie name, nick, image, etc. in das eigene Profile, um beim Browsen schneller Informationen anzeigen zu können. Eine Aktualisierung bei gleichbleibendem *to* ist möglich.

Return: *Alle vorhandenen Beziehungen zu anderen Personen.*

- **/api/editor/relationships/delete**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *to* (Benötigt)

Beschreibung: Löscht die Beziehung zu einer Person.

Return: *Alle vorhandenen Beziehungen zu anderen Personen.*

# Browser Methoden

- **/api/browser/profile**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *uri* (Benötigt)

Beschreibung: Liefert die Eigenschaften einer Person. *uri* kann entweder die dereferenzierbare URI der Person sein oder die URI des foaf:PersonalProfileDocument mit vorhandenem foaf:primaryTopic.

Return: Die einfach vorkommenden Eigenschaften dieser Person.

- **/api/browser/relationships**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *uri* (Benötigt) *acquaintanceof*, *ambivalentof*, *ancestorof*, *antagonistof*, *apprenticeto*, *childof*, *closefriendof*, *collaborateswith*, *colleagueof*, *descendantof*, *employedby*, *employerof*, *enemyof*, *engagedto*, *friendof*, *grandchildof*, *grandparentof*, *hasmet*, *knowsbyreputation*, *knowsinpassing*, *knowsof*, *lifepartnerof*, *liveswith*, *lostcontactwith*, *mentorof*, *neighborof*, *parentof*, *participant*, *participantin*, *siblingof*, *spouseof*, *workswith*, *wouldliketoknow* (Optional)

Beschreibung: Liefert die Beziehungen einer Person. Mit dem zusätzlichen Parametern können die Beziehungen speziell gefiltert werden, ansonsten werden alle zurückgeliefert. *uri* kann entweder die dereferenzierbare URI der Person sein oder die URI des foaf:PersonalProfileDocument mit vorhandenem foaf:primaryTopic.

Return: Die Beziehungen dieser Person.

- **/api/browser/clean**

Methode: *POST*

Authentifizierung: *Ja*

Parameter: *uri* (Optional)

Beschreibung: Löscht alle für den Browser gespeicherten Daten oder falls *uri* vorhanden nur die mit *uri* in Verbindung stehenden Daten.

Return: *<result>1</result>* im Erfolgsfall.