Origo - social client

Dokumentation

Mario Volke

Origo - social client: Dokumentation

Mario Volke

Inhaltsverzeichnis

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

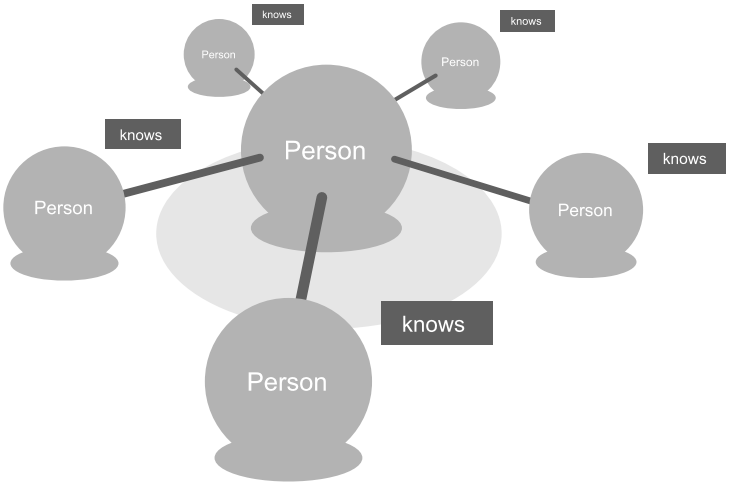
Motivation

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 veröffentlich 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**.

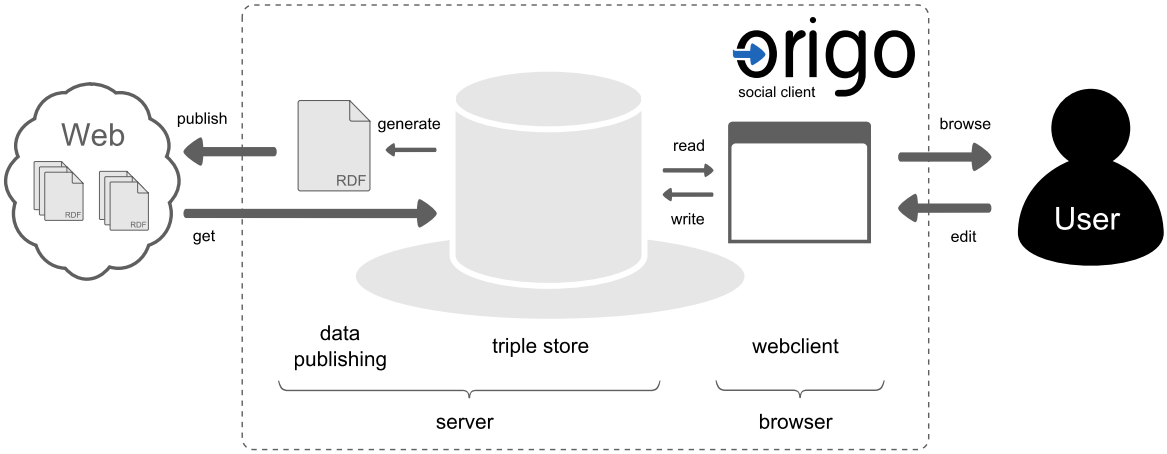


Ein verteiltes soziales Netzwerk

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 Webserver laden. Die Verwaltung dieser Daten wird aber schnell unübersichtlich und die XML-Dokumente selbst zu schreiben kostet Zeit. *Orgio* kann auf dem eigenen Webserver selbst installiert und eingerichtet werden. Der Origo 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 Origo 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 Bedeuted „*Ursprung*“. Der URI des Nutzers wird damit quasi zum Ursprung seines persönlichen Netzwerks im *Semantic Web*.



Das Origo Konzept

Das Origo Paket besteht aus mehreren Teilen, die teilweise optional zu verwenden sind:

- **Content Negotiator**

  Der Content Negotiator leitet eine HTTP-Anfrage abhängig vom HTTP-Header entweder auf das RDF-Dokument oder auf ein festgelegtes HTML-Dokument weiter.

  Pfad: *origo/*

- **RDF Distributor**

  Der RDF Distributor generiert die RDF-Daten aus dem sog. Triple Store und liefert diese über einen Caching-Mechanismus aus.

  Pfad: *origo/rdf*

- **Origo Client**

  Das Hauptmodul von Origo ist der Client. Er besteht wiederum aus zwei Modulen, die allerdings nur logisch voneinander getrennt sind:

  - **Origo Editor**

    Mit dem Origo Editor kann der Nutzer seine persönlichen Daten verwalten und seinen persönliche URI mit weiteren Profilen verknüpfen.

  - **Origo Browser**

    Der Origo Browser erlaubt es dem Nutzer sein soziales Netzwerk zu durchstöbern und neue Freunde hinzuzufügen.

  Der Client bietet verschiedene Authentifizierungsmöglichkeiten und nutzt einen SPARQL-Endpoint,

um auf den Triple Store zuzugreifen.

Pfad: *origo/client/*

- **Konfiguration**

  Die Konfiguration von Origo wird über eine zentrale INI-Datei vorgenommen. Der Negotiator wird separat konfiguriert, da sich dieser auch auf einem anderen Server befinden kann.

  Pfad: *origo/config/*

- **Installer** (optional)

  Die Konfiguration von Origo kann bequem über den Installer vorgenommen werden. Dieser schreibt die benötigten Konfigurationsdateien und unterstützt den User darin, die Zugriffsberechtigungen auf die verschiedenen Ordner richtig zu setzen.

  Pfad: *origo/install/*

- **Identifier** (optional)

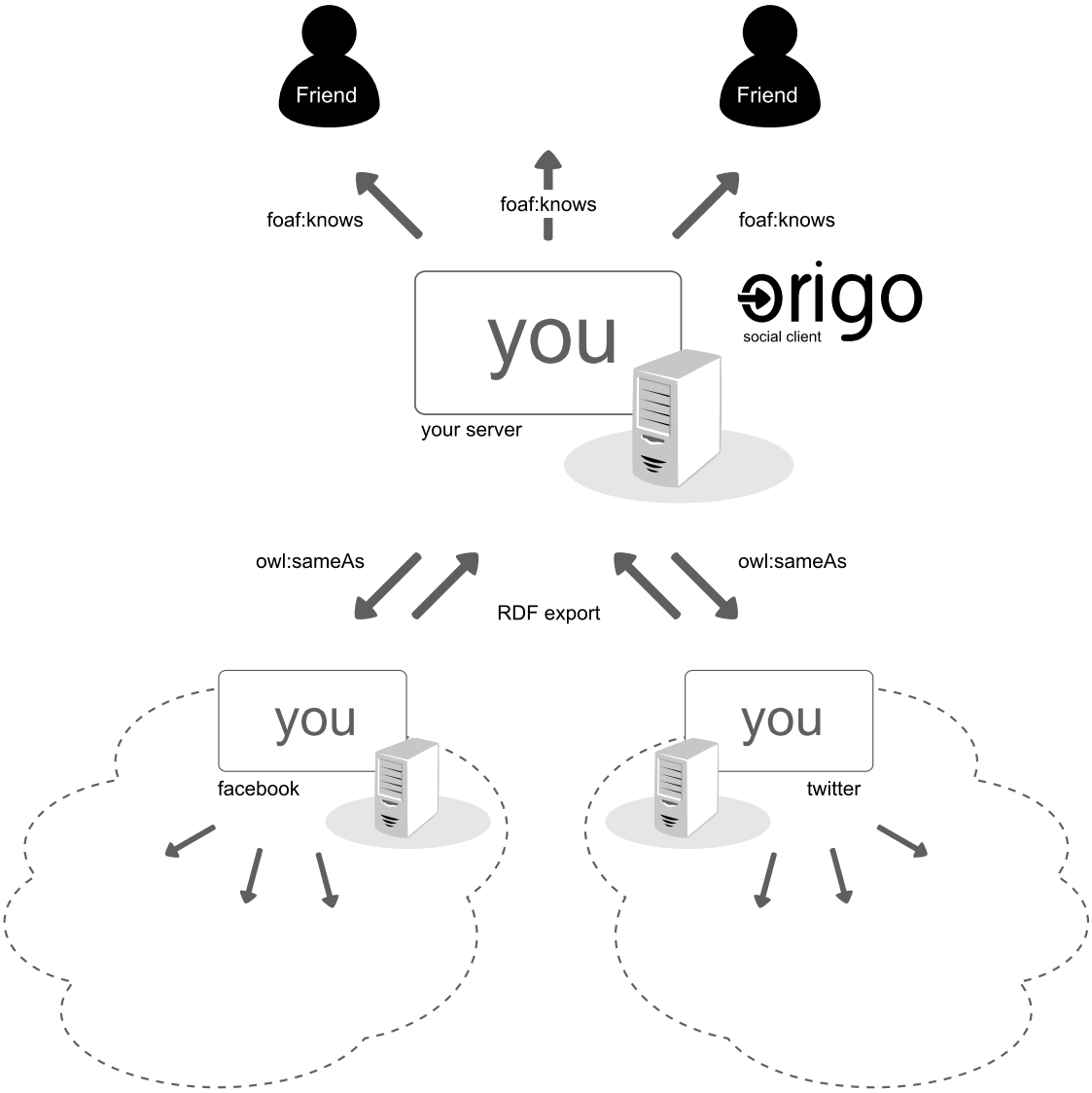
  Der Identifier ist eigentlich kein richtiges Modul. Es ist vielmehr eine Vorgehensweise, um über eine 303 Weiterleitung einen beliebigen URI, der sich auch auf einer anderen Domain befinden kann, quasi vorzuschalten. Ein Vorteil dieser Methode ist, dass der persönliche URI beliebig gewählt werden kann und dieser dadurch kein Hash-URI mehr sein muss. Diese Möglichkeit ist nur für fortgeschrittene Nutzer geeignet.

  Pfad: *origo/optional/identifier/*

Kapitel 3. Das verteilte soziale Netzwerk

Das Origo Packet wird auf dem eigenen Webserver oder Webspace 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.

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.



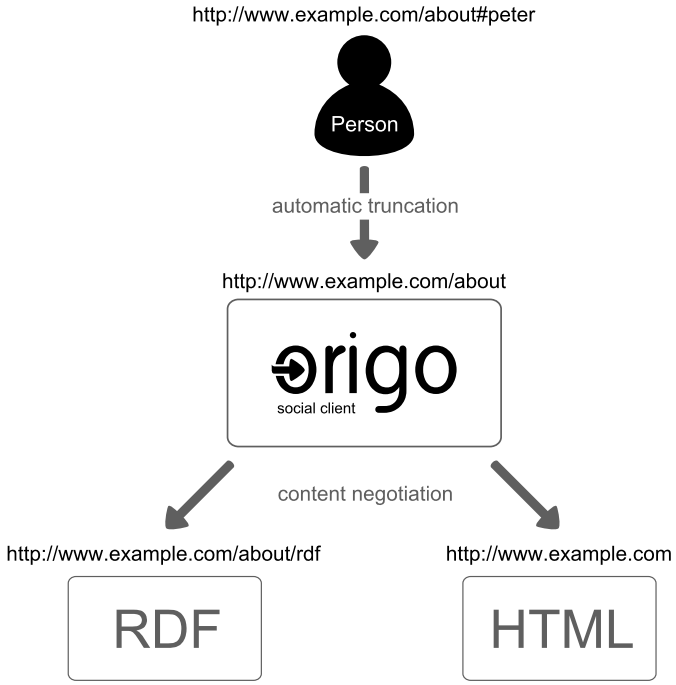
Origo im Zentrum des sozialen Netzwerks

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.

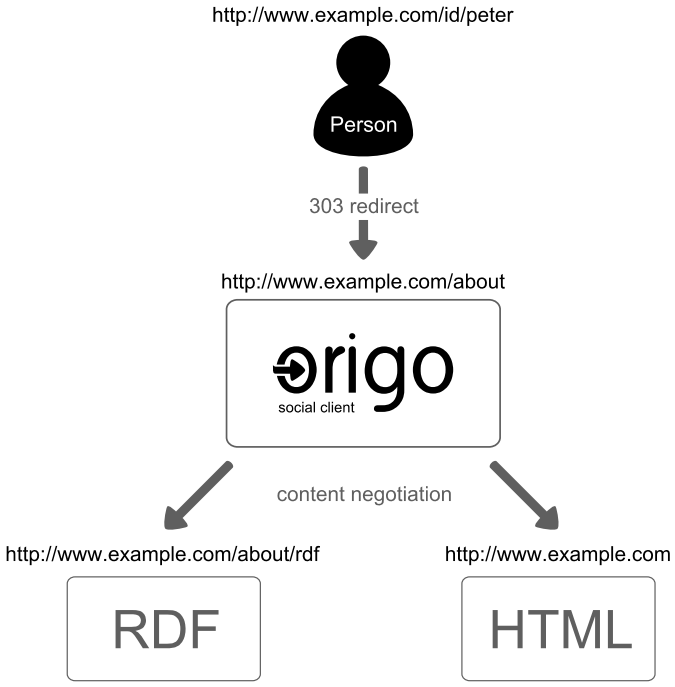
Der sog. *Hash URI* wird vom Installer unterstützt und steht damit schnell und einfach zur Verfügung. Der vordere Teil der URI ergibt sich direkt aus dem Ort an dem Origo installiert wurde. Zusätzlich wird ein Hash-Wert angehängt, um den Nutzer zu identifizieren. Der Hash-Wert kann frei gewählt werden. 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 Negotiatior ein und leitet die Anfrage entsprechend des angeforderten Inhalts entweder auf die RDF-Daten oder auf ein vorher konfiguriertes HTML-Dokument weiter.



Hash URI

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.


http://www.example.com/id/peter
Person
303 redirect
http://www.example.com/about
origo
social client
content negotiation
http://www.example.com/about/rdf
http://www.example.com
RDF
HTML

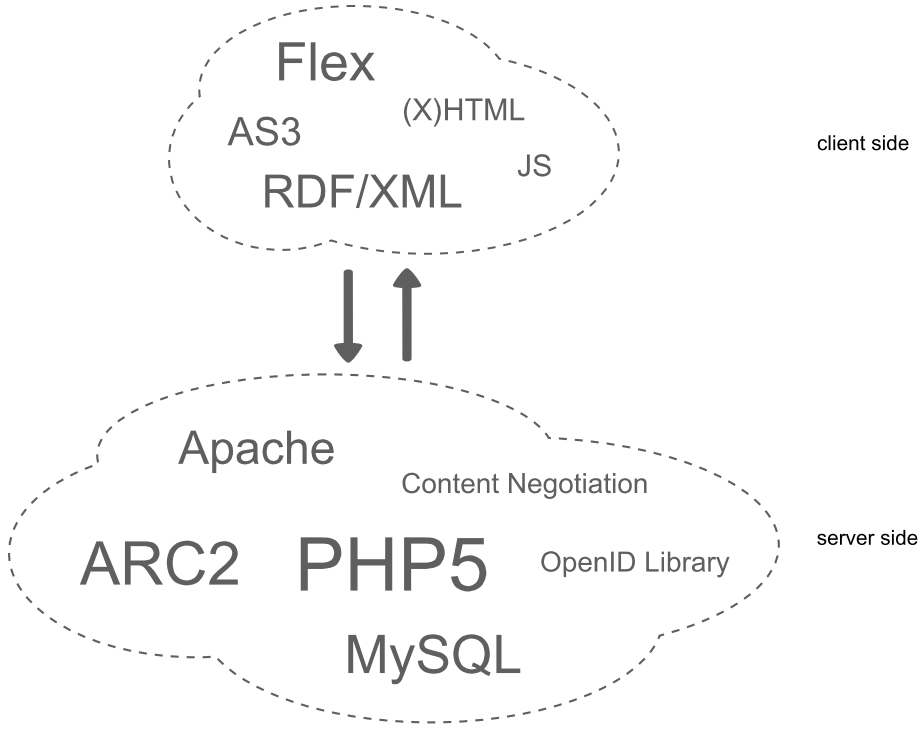
303 URI

Kapitel 5. Technische Umsetzung

Origo verwendet bevorzugt weit verbreitete Technologien, um auch auf einfachen Webhosting-Packeten 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 System 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 das Semantic Web bereit. Eine *Content Negotiation* Implementierung in PHP ist zentraler Bestandteil für die korrekte Weiterleitung des persönlichen URI anhand des HTTP-Headers. Optional wird für die Authentifizierung eine PHP OpenID Library [http://openidenabled.com/php-openid/] 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.



Technologien, Sprachen und Libraries in Origo

Kapitel 6. Installation & Konfiguration

[Inhalt]

Kapitel 7. Quellen

[Inhalt]

Kapitel 8. Anhang: 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_sha1sum, 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.