Mitzi László

email@mitzilaszlo.org | +31 (0) 611412377

**Solid Specifications**

A complete description of Solid for both non-engineers and engineers

Table of Contents

[Overview 2](#_Toc2255835)

[Solid Values 3](#_Toc2255836)

[Solid Definitions 4](#_Toc2255837)

[Web Identification 8](#_Toc2255838)

[Ethics 8](#_Toc2255839)

[Design 8](#_Toc2255840)

[Technical 8](#_Toc2255841)

[Legal 16](#_Toc2255842)

[Personal Online Data Storage 17](#_Toc2255843)

[Ethics 17](#_Toc2255844)

[Design 18](#_Toc2255845)

[Technical 18](#_Toc2255846)

[Legal 24](#_Toc2255847)

[Data Sharing Preferences 25](#_Toc2255848)

[Ethics 25](#_Toc2255849)

[Design 25](#_Toc2255850)

[Technical 25](#_Toc2255851)

[Legal 36](#_Toc2255852)

[Finding Solid Apps 37](#_Toc2255853)

[Ethics 37](#_Toc2255854)

[Design 37](#_Toc2255855)

[Technical 37](#_Toc2255856)

[Legal 38](#_Toc2255857)

[Finding Contacts 39](#_Toc2255858)

[Ethics 39](#_Toc2255859)

[Design 39](#_Toc2255860)

[Technical 39](#_Toc2255861)

[Legal 44](#_Toc2255862)

# Overview

Solid (derived from "**so**cial **li**nked **d**ata") is a specification detailing how to decentralise power on the web to ensure that the web is used for the global public interest.

The Solid specifications are designed for engineers, lawyers, designers, and ethicists to be able to implement Solid. Solid specifications define how to identify persons and verify that they are the person they claim to be, collect and store the personal data describing a person in a single point, and use data for specific applications while protecting the person from unnecessary harmful side effects. The Solid specification is written in the chronological order of the user stories.

* As a person I want to have a recognisable unique identity so that I can communicate to others who I am and what belongs to me.
* As a parent or guardian, I want to set up a recognizable unity identity for my child so that they can communicate to others who they are and what belongs to them.
* As a person I want to have a place online to collect and store information about myself so that I can present myself to others online.
* As a parent or guardian, I I want to have a place online to collect and store information about my child so that my child can present themselves to others online
* As a person I want to understand the implications of my data sharing preferences and choose a fixed data sharing preference that will operate on all applications until I choose to change those data sharing preferences.
* As a parent or guardian, I want to maximise the protection of my child until they reach the age of consent while still giving them access to health and education.
* As a person I want to search for applications to use information about me so that I can get stuff done efficiently to my own benefit
* As a person I want to search for the up to date contact details of people I know.
* As a person I want to share my up to date contact details with people I know.

The elements of Solid described above need to be held under a governance structure to uphold the core values and provide a source of truth.

There are other elements of Solid that can be carried out by Solid Providers. The Solid specifications are intended to provide guidance to Solid Providers who build elements of Solid and collaborate together with other Solid Providers.

Solid is modular and extensible and builds on top of existing [W3C](http://www.w3.org/) standards and protocols.

For a complete definition of terms look at the end of this document.

## Solid Values

* **Self Determination**: People and legal entities should be given the tools for self-determination. Self-determination involves the app user controlling data generated as a result of using an application, rather than the developer or owner of the application. Data control includes deciding where the data is stored as well as who sees what data, for what purpose, when. No individual should make a decision on behalf of another individual without consultation and ensuring that the individually is equipped with the necessary information to make an informed decision on the matter at hand that does not take advantage of convenience nudging.
* **Interoperability**: Data should be interoperable to ensure that individuals or legal entities are able to self-determine independently of the technical possibilities. Technical hurdles should not be used as an excuse to not provide interoperability.
* **Equal Access**: There should be equal access to the web by all individuals both for consumption and creation. Although, for example, development, may take education, this education should be freely available and accessible by all.
* **Sourcing**: It should be easy to find the source of a piece of data so that one can judge the trustworthiness of the data.
* **Public Value**: The web should serve humanity to become more functional through collaboration leading toward mutual benefit. The long term public interest should be given priority over the short term private interest.
* **Linking**: An individual or legal entity should be able to connect data they control to the data others control (what is a religious institution or government body insists on including a person as a member of a group that means they could be persecuted? Could this conflict with self-determination? Right to be forgotten? Right to disconnect?). The Semantic Web isn't just about putting data on the web. It is about making links, so that a person or machine can explore the web of data.  With linked data, when you have some of it, you can find other, related, data. Like the web of hypertext, the web of data is constructed with documents on the web. However, unlike the web of hypertext, where links are relationships anchors in hypertext documents written in HTML, for data they links  between arbitrary things described by RDF,.  The URIs identify any kind of object or concept.   But for HTML or RDF, the same expectations apply to make the web grow:

## Solid Definitions

[**Semantic Web**](https://www.w3.org/standards/semanticweb/)

The Semantic Web is an extension of the World Wide Web through standards by the World Wide Web Consortium (W3C). The standards promote common data formats and exchange protocols on the Web, most fundamentally the [Resource Description Framework](https://en.wikipedia.org/wiki/Resource_Description_Framework) (RDF). According to the W3C, "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries". The Semantic Web is therefore regarded as an integrator across different content, information applications and systems.

[**Resource Description Framework (RDF)**](https://www.w3.org/RDF/)

A generic graph-based data model with which to structure and link data on the web. The data model describes Things, including their relationships with other Things. RDF can be written in a number of ways known as ‘serialisations’ an example of which is Turtle. rdf-schema

The Solid spec uses the [RDF Schema](https://www.w3.org/TR/rdf-schema/) vocab for specifying [Extended Profiles](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#extended-profile), via rdfs:seeAlso links from the WebID Profile document.

Canonical prefix URL: http://www.w3.org/2000/01/rdf-schema#

[**Turtle**](https://www.w3.org/TR/2014/REC-turtle-20140225/)

Turtle derived from Turse RDF Triple Language is a structural syntax for RDF. “Turtle allows RDF graph to be completely written in a compact and natural text form, with abbreviations for common usage patterns and datatypes.” Provides compatibility with the triple pattern syntax of the SPARQL. All Turtle files have the extension “.ttl”.

[**RDFS (RDF Schema)**](https://www.w3.org/TR/2014/REC-rdf-schema-20140225/)

An extension of the basic RDF data modelling vocabulary

**Graph Database**

The data storage model used by the semantic web. An RDF graph can be visualized as a node and directed-arc diagram, in which each triple is represented as a node-arc-node link. RDF graphs are set(s) of “subject - predicate - object” triples that are used to express descriptions of resources. The Turtle language is used to express these triples.

**Web Ontology Language (**[**OWL)**](https://www.w3.org/2001/sw/wiki/OWL)

A Semantic Web language that builds on RDF

[**SPARQL**](https://www.w3.org/TR/sparql11-query/)

The query language for RDF

[**WebID**](https://www.w3.org/wiki/WebID)

A unique way to identify an entity such as an individual, company, organisation, or institutions. The WebID allows for an entity to be references using a Uniform Resource Identifier (URI) when their profile is hosted on a different web server than their own.

**Public Key WebID**

A URI that refers to a person that uniquely identifies a user to their public key

**Thing(s)**

A piece of data, file, or collection of files that can be manipulated, shared, linked to additional things and data, or people via codified known ontologies.

**File(s)**

A piece of data of some type, that is stored on a pod server. This file has a url endpoint.

**Node**

May refer to two things: aka Node.js, a JavaScript runtime for things not running in a browser, typically running the server. Or a RDF node, i.e. one thing or string in an RDF graph.

**Subject**

Source of the link

**Predicate**

The link type

**Object**

Target of the link

**Pod (Personal Online Datastore)**

A secure repository containing the user’s data. Users control application access to their POD(s).

**POD Manager**

An application that provides a UI and UX that allows the user to manage their Things, Files, People, and Applications. This management includes granting permission.

**Homepage**

A Solid users' exposed, customizable, profile

**Access Control List File (ACL)**

Defines the properties associated with the level of access of a user. Comprised of classes such as ‘Agent’.

[**WebAccessControl**](https://www.w3.org/wiki/WebAccessControl#Resources_covered_by_the_ACL)

A decentralized system for allowing different users and groups of users, forms of access to resources. Users are identified by WebID and groups of users are identified by HTTP URIs. auth/acl

Used for Authorization/[Web Access Control](https://github.com/solid/solid-spec#web-access-control) of Solid resources.

Canonical prefix URL: http://www.w3.org/ns/auth/acl#

**Agent**

A class of Access Control List that allows anyone and anything to access

solid/app

Contains a snapshot of the previous App Configuration vocabulary for Solid apps. Warning: do not use, the app configuration spec is currently in flux.

Source code: [solid-app.ttl](https://github.com/solid/vocab/blob/master/solid-app.ttl)

solid/terms

Contains core Solid terms, such as links from a [Solid WebID Profile](https://github.com/solid/solid-spec#webid-profile-documents) to user account resources, [WAC](https://github.com/solid/solid-spec#web-access-control) related predicates, and so on.

Canonical prefix URL: http://www.w3.org/ns/solid/terms#

Source code: [solid-terms.ttl](https://github.com/solid/vocab/blob/master/solid-terms.ttl)

vcard

The [vCard Ontology](https://www.w3.org/TR/vcard-rdf/) is recommended for Contact Management like applications for use with the Solid ecosystem.

Canonical prefix URL: http://www.w3.org/2006/vcard/ns#

Solid Extensions:

AddressBook - an RDF class for representing address books that contain contacts.

auth/cert

Used in the Public Key Certificates portion of the Solid WebID Profile document.

Canonical prefix URL: http://www.w3.org/ns/auth/cert#

dc/terms

Solid uses the [Dublin Core Terms](http://dublincore.org/documents/dcmi-terms/) vocabulary in several apps, but specifically in the Certificates section of the WebID Profile document.

Canonical prefix URL: http://purl.org/dc/terms/

foaf

The [FOAF](http://xmlns.com/foaf/spec/) ontology is used in [WebID Profile](https://github.com/solid/solid-spec#webid-profile-documents) documents and [Web Access Control](https://github.com/solid/solid-spec#web-access-control) .acl resources.

Canonical prefix URL: http://xmlns.com/foaf/0.1/

ldp

Describes [Linked Data Platform](https://www.w3.org/TR/ldp/) terms. Used in the [Content Representation](https://github.com/solid/solid-spec#content-representation) and [REST API](https://github.com/solid/solid-spec#https-rest-api) sections of the Solid spec.

Canonical prefix URL: http://www.w3.org/ns/ldp#

owl

The Solid spec uses the [Web Ontology Language](https://www.w3.org/standards/techs/owl#w3c_all) vocab for specifying [Extended Profiles](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#extended-profile), via owl:sameAs links from the WebID Profile document.

Canonical prefix URL: http://www.w3.org/2002/07/owl#

pim/space

Used in the [Solid WebID Profile](https://github.com/solid/solid-spec#webid-profile-documents) spec for discovery of user account resources (to link to Root Storage and Preferences resources).

Canonical prefix URL: http://www.w3.org/ns/pim/space#

posix/stat

Describes terms for POSIX-like files and directory listings. Used in the [Solid REST API](https://github.com/solid/solid-spec#https-rest-api) when listing Containers.

Canonical prefix URL: http://www.w3.org/ns/posix/stat#

Source code: [posix-stat.rdf](https://github.com/solid/vocab/blob/master/posix-stat.rdf)

# Web Identification

|  |
| --- |
| As a person I want to have a recognisable unique identity so that I can communicate to others who I am and what belongs to me. As a parent or guardian, I want to set up a recognizable unity identity for my child so that they can communicate to others who they are and what belongs to them. |

## Ethics

Why do persons need to be identified online? Self-determination is For self-determination the delineation of self needs to be defined. One needs to know who is who before being able to say what belongs to who. Having a WebID means that each person is able to control their identity and place that identity into a web of relationships that reflect relationships offline. Self-identification should be an independent activity to ensure that one person cannot assume the identity of another.

## Design

Go onto a website and apply for a WebID

Postal verification?

Need for face to face?

## Technical

Legal persons, meaning individuals and legal entities such as companies, have identity numbers when they use Solid called a <WebID>. WebIDs provide globally unique decentralized identifiers, enable cross-service federated signin, prevent service provider lock-in.

How are WebIDs formed. A WebID is formed from a uniform resource identifiers (URIs). These URIs form the basis of most other Solid-related technologies, such as authentication, authorization, access control, user profiles, discovery of user preferences and server capabilities, and more.

An example of a WebID is https://alice.databox.com/profile/card#me or <http://somepersonalsite.com/#webid>

The structure of an identification is important to ensure that no personal data is inferred unwillingly. For example, the order and structure of letters and numbers in a passport number can reveal the nationality of a person. A name can reveal the culture of a person. Therefore, the default structure of a WebID is: Number number letter number letter letter number number letter number letter letter number number letter number letter letter.

There are x variations of this structure and you can randomly generate a new WebID and check if it is available with the WebID generator.

How to ensure that a WebID is not depending on the extension dependencies?

What makes a WebID legitimate? Authentication is the process of determining a user’s identity, of asking the question “How do I know you are who you say?”.

How do web applications typically authenticate users (that is, how do they verify identity)? The most common method is usernames and passwords. A *username* uniquely identifies a user (and ties them to a user profile), and a *password* verifies that the user is who they say they are. Many applications or services also have a *secondary authentication mechanism*(usually an external email address) that they use for account recovery (in case the user forgets or loses their primary authentication tokens, username and password).

Solid currently uses WebID-TLS as its primary authentication mechanism. Alternative complementary mechanisms are also being actively investigated. In addition, Solid recommends that server implementations also offer secondary authentication available for users for Account Recovery (via email or some other out-of-band mechanism).

Primary Authentication

Solid, being a decentralized web application platform, has a set of requirements for its authentication mechanisms that are not commonly encountered by most platforms and ecosystems. Specifically, it requires *cross-domain*, de-centralized authentication mechanisms not tied to any particular identity provider or certificate authority.

WebID-TLS

**Note:** Several browser vendors (Chrome, Firefox) have removed support for the KEYGEN element, on which WebID-TLS relied for in-browser certificate generation. Solid uses the [WebID-TLS protocol](http://www.w3.org/2005/Incubator/webid/spec/tls/) as one of its primary authentication mechanism. Instead of usernames, it uses WebIDs as unique identifiers, as previously mentioned. And instead of using passwords as bearer tokens, it uses cryptographic certificates (stored and managed by the user's web browser) to prove a user's identity.

When accessing a Solid server using WebID-TLS, a user is presented by their web browsers with a popup asking them to select an appropriate security certificate for that site. After a user makes their selection, the server securely matches the private key stored by the browser with the public key stored in that user's [WebID Profile Document](https://github.com/solid/solid-spec#webid-profile-documents), and authenticates them.

WebID-OIDC

The Solid team is currently implementing support for WebID-OIDC as another primary authentication mechanism. It is based on the OAuth2/OpenID Connect protocols, adapted for WebID based decentralized use cases.

<https://www.w3.org/2005/Incubator/webid/spec/identity/>

WebID-OIDC (see [CHANGELOG.md](https://github.com/solid/webid-oidc-spec/blob/master/CHANGELOG.md)) is an authentication delegation protocol (as well as a toolkit of useful auth-related verification techniques) suitable for WebID-based decentralized systems such as [Solid](https://github.com/solid/solid), as well as most LDP-based systems. It is based on decentralized OAuth2/OpenID Connect.

The end result of any WebID-based authentication workflow is a verified WebID URI (specifically, the recipient verifies that the agent controls that URI). For example, [WebID-TLS](https://github.com/solid/solid-spec/blob/master/authn-webid-tls.md) derives the WebID URI from a TLS certificate, and verifies the certificate against the public key in an agent's WebID Profile. Similarly, the end result of [OpenID Connect (OIDC)](https://openid.net/specs/openid-connect-core-1_0.html) workflows is a verified ID Token. The WebID-OIDC protocol specifies a mechanism for getting a WebID URI from an OIDC ID Token, and gains the benefits of both the decentralized flexibility of WebID, and the field-proven security of OpenID Connect.

See also: [Motivation for WebID-OIDC](https://github.com/solid/webid-oidc-spec/blob/master/motivation.md).

Benefits and Capabilities Fully decentralized cross-domain authentication (any peer node can serve as an identity provider as well as a relying party to any other node)

Builds on decades of real-world authentication industry experience

Incorporates lessons from, and fixes to threat models of: SAML, OpenID and OpenID 2, OAuth and OAuth 2. See, for example, [RFC 6819 - OAuth 2.0 Threat Model and Security Considerations](http://tools.ietf.org/html/rfc6819) -- OpenID Connect was developed in large part to address the threats outlined there.

Stands on the shoulders of giants (makes use of the JOSE suite of standards for token representation, cryptographic signing and encryption, including [JWT](https://tools.ietf.org/html/rfc7519), [JWA](https://tools.ietf.org/html/rfc7518), [JWE](https://tools.ietf.org/html/rfc7516) and [JWS](https://tools.ietf.org/html/rfc7515))

Sign Off (and Single Sign Off) capability

Capability for [revocations](https://tools.ietf.org/html/rfc7009), black lists and white lists of both providers and client apps

Supports authentication for the full range of agents and clients: in-browser Javascript apps, traditional server-side web apps, mobile and desktop apps, and IoT devices.

Compatibility with existing [Web Access Control](https://github.com/solid/web-access-control-spec) ACL implementations such as those in Solid servers.

Sets up the infrastructure for adding Capabilities functionality to Solid

If You're Unfamiliar with OIDC

If you're not familiar with the OIDC/OAuth2 workflow, you should do the following:

Read the [Brief Workflow Summary](https://github.com/solid/webid-oidc-spec#brief-workflow-summary) section below

Refer to the [Decentralized Authentication Glossary](https://github.com/solid/webid-oidc-spec#decentralized-authentication-glossary) to help clarify how the various terms (Relying Party, Provider, etc) apply to WebID systems.

Read the [OpenID Connect explained](http://connect2id.com/learn/openid-connect) article. Becoming familiar with the basic OIDC concepts will be quite helpful with understanding this spec.

Differences from Classic OpenID Connect

WebID-OIDC makes the following changes to the base OpenID Connect protocol (which itself improves and builds on OAuth 2):

Discusses and formalizes the [Provider Selection](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md#21-provider-selection) step.

Adds a procedure for [Deriving a WebID URI from ID Token](https://github.com/solid/webid-oidc-spec#deriving-webid-uri-from-id-token), since WebID-based protocols use the WebID URI as a globally unique identifier (rather than the combination of issuer and subject claims).

Adds an additional step: [WebID Provider Confirmation](https://github.com/solid/webid-oidc-spec#webid-provider-confirmation). After the WebID URI is extracted, the recipient of the ID Token must confirm that the Provider was indeed authorized by the holder of the WebID profile.

Specifies the [Authorized OIDC Issuer Discovery](https://github.com/solid/webid-oidc-spec#authorized-oidc-issuer-discovery) process (used as part of Provider Confirmation, and during Provider Selection steps).

It's also worth mentioning that while traditional OpenID Connect use cases are concerned with retrieving user-related claims from [UserInfo endpoints](https://openid.net/specs/openid-connect-core-1_0.html%22%20%5Cl%20%22UserInfo), most WebID based systems replace the UserInfo mechanism with the contents of [WebID Profile](https://github.com/solid/solid-spec#webid-profile-documents)documents.

Brief Workflow Summary

The overall sign in workflow used by the WebID-OIDC protocol is as follows. For example, here is what happens when Alice tries to request the resource https://bob.com/resource1.

[Initial Request](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md#1-initial-request): Alice (unauthenticated) makes a request to bob.com, receives an HTTP 401 Unauthorized response, and is presented with a 'Sign In With...' screen.

[Provider Selection](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md#21-provider-selection): She selects her WebID service provider by clicking on a logo, typing in a URI (for example, alice.solidtest.space), or entering her email.

[Local Authentication](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md#3-local-authentication-to-provider): Alice gets redirected towards her service provider's own Sign In page, thus requestinghttps://alice.solidtest.space/signin, and authenticates using her preferred method (password, WebID-TLS certificate, FIDO 2 / [WebAuthn](https://w3c.github.io/webauthn/) device, etc).

[User Consent](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md#4-user-consent): (Optional) She'd also be presented with a user consent screen, along the lines of "Do you wish to sign in to bob.com?".

[Authentication Response](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md#5-authentication-response): She then gets redirected back towards https://bob.com/resource1 (the resource she was originally trying to request). The server, bob.com, also receives a signed ID Token from alice.solidtest.space that was returned with the response in point 3, attesting that she has signed in.

[Deriving a WebID URI](https://github.com/solid/webid-oidc-spec#deriving-webid-uri-from-id-token): bob.com (the server controlling the resource) validates the ID Token, and extracts Alice's WebID URI from inside it. She is now signed in to bob.com as user https://alice.solidtest.space/#i.

[WebID Provider Confirmation](https://github.com/solid/webid-oidc-spec#webid-provider-confirmation): bob.com confirms that solidtest.space is indeed Alice's authorized OIDC provider (by matching the provider URI from the iss claim with Alice's WebID).

There is a lot of heavy lifting happening under the hood, performed by bob.com and alice.solidtest.space, the two servers involved in this exchange. They establish a trust relationship with each other (via [Discovery](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md#22-provider-discovery), and [Dynamic Registration](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md#23-dynamic-client-registration-first-time-only)), they verify each other's signatures against their public keys, and verify Alice's client app (if she's using one). Fortunately, all of that complexity is hidden from the user (and most of it is also hidden from the app developer).

Deriving WebID URI from ID Token

A WebID-OIDC conforming Relying Party tries the following methods, in order, to obtain a WebID URI from an ID Token:

Method 1 - Custom webid claim

First, check the ID Token payload for the webid claim. This claim is added to the set of [OpenID Connect ID Token](https://openid.net/specs/openid-connect-core-1_0.html#IDToken) claims by this WebID-OIDC spec. (Note that the set of ID Token claims is extensible, by design, as explained in the OIDC Core spec.) If the webid claim is present in the ID Token, its value should be used as the WebID URI by the Relying Party, instead of the traditional sub (subject identifier) claim.

This method is useful when the identity providers issuing the tokens can customize the contents of their ID tokens (and can add the webid claim).

Method 2 - Value of sub claim contains a valid HTTP(S) URI

If the webid claim is not present in the ID Token, the Relying Party should check if the sub claim contains as its value a valid HTTP(S) URI. If the value of the subject claim is a valid URI, it should be used as the WebID URI by the Relying Party.

This method is useful when the identity providers issuing the tokens cannot add claims but can set their own values of their subject claims (that is, they're not automatically generated as UUIDs, etc).

Method 3 - UserInfo request + website claim

If a WebID URI is not found in either the webid or sub claim, the Relying Party should proceed to make an OpenID Connect [UserInfo Request](https://openid.net/specs/openid-connect-core-1_0.html%22%20%5Cl%20%22UserInfo), with the appropriate Access Token that it received alongside the ID Token. This method is provided for cases where users do not have control over the contents of the ID Tokens issued by their Providers and so would not be able to use the previous methods. This would be the case, for example, if a user wanted to sign in to a WebID-OIDC Relying Party using an existing mainstream Provider such as Google. Once the UserInfo response is received by the Relying Party, the standard website claim should be used as the WebID URI by that RP.

WebID Provider Confirmation

The Problem

The OIDC spec uses the ID Token's subject claim as a unique user id. However, it requires that the id is unique for a given Provider. Given that a WebID is a globally unique user identifier, the WebID-OIDC protocol needs to take an additional step and confirm that the holder of that WebID has authorized a given Provider to use that WebID. Otherwise, the following situation can happen:

Alice logs in to bob.com with her identity Provider of choice, alice.com. The ID Token from alice.com claims Alice's WebID is https://alice.com/#i. So far so good.

An attacker also logs in to bob.com, using evilbox.com as an identity Provider. And because they happen to control that server, they can put anything they want in the webid claim of any ID Token coming out of that server. So they alsoclaim that their webid is https://alice.com/#i.

Without an additional confirmation step, how can a recipient of an ID Token (here, bob.com) know which of those login attempts is correct? To put it another way, how can a recipient know which Provider is approved by the owner of the WebID?

The Solution

When presented with WebID-OIDC credentials in the form of bearer tokens, the Resource Server MUST confirm that the Identity Provider (the value in the issuer claim) is authorized by the holder of the WebID, by doing the following:

(Common case) If the server that issued the ID Token is the same entity that hosts the WebID profile, then it is considered the authorized OIDC provider for that WebID (short-circuiting the Provider Confirmation process), and no further steps need to be taken. Specifically, one of the following must be true:

The [origin](https://developer.mozilla.org/en-US/docs/Web/API/URL/origin) of the WebID URI is the same as the origin of the URI in the issuer claim. (For example, iss: 'https://example.com' and the WebID URI is https://example.com/profile#me).

The WebID URI is a subdomain of the issuer of the ID Token. For example, iss: 'https://example.com' and the WebID URI is https://alice.example.com/profile#me. If neither of the above is the case (and the WebID is hosted on a security realm different than that of its OIDC provider), further steps need to be taken for confirmation.

Determine the **authorized OIDC provider** URI for that WebID, by performing [Authorized OIDC Issuer Discovery](https://github.com/solid/webid-oidc-spec#authorized-oidc-issuer-discovery).

If the Provider URI is not discoverable (either from the header or the body of the [WebID Profile](https://github.com/solid/solid-spec#webid-profile-documents), the Resource Server MUST reject the credentials or authentication attempt.

If the Provider URI is discovered, it MUST match the Issuer URI in the ID Token (the iss claim), reject the credentials otherwise.

Authorized OIDC Issuer Discovery

During the Provider Selection or [Provider Confirmation](https://github.com/solid/webid-oidc-spec#webid-provider-confirmation) steps it is necessary to discover, for a given WebID, the URI of the authorized OIDC provider for that WebID.

First, attempt to [discover from link headers](https://github.com/solid/webid-oidc-spec#issuer-discovery-from-link-header)

If not found, proceed to [discover from the WebID Profile](https://github.com/solid/webid-oidc-spec#issuer-discovery-from-webid-profile)

Note that this procedure is different from the classic [OpenID Provider Issuer Discovery](https://openid.net/specs/openid-connect-discovery-1_0.html#IssuerDiscovery) process, since that spec is concerned with discovering the issuer URI from a user's email address, using the WebFinger protocol. Whereas this spec needs to derive the issuer URI from a WebID URI (which is often hosted on a different domain than the issuer).

Issuer Discovery From Link Header

To discover the authorized OIDC Issuer for a given WebID from Link rel headers:

Make an HTTP OPTION request to the WebID URI.

Parse the Link: header, and check for the value of the http://openid.net/specs/connect/1.0/issuer link relation. If present, use the value of that link relation as the authorized Provider URI. For example: Link: <https://provider.example.com>; rel="http://openid.net/specs/connect/1.0/issuer" means that https://provider.example.com is the authorized OIDC provider for that URI.

If the Link header is not present (or does not contain the relevant link relation), proceed to discovering the issuer from the WebID profile contents.

Issuer Discovery From WebID Profile

To discover the authorized OIDC Issuer for a given WebID from the WebID Profile contents (this requires Turtle/RDF parsing capability):

Dereference the WebID URI (make an HTTP GET request) and fetch the contents of the WebID Profile (typically in Turtle or JSON-LD format or some other RDF serialization).

Parse the RDF, and query for the object of the statement containing the<http://www.w3.org/ns/solid/terms#oidcIssuer> predicate.

For example, if Alice (with the WebID of https://alice.example.com/profile#me) wanted to specify https://provider.com as the authorized OIDC provider for that profile, she would add the following triple to her profile:

@prefix solid: <http://www.w3.org/ns/solid/terms#>.

# ...

<#me> solid:oidcIssuer <https://provider.com> .

Detailed Sign In Workflow Example

To walk through a more detailed example for WebID-OIDC login, refer to the [Example WebID-OIDC Workflow](https://github.com/solid/webid-oidc-spec/blob/master/example-workflow.md) doc.

Decentralized Authentication Glossary

In order to discuss decentralized authentication protocol details, it would be helpful to familiarize ourselves with the terminology that is frequently used by various decentralized protocol specs (such as OAuth2, OpenID Connect).

User

Human user. If the user is an app or service (that has its own WebID Profile), this can be generalized to Agent. Also called Resource Owner. In the following examples, Alice and Bob are Users.

User-Agent

A formal name for a Browser. Note that this is often separate from a Client application (in many cases, client apps written in Javascript run inside the browser).

Identity Provider (OP)

An OpenID Connect Identity Provider (called OP in most OIDC specs). Also sometimes referred to as Issuer. This can be either a POD (see below) or an external OIDC provider such as [Google](https://developers.google.com/identity/protocols/OpenIDConnect). In the spec, Alice's POD, alice.com, will mostly play the role of a Provider.

Resource Server (RS)

A server hosting resources that the user wants to access, such as HTML, images, Linked Data / RDF sources, and so on. In the spec, bob.com will be used as the Resource Server (that is, Alice will be requesting resources on Bob's server). Note:In the traditional federated social sign on context, a provider (such as Facebook) serves as both an Identity Provider and a Resource Server.

Relying Party (RP)

A Relying Party is a POD or a client app that has to rely on an ID Token that's issued by a Provider. In the spec, when Alice tries to access a resource on bob.com, Bob's POD acts as the Relying Party, in that interaction. And correspondingly, Alice's POD, alice.com, will serve as the Identity Provider, again for that interaction.

Incidentally, when Alice tries to access a resource on her own POD, alice.com plays all of the roles -- it's both the Provider and a Relying Party (as well as the Resource Server).

POD

A Personal Online Datastore (POD for short). It plays several roles -- firstly, it stores a user's data (and so acts as a Resource Server). In many cases, it also hosts the user's WebID Profile, and implements the API endpoints that allow it to act as a WebID-OIDC Identity Provider (OP). Lastly, when users requests resources from it, the POD also acts as a Relying Party (a recipient of those users' ID Tokens). In this spec, alice.com and bob.com are both PODs.

Home POD vs Other POD

A user's Home POD is one that hosts their WebID Profile, and also acts as that user's Identity Provider. We use the term Other POD in this spec to denote some other WebID-OIDC compliant POD, acting as a Resource Server and Relying Party, that a user is trying to access using the WebID URI and Profile of their Home POD.

When Alice tries to access a resource on Bob's POD, alice.com is her Home POD, and bob.com plays the role of the Other POD.

Public Client vs Confidential Client

Public - in-browser, mobile or desktop app, cannot be trusted with securely storing secrets (private key material, secret client IDs, etc). Confidential - server-side app, can be trusted with secrets.

Presenter

A public client app that is trying to present a user's credentials from their home POD to some other POD. For example, Bob is trying to access, via a client app, a shared file on Alice's alice.com POD, logging in using his own bob.comPOD/provider. In this example, bob.com is a Provider, alice.com is a Relying Party, and the client app (say, a browser-based HTML editor) is a Presenter.

Alternative Authentication Mechanisms

There are several other authentication mechanisms that are currently being investigated, such as combinations of traditional username-and-password authentication and WebID-TLS Delegation).

Secondary Authentication: Account Recovery

Regardless of the primary authentication mechanism, bearer tokens and other proofs of identity tend to get lost by users. Passwords can be forgotten, browser certificates can be lost to hardware failure, and so on. Solid recommends that secondary Account Recovery mechanisms are provided by server implementers, to aid in these scenarios.

## Legal

Who should be responsible for governing the WebID registry?

An institution? How to build trust? How to finance?

# Personal Online Data Storage

|  |
| --- |
| As a person I want to have a place online to collect and store information about myself so that I can present myself to others online. As a parent or guardian, I I want to have a place online to collect and store information about my child so that my child can present themselves to others online |

## Ethics

A Pod is a physical hard drive containing the personal data describing a single person.

 “Personal Data means any information relating to an identified or identifiable natural person (‘data subject’); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person;”

* General Data Protection Regulation Article 4 (1)

The collective of one individual's personal data forms a digital identity (or perhaps digital alter ego is more fitting). A digital identity encompasses all of our personal data shadowing, representing and connected to our physical and ideological self.

Data points are linked in meaning. For example, the genetic profile of one person can give insight into the genetic profile of family members. Data points can lead to deductions that result in other data points. When does one digital identity start and another digital identity stop? At what point does data become public as a result of describing a crowd?

If one person records their observations on another person who owns those observations? The observer or the observed? What responsibilities do the observer and the observed have in relation to each other? Since the massive scale and systematisation of observation of people and their thoughts as a result of the Web, these questions are increasingly important to address.

The question of personal data falls into an unknown territory in between corporate ownership, intellectual property, and slavery.

Ownership involves determining rights and duties over property. While the Web is not owned by anyone, corporations have come to collect, store, and control the personal data, creating value making use of data collection, search engines and communication tools. By default, as a side effect to owning the intellectual property making up the online tools, these corporations have been collecting our digital identities as raw material for the services delivered to other companies at a profit.

Data can be replicated making it somewhat like intellectual property. Multiple individuals can hold a copy of a digital identity. The replicability of data makes the question of control complex. Ownership implies exclusivity, particularly with abstract concepts like ideas or data points. It is not enough to simply have a copy of your own data. Others should be restricted in their access to what is yours. Knowing what data others keep is a near impossible task. The simpler approach would be to cloak yourself in nonsense. To ensure that corporations or institutions do not have a copy of your data it is possible to send noise to confuse the data that they have. For example, a robot could randomly search terms that you would not be inclined to usually search for making that data obtained by the search engine useless through confusion.

Can one person legitimately control the digital identity of another person? Slavery, the ownership of a person, is outlawed in all recognised countries. The institution of marriage originates from the need to grant property rights. The sexual and reproductive identity of the woman was controlled by the father and passed to the husband. Today, our digital identities are controlled by others, a situation of which the legitimacy is being increasingly questioned.

The ability for each and every person to self-determine is a fairly novel concept. If we go from the starting point that individuals should be in control of their digital identities, use of that data by another person, company or institution, requires the explicit permission of the owner.

## Design

Order a box online or via a shop? How to connect the WebID to the Pod?

## Technical

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF\*, SPARQL)
4. Include links to other URIs. so that they can discover more things.

Simple.  In fact, though, a surprising amount of data isn't linked in 2006, because of problems with one or more of the steps.  This article discusses solutions to these problems, details of implementation, and factors affecting choices about how you publish your data.

The four rules

I'll refer to the steps above as rules, but they are expectations of behavior.  Breaking them does not destroy anything, but misses an opportunity to make  data interconnected.  This in turn limits the ways it can later be reused in unexpected ways.  It is the unexpected re-use of information which is the value added by the web.

The first rule, to identify things with URIs,  is pretty much understood by most people doing semantic web technology.  If it doesn't use the universal URI set of symbols, we don't call it Semantic Web.

The second rule, to use HTTP URIs,  is also widely understood.  The only deviation has been, since the web started,  a constant tendency for people to invent new URI schemes (and sub-schemes within the urn: scheme)  such as LSIDs and handles and XRIs and DOIs and so on, for various reasons.  Typically, these involve not wanting to commit to the established Domain Name System (DNS) for delegation of authority but to construct something under separate control.   Sometimes it has to do with not understanding that HTTP URIs are names (not addresses) and that HTTP name lookup is a complex, powerful and evolving set of standards. This issue discussed at length elsewhere, and time does not allow us to delve into it here. [ @@ref TAG finding, etc])

The third rule, that one should serve information on the web against a URI, is, in 2006, well followed for most ontologies, but, for some reason, not for some major datasets.  One can,  in general,  look up the properties and classes one finds in data, and get information from the RDF, RDFS, and OWL ontologies including the relationships between the terms in the ontology.

The basic format here for RDF/XML, with its popular alternative serialization N3 (or Turtle). Large datasets provide a SPARQL query service, but the basic linked data should br provided as well.

Many research and evaluation projects in the few years of the Semantic Web technologies produced ontologies, and significant data stores, but the data, if available at all, is buried in a zip archive somewhere, rather than being accessible on the web as linked data.  The Biopax project, the CSAktive data on computer science research people and projects were two examples. [The CSAktive data is now (2007) available as linked data]

There is also a large and increasing amount of URIs of non-ontology data which can be looked up.  [Semantic wikis](http://ontoworld.org/wiki/Semantic_wiki) are one example. The "Friend of a friend" (FOAF) and *Description of a Project* (DOAP) ontologies are used to build social networks across the web.    Typical [social network portals](http://en.wikipedia.org/wiki/List_of_social_networking_websites) do not provide links to other sites, nor expose their data in a standard form.

LiveJournal and Opera Community are two portal web sites which do in fact publish their data in RDF on the web.   (Plaxo has a trail scheme, and I'm not sure whether they support *knows* links). This means that I can write in my FOAF file that I know Håkon Lie by using his URI in the Opera Community data, and a person or machine browsing that data can then follow that link and find all his friends. *[Update:]* Also, the Opera Community site allows you to register the RDF URI for yourelf on another site. This means that public data about you from different sites can be linked together into one web, and a person or machine starting with your Opera identity can find the others.

The fourth rule, to make links elsewhere,  is necessary to connect the data we have into a web, a serious, unbounded web in which one can find al kinds of things,  just as on the hypertext web we have managed to build.

In hypertext web sites it is considered generally rather bad etiquette not to link to related external material.  The value of your own information is very much a function of what it links to, as well as the inherent value of the information within the web page.  So it is also in the Semantic Web.

So let's look at the ways of linking data, starting with the simplest way of making a link.

*Basic web look-up*

The simplest way to make linked data is to use, in one file, a URI which points into another.

When you write an RDF file,   say <http://example.org/smith>, then you can use local identifiers within the file, say  #albert, #brian and #carol.  In N3 you might say

<#albert> fam:child <#brian>, <#carol>.

or in RDF/XML

<rdf:Description about="#albert"
 <fam:child rdf:Resource="#brian">
 <fam:child rdf:Resource="#carol">
</rdf:Description>

The WWW architecture now gives a global identifier  "http://example.org/smith#albert" to Albert.  This is a valuable thing to do, as anyone on the planet can now use that global identifier to refer to Albert and give more information.

For example, in the document <http://example.org/jones> someone might write:

<#denise> fam:child <#edwin>, <smith#carol>.

or in RDF/XML

<rdf:Description about="#denise"
 <fam:child rdf:Resource="#edwin">
 <fam:child rdf:Resource="http://example.org/smith#carol">
</rdf:Description>

Clearly it is reasonable for anyone who comes across the identifier 'http://example.org/smith#carol" to:

Form the URI of the document by truncating before the hash

Access the document to obtain information about #carol

We call this dereferencing the URI.  This is basic semantic web.

There are several variations.

*Variation: URIs without Slashes and HTTP 303*

There are some circumstances in which dividing identifiers into documents doesn't work very well.   There may logically be one global symbol per document per document, and there is a reluctance to include a # in the URI such as

http://wordnet.example.net/antidisesablishmentarianism#word

Historically, the early Dublin Core and FOAF vocabularies did not have # in their URIs.   In any event when HTTP URIs without hashes are used for abstract concepts, and there is a document that carries information about them, then:

An HTTP GET  request on the URI of the concept returns 303 See Also and gives in the Location: header, the URI of the document.

The document is retrieved as normal

This method has the advantage that URIs can be made up of all forms.  It has the disadvantage that an HTTP request mBrowse-ableust be made for every single one.  In the case of Dublin Core, for example, dc:title and dc:creator etc are in fact served by the same ontology document, but  one does not know until they have each been fetched and returned HTTP redirections.

*Variation: FOAF and rdfs:seeAlso*

The [Friend-Of-A-Friend](http://foaf-project.org/) convention uses a form of data link, but  not using either of the two forms mentioned above.  To refer to another person in a FOAF file, the convention was to give two properties, one pointing to the document they are described in, and the other for identifying them within that document.

<#i> foaf:knows [
 foaf:mbox <mailto:joe@example.com>;
 rdfs:seeAlso <http://example.com/foaf/joe> ].

Read, "I know that which has email  joe@example.com and about which more information is in <http://example.com/foafjoe>".

In fact, for privacy, often people don't put their email addresses on the web directly, but in fact put a one-way hash (SHA-1) of their email address and give that. This clever trick allows people who know their email address already to work out that it is the same person, without giving the email away to others.

<#i> foaf:knows [
 foaf:mbox\_sha1sum "2738167846123764823647"; # @@ dummy
 rdfs:seeAslo <http://example.com/foaf/joe> ].

This linking system was very successful, forming a  growing social network, and dominating, in 2006, the linked data available on the web.

However, the system has the snag that it does not give URIs to people, and so basic links to them cannot be made.

I  recommend (e.g in weblogs on [Links on the Semantic Web](http://dig.csail.mit.edu/breadcrumbs/node/62) , [Give yourself a URI](http://dig.csail.mit.edu/breadcrumbs/node/71), and and [Backward and Forward links in RDF just as important](http://dig.csail.mit.edu/breadcrumbs/node/72)) that those making a FOAF file give themselves a URI as well as using the FOAF convention.     Similarly, when you refer to a FOAF  file which gives  a URI to a person, use it in your reference to that person, so that clients which just use URIs and don't know about the FOAF convention can follow the link.

Browsable graphs

So now we have looked at ways of making a link, let's look at the  choices of when to make a link.

One important pattern is a set of data which you can explore as you go link by link by fetching data.   Whenever one looks up the URI for a node in the RDF graph, the server returns information about the arcs out of that node, and the arcs in.  In other words, it returns any RDF statements in which the term appears as either subject or object.

Formally,  call a graph G *browsable* if, for  the URI of any node in G, if I look up that URI I will be returned information which describes the node, where describing a node means:

Returning all statements where the node is a subject or object; and

Describing all blank nodes attached to the node by one arc.

(The subgraph returned has been referred to as "minimum Spanning Graph (MSG [@@ref] ) or  RDF molecule [@@ref], depending on whether nodes are considered identified if they can be expressed as a path of function, or reverse inverse functional properties. A concise bounded description, which only follows links from subject to object,  does not work.)

In practice, when data is stored in two documents, this means that any RDF statements which relate things in the two files must be repeated in each.  So, for example, in my FOAF page I mention that I am a member of the DIG group, and that information is repeated on the DIG group data. Thus, someone starting from the concept of the group can also find out that I am a member.  In fact, someone who starts off with my URI can find all the people who are in the same group.

*Limitations on browseable data*

So statements which relate things in the two documents must be repeated in each. This clearly is against the first rule of data storage: don't store the same data in two different places: you will have problems keeping it consistent.  This is indeed an issue with browsable data.   A set of  of completely browsable data with links in both directions has to be completely consistent, and that takes coordination, especially if different authors or different programs are involved.

We can have completely browsable data, however, where it is automatically generated.  The [dbview](http://dig.csail.mit.edu/2006/dbview/dbview.py)  server, for example,  provides a browsable virtual  documents containing the data from any arbitrary relational database.

When we have a data from multiple sources, then we have compromises.  These are often settled by common sense, asking the question,

*"If someone has the URI of that thing, what relationships to what other objects is it useful to know about?"*

Sometimes, social questions  determine the answer.  I have links in my FOAF file that I know various people.  They don't generally repeat that information in their FOAF files. Someone may say that they know me, which is an assertion which, in the FOAF convention, is theirs to assert, and the reader's to trust or not.

Other times, the number of arcs makes it impractical.   A GPS track gives thousands of times at which my latitude, longitude are known. Every person loading my FOAF file can expect to get my business card information, but not all those trackpoints. It is reasonable to have a pointer from the track (or even each point) to the person whose position is represented, but not the other way.

One pattern is to have links of a certain property in a separate document.   A person's homepage doesn't list all their publications, but instead puts a link to it a separate document listing them.  There is an understanding that foaf:made gives a work of some sort, but foaf:pubs points to a document giving a list of works.  Thus, someone searching for something foaf:made link would do well to follow a foaf:pubs link.  It might be useful to formalize the notion with a statement like

foaf:made link:listDocumentProperty foaf:pubs.

in one of the ontologies.

*Query services*

Sometimes the sheer volume of data makes serving it as lots of files possible, but cumbersome for efficient remote queries over the dataset.  In this case, it seems reasonable to provide a SPARQL query service.  To make the data be effectively linked, someone who only has the  URI of something must be able to find their way the SPARQL endpoint.

Here again the HTTP 303 response can be used, to refer the enquirer to a document with metadata about which query service endpoints can provide what information about which  classes of URIs.

Vocabularies for doing this have not yet been standardized.

Is your Linked Open Data 5 Star?

(Added 2010). This year, in order to encourage people -- especially government data owners -- along the road to good linked data, I have developped this star rating system.

Linked Data is defined above. Linked Open Data (LOD) is Linked Data which is released under an open licence, which does not impede its reuse for free. Creative Commons CC-BY is an example open licence, as is the UK's [Open Government Licence](http://www.nationalarchives.gov.uk/doc/open-government-licence/). Linked Data does not of course in general have to be open -- there is a lot of important use of lnked data internally, and for personal and group-wide data. You can have 5-star Linked Data without it being open. However, if it claims to be Linked Open Data then it does have to be open, to get any star at all.

Under the star scheme, you get one (big!) star if the information has been made public at all, even if it is a photo of a scan of a fax of a table -- if it has an open licence. The you get more stars as you make it progressively more powerful, easier for people to use.

|  |  |
| --- | --- |
| ★ | Available on the web (whatever format) *but with an open licence, to be Open Data* |
| ★★ | Available as machine-readable structured data (e.g. excel instead of image scan of a table) |
| ★★★ | as (2) plus non-proprietary format (e.g. CSV instead of excel) |
| ★★★★ | All the above plus, Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff |
| ★★★★★ | All the above, plus: Link your data to other people’s data to provide context |

How well does your data do? You can buy [5 star data mugs](http://www.cafepress.co.uk/w3c_shop.480759174), T-shirts and bumper stickers from the W3C shop at cafepress: use them to get your colleages and fellows conference-goers thinking 5 star linked data. (Profits also help W3C :-).

Now in 2010, people have been pressing me, for governmet data, to add a new requirement, and that is there should be metadata about the data itself, and that that metadata should be availble from a major catalog. Any open dataset (or even datasets which are not but should be open) can be regisetreed at ckan.net. Government datasets from the UK and US hsould be regisetred at data.gov.uk or data.gov respectively. Other copuntries I expect to develop their own registries. Yes, there should be metadata about your dataset. That may be the subject of a new note in this series.

Conclusion

Linked data is essential to actually connect the semantic web.  It is quite easy to do with a little thought, and becomes second nature.   Various common sense considerations determine when to make a link and when not to.

The [Tabulator](http://dig.csail.mit.edu/2005/ajar/ajaw/tab) client (running in a suitable browser)  allows you to browse linked data using the above conventions, and can be used to check that your linked data works.

References

[Ding2005] Li Ding, et. al.,  [*Tracking RDF Graph Provenance using RDF Molecules*](http://ebiquity.umbc.edu/paper/html/id/240/), UMBC Tech Report TR-CS-05-06

Followup

2006-02 Rob Crowell adapts Dan Connolly's DBView (2004) which maps SQL data into linked RDF, adding backlinks.

2006-09-05 Chris Bizer et al adapt [D2R Server](http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2r-server/) to provide a linked data view of a database.

2006-10-10 Chris Bizer et al produce the [Semantic Web Client Library](http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/semwebclient/), "Technically, the library represents the Semantic Web as a single Jena RDF graph or Jena Model." The code feteches web documents as needed to answer queries.

2007-01-15 Yves Raimond has produced a [Semantic Web client for SWI prolog](http://moustaki.org/swic/) wit similar functionality.

I have a talk at the 2009 O'Reilly eGovernment 2.0 conference in Washington DC, talking about "Just a Bag of Chips" @@ref, and talking about the 5 star scheme. Following that, From InkDroid blogged summary (and CSS) of my 5 star sceheme adapted here

[Up to Design Issues](https://www.w3.org/DesignIssues/Overview.html)

[Tim BL](https://www.w3.org/People/Berners-Lee)

A set of [servers](https://github.com/solid/solid-platform#servers) that implement this specification.

## Legal

A Pod provider cannot hijack the Pod of another person

# Data Sharing Preferences

|  |
| --- |
| As a person I want to understand the implications of my data sharing preferences and choose a fixed data sharing preference that will operate on all applications until I choose to change those data sharing preferences. As a parent or guardian, I want to maximise the protection of my child until they reach the age of consent while still giving them access to health and education.  |

## Ethics

How does a person decide what data in their Pod to share with who?

## Design

What are the design requirements for a Pod?

## Technical

Authorization and Access Control

Authorization is the process of deciding whether a user has *access* to a particular resource. If authentication asks "who is the user?", authorization is concerned with "what is the user allowed to do?".

Solid currently uses the Web Access Control (WAC) mechanism for cross-domain authorization for all its resources.

Web Access Control

[Web Access Control (WAC)](https://github.com/solid/web-access-control-spec) is a decentralized system that allows different users and groups various forms of access to resources where users and groups are identified by HTTP URIs. The system is similar to the access control system used within many file systems except that the documents controlled, the users, and the groups, are all identified by URIs. Users are identified by WebIDs. Groups of users are identified by the URI of a class of users which, if you look it up, returns a list of users in the class. This means a WebID hosted by any server can be a member of a group hosted some other server.

Users do not need to have an account (i.e. WebID) on a given server to have access to documents on it.

Web Access Control (WAC)

Web Access Control (WAC) specification (as used by the Solid project). It is based on Tim Berners-Lee's May 2009 proposal, as originally captured, and subsequently evolved by the community, at [Web Access Control Wiki](https://www.w3.org/wiki/WebAccessControl). This spec is a particular subset of the options and extensions described in the wiki.

For use with [LDP](https://www.w3.org/TR/ldp/) (and [LDP Next](https://www.w3.org/community/ldpnext/)) type systems, such as the [Solid](https://github.com/solid/solid) project (see also the parent [spec](https://github.com/solid/solid-spec)).

**Current Spec version:** v.0.5.0 (see [CHANGELOG.md](https://github.com/solid/web-access-control-spec/blob/master/CHANGELOG.md))

Overview

Web Access Control (WAC) is a decentralized cross-domain access control system. The main concepts should be familiar to developers, as they are similar to access control schemes used in many file systems. It's concerned with giving access to agents (users, groups and more) to perform various kinds of operations (read, write, append, etc) on resources. WAC has several key features:

The resources are identified by URLs, and can refer to any web documents or resources.

It is *declarative* -- access control policies live in regular web documents, which can be exported/backed easily, using the same mechanism as you would for backing up the rest of your data.

Users and groups are also identified by URLs (specifically, by [WebIDs](https://github.com/solid/solid-spec%22%20%5Cl%20%22identity))

It is *cross-domain* -- all of its components, such as resources, agent WebIDs, and even the documents containing the access control policies, can potentially reside on separate domains. In other words, you can give access to a resource on one site to users and groups hosted on another site.

Access Control List Resources

In a system that uses Web Access Control, each web resource has a set of *Authorization* statements describing:

Who has access to that resource (that is, who the authorized *agents* are)

What types (or *modes*) of access they have

These Authorizations are either explicitly set for an individual resource, or (more often) inherited from that resource's parent folder or container. In either case, the Authorization statements are placed into separate WAC documents called *Access Control List Resources* (or simply *ACLs*).

Containers and Inherited ACLs

The WAC system assumes that web documents are placed in hierarchical containers or folders. For convenience, users do not have to specify permissions on each individual resource -- they can simply set permissions on a container, add a[acl:default](https://github.com/solid/web-access-control-spec#default-inherited-authorizations) predicate, and have all of the resources in that container [inherit](https://github.com/solid/web-access-control-spec#acl-inheritance-algorithm) those permissions.

Individual Resource ACLs

For fine-grained control, users can specify a set of permissions for each individual resource (which overrides any permissions of its parent container). See the [Example WAC Document](https://github.com/solid/web-access-control-spec#example-wac-document) section to get an idea for what that would look like.

ACL Resource Location Discovery

Given a URL for an individual resource or container, a client can discover the location of its corresponding ACL by performing a HEAD (or a GET) request and parsing the rel="acl" link relation.

Example request to discover the location of the ACL resource for a web document at http://example.org/docs/file1 is given below:

HEAD /docs/file1 HTTP/1.1

Host: example.org

HTTP/1.1 200 OK

Link: <file1.acl>; rel="acl"

The request to discover the location of a container's ACL resource looks similar:

HEAD /docs/ HTTP/1.1

Host: example.org

HTTP/1.1 200 OK

Link: <.acl>; rel="acl"

Note that the acl link relation uses relative path URLs (the absolute URL of the ACL resource in the above example would be /docs/.acl).

Clients MUST NOT assume that the location of an ACL resource can be deterministically derived from a document's URL. For example, given a document with a URL of /docs/file1, clients cannot rely on the assumption that an ACL resource exists at /docs/file1.acl, simply using .acl as a prefix. The actual naming convention for ACL resources can differ for each individual implementation (or even for each individual server). If one server locates the ACL resource by appending the suffix .acl, another server could place the ACL resources into a sub-container (locating it at /docs/.acl/file1.acl for the example above).

ACL Inheritance Algorithm

The following algorithm is used by servers to determine which ACL resources (and hence which set of Authorization statements) apply to any given resource:

Use the document's own ACL resource if it exists (in which case, stop here).

Otherwise, look for authorizations to inherit from the ACL of the document's container. If those are found, stop here.

Failing that, check the container's *parent* container to see if *that* has its own ACL file, and see if there are any permissions to inherit.

Failing that, move up the container hierarchy until you find a container with an existing ACL file, which has some permissions to inherit.

The *root container* of a user's account MUST have an ACL resource specified. (If all else fails, the search stops there.)

It is considered an anti-pattern for a *client* to perform those steps, however. A client may [discover](https://github.com/solid/web-access-control-spec#acl-resource-location-discovery) and load a document's individual resource (for example, for the purposes of editing its permissions). If such a resource does not exist, a client SHOULD NOT search for, or interact with, the inherited ACLs from an upstream container.

ACL Inheritance Algorithm Example

*Note:* The server in the examples below is using the ACL naming convention of appending .acl to resource URLs, for simplicity. As mentioned in the section on [ACL discovery](https://github.com/solid/web-access-control-spec#acl-resource-location-discovery), clients should not use or assume any naming convention.

A request (to read or write) has arrived for a document located at /documents/papers/paper1. The server does as follows:

First, it checks for the existence of an individual corresponding ACL resource. (That is, it checks if paper1.acl exists.) If this individual ACL resource exists, the server uses the Authorization statements in that ACL. No other statements apply.

If no individual ACL exists, the server next checks to see if the /documents/papers/ container (in which the document resides) has its own ACL resource (here, /documents/papers/.acl). If it finds that, the server reads each authorization in the container's ACL, and if any of them contain an acl:default predicate, the server will use them (as if they were specified in paper1.acl). Again, if any such authorizations are found, the process stops there and no other statements apply.

If the document's container has no ACL resource of its own, the search continues upstream, in the *parent* container. The server would check if /documents/.acl exists, and then /.acl, until it finds some authorizations that contain acl:default.

Since the root container (here, /) MUST have its own ACL resource, the server would use the authorizations there as a last resort.

See the [Default (Inherited) Authorizations](https://github.com/solid/web-access-control-spec#default-inherited-authorizations) section below for an example of what a container's ACL resource might look like.

Representation Format

The permissions in an ACL resource are stored in Linked Data format ([Turtle](http://www.w3.org/TR/turtle/) by default, but also available in other serializations).

*Note: A familiarity with Linked Data and*[*RDF Concepts*](http://www.w3.org/TR/rdf11-concepts/)*helps with understanding the terminology used in this spec.*

WAC uses the <http://www.w3.org/ns/auth/acl> ontology for its terms. Through the rest of the spec, the prefix acl: is assumed to mean @prefix acl: <http://www.w3.org/ns/auth/acl#> .

Example WAC Document

Below is an example ACL resource that specifies that Alice (as identified by her WebID https://alice.databox.me/profile/card#me) has full access (Read, Write and Control) to one of her web resources, located at https://alice.databox.me/docs/file1.

# Contents of https://alice.databox.me/docs/file1.acl

@prefix acl: <http://www.w3.org/ns/auth/acl#> .

<#authorization1>

 a acl:Authorization;

 acl:agent <https://alice.databox.me/profile/card#me>; # Alice's WebID

 acl:accessTo <https://alice.databox.me/docs/file1>;

 acl:mode acl:Read,

 acl:Write,

 acl:Control.

Describing Agents

In WAC, we use the term *Agent* to identify *who* is allowed access to various resources. In general, it is assumed to mean "someone or something that can be referenced with a WebID", which covers users, groups (as well as companies and organizations), and software agents such as applications or services.

Singular Agent

An authorization may list any number of individual agents (that are being given access) by using the acl:agent predicate, and using their WebID URIs as objects. The example WAC document in a previous section grants access to Alice, as denoted by her WebID URI, https://alice.databox.me/profile/card#me.

Groups of Agents

To give access to a group of agents, use the acl:agentGroup predicate. The object of an agentGroup statement is a link to a Group Listing document. The WebIDs of group members are listed in it, using the vcard:hasMember predicate. If a WebID is listed in that document, it is given access.

Example ACL resource, shared-file1.acl, containing a group permission:

# Contents of https://alice.databox.me/docs/shared-file1.acl

@prefix acl: <http://www.w3.org/ns/auth/acl#>.

# Individual authorization - Alice has Read/Write/Control access

<#authorization1>

 a acl:Authorization;

 acl:accessTo <https://alice.example.com/docs/shared-file1>;

 acl:mode acl:Read,

 acl:Write,

 acl:Control;

 acl:agent <https://alice.example.com/profile/card#me>.

# Group authorization, giving Read/Write access to two groups, which are

# specified in the 'work-groups' document.

<#authorization2>

 a acl:Authorization;

 acl:accessTo <https://alice.example.com/docs/shared-file1>;

 acl:mode acl:Read,

 acl:Write;

 acl:agentGroup <https://alice.example.com/work-groups#Accounting>;

 acl:agentGroup <https://alice.example.com/work-groups#Management>.

Corresponding work-groups Group Listing document:

# Contents of https://alice.example.com/work-groups

@prefix acl: <http://www.w3.org/ns/auth/acl#>.

@prefix vcard: <http://www.w3.org/2006/vcard/ns#>.

<> a acl:GroupListing.

<#Accounting>

 a vcard:Group;

 vcard:hasUID <urn:uuid:8831CBAD-1111-2222-8563-F0F4787E5398:ABGroup>;

 dc:created "2013-09-11T07:18:19+0000"^^xsd:dateTime;

 dc:modified "2015-08-08T14:45:15+0000"^^xsd:dateTime;

 # Accounting group members:

 vcard:hasMember <https://bob.example.com/profile/card#me>;

 vcard:hasMember <https://candice.example.com/profile/card#me>.

<#Management>

 a vcard:Group;

 vcard:hasUID <urn:uuid:8831CBAD-3333-4444-8563-F0F4787E5398:ABGroup>;

 # Management group members:

 vcard:hasMember <https://deb.example.com/profile/card#me>.

Group Listings - Implementation Notes

When implementing support for acl:agentGroup and Group Listings, keep in mind the following issues:

Group Listings are regular documents (potentially each with its own .acl).

What authentication mechanism should the ACL checking engine use, when making requests for Group Listing documents on other servers?

Infinite request loops during ACL resolution become possible, if an .acl points to a group listing on a different server.

Therefore, for the moment, we suggest that all Group files which are used for group ACLs are public.

Possible future methods for a server to find out whether a given agent is a member of s group are a matter for future research and possible addition here.

**Public Access (All Agents)**

To specify that you're giving a particular mode of access to *everyone* (for example, that your WebID Profile is public-readable), you can use acl:agentClass foaf:Agent to denote that you're giving access to the class of *all* agents (the general public). For example:

@prefix acl: <http://www.w3.org/ns/auth/acl#>.

@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<#authorization2>

 a acl:Authorization;

 acl:agentClass foaf:Agent; # everyone

 acl:mode acl:Read; # has Read-only access

 acl:accessTo <https://alice.databox.me/profile/card>. # to the public profile

Authenticated Agents (Anyone logged on)

Authenticated access is a bit like public access but it is not anonymous. Access is only given to people who have logged on and provided a specific ID. This allows the server to track the people who have used the resource.

To specify that you're giving a particular mode of access to anyone *logged on* (for example, that your collaborative page is open to anyone but you want to know who they are), you can use acl:agentClass acl:AuthenticatedAgent to denote that you're giving access to the class of *all* authenticated agents. For example:

@prefix acl: <http://www.w3.org/ns/auth/acl#>.

@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<#authorization2>

 a acl:Authorization;

 acl:agentClass acl:AuthenticatedAgent; # everyone

 acl:mode acl:Read; # has Read-only access

 acl:accessTo <https://alice.databox.me/profile/card>. # to the public profile

Note that this is a special case of acl:agentClass usage, since it doesn't point to a Class Listing document that's meant to be de-referenced.

An application of this feature is to throw a resource open to all logged on users for a specific amount of time, accumulate the list of those who case as a group, and then later restrict access to that group, to prevent spam.

**Referring to Origins, i.e. Web Apps**

When a compliant server receives a request from a web application running in a browser, the browser will send an extra warning HTTP header, the Origin header.

Origin: https://scripts.example.com:8080

(For background, see also [Backgrounder on Same Origin Policy and CORS](https://solid.github.io/web-access-control-spec/Background)) Note that the origin comprises the protocol and the DNS and port but none of the path, and no trailing slash. All scripts running on the same origin are assumed to be run by the same social entity, and so trusted to the same extent.

When an Origin header is present then BOTH the authenticated agent AND the origin MUST be allowed access

As both the user and the web app get to read or write (etc) the data, then they most BOTH be trusted. This is the algorithm the server must go through.

If the requested mode is available to the public, then succeed 200 OK with added CORS headers ACAO and ACAH \*\*

If the user is not logged on, then fail 401 Unauthenticated

Is the User authenticated is not allowed access required, AND the class AuthenticatedAgent is not allowed access, then fail 403 User Unauthorized

If the Origin header is not present, the succeed 200 OK

If the Origin is allowed by the ACL, then succeed 200 OK with added CORS headers ACAO and ACAH

(In future proposed) Look up the owner's webid(s) to check for trusted apps declared there, and if match, succeed 200 OK with added CORS headers ACAO and ACAH

Fail 403 Origin Unauthorized

Note it is a really good idea to make it clear both in the text of the status message and in the body of the message the difference between the user not being allowed and the web app they are using not being trusted.

\*\* Possible future alternative: Set ACAO header to "\*" indicating that the document is public. This will though block in the browser any access made using credentials.

**Adding trusted web apps.**

The authorization of trusted web app is a running battle between readers and writers on the web, and malevolent parties trying to break in to get unauthorized access. The history or Cross-Site Scripting attacks and the introduction of the Same Origin Policy is not detailed here, The CORS specification in general prevents any web app from accessing any data from or associated with a different origin. The web server can get around CORS. It is a pain to to do so, as it involves the server code echoing back the Origin header in the ACAO header, and also it must be done only when the web app in question actually is trustworthy.

In solid a maxim is, you have complete control of he data. Therefore it is up to the owner of the data, the publisher, the controller of the ACL, or more broadly the person running the solid server, to specify who gets access, be it people or apps. However another maxim is that you can chose which app you use. So of Alice publishes data, and Bob want to use his favorite app, then how does that happen?

**Now:**

The web server can run with a given trusted domain created by the solid developers.

A specific ACL can be be made to allow a given app to access a given file or folder of files.

**Possible future:**

A writer could give in their profile a statement that they will allow readers to use a given app.

 <#me> acl:trustedApp [ acl:origin <https://calendar.example.com>;

 acl:mode acl:Read,

 acl:Append].

 <#me> acl:trustedApp [ acl:origin <https://contacts.example.com>;

 acl:mode acl:Read,

 acl:Write,

 acl:Control].

We define the owners of the resource as people given explicit Control access to it. (Possible future change: also anyone with Control access, even through a group, as the group can be used as a role)

For each owner x, the server looks up the (extended?) profile, and looks in it for a triple of the form

?x acl:trustedApp ?y.

The set of trust objects is the accumulated set of ?y found in this way.

For the app ?z to have access, for every mode of access ?m required there must be some trust object ?y such that

?y acl:origin ?z;

 acl:mode ?m.

Note access to different modes may be given in the same or different trust objects.

Referring to Resources

The acl:accessTo predicate specifies *which resources* you're giving access to, using their URLs as the subjects.

**Referring to the ACL Resource Itself**

Since an ACL resource is a plain Web document in itself, what controls who has access to *it*? While an ACL resource *could* in theory have its own corresponding ACL document (for example, file1.acl controls access to file1, and file1.acl.acl could potentially control access to file1.acl), one quickly realizes thats this recursion has to end somewhere.

Instead, the [acl:Control access mode](https://github.com/solid/web-access-control-spec%22%20%5Cl%20%22aclcontrol) is used (see below), to specify who has access to alter (or even view) the ACL resource.

**Modes of Access**

The acl:mode predicate denotes a class of operations that the agents can perform on a resource.

**acl:Read**

gives access to a class of operations that can be described as "Read Access". In a typical REST API (such as the one used by [Solid](https://github.com/solid/solid-spec#https-rest-api)), this includes access to HTTP verbs GET, and HEAD. This also includes any kind of QUERY or SEARCH verbs, if supported.

**acl:Write**

gives access to a class of operations that can modify the resource. In a REST API context, this would include PUT, POST, DELETE and PATCH. This also includes the ability to perform SPARQL queries that perform updates, if those are supported.

**acl:Append**

gives a more limited ability to write to a resource -- Append-Only. This generally includes the HTTP verb POST, although some implementations may also extend this mode to cover non-overwriting PUTs, as well as the INSERT-only portion of SPARQL-based PATCHes. A typical example of Append mode usage would be a user's Inbox -- other agents can write (append) notifications to the inbox, but cannot alter or read existing ones.

**acl:Control**

is a special-case access mode that gives an agent the ability to *view and modify the ACL of a resource*. Note that it doesn't automatically imply that the agent has acl:Read or acl:Write access to the resource itself, just to its corresponding ACL document. For example, a resource owner may disable their own Write access (to prevent accidental over-writing of a resource by an app), but be able to change their access levels at a later point (since they retain acl:Control access).

**Default (Inherited) Authorizations**

As previously mentioned, not every document needs its own individual ACL resource and its own authorizations. Instead, one can can create an Authorization for a container (in the container's own ACL resource), and then use the acl:defaultpredicate to denote that any resource within that container will *inherit* that authorization. To put it another way, if an Authorization contains acl:default, it will be applied *by default* to any resource in that container.

You can override the default inherited authorization for any resource by creating an individual ACL just for that resource.

An example ACL for a container would look something like:

# Contents of https://alice.databox.me/docs/.acl

@prefix acl: <http://www.w3.org/ns/auth/acl#>.

<#authorization1>

 a acl:Authorization;

 # These statements specify access rules for the /docs/ container itself:

 acl:agent <https://alice.databox.me/profile/card#me>;

 acl:accessTo <https://alice.databox.me/docs/>;

 acl:mode acl:Read,

 acl:Write,

 acl:Control;

 # default says: this authorization (the statements above)

 # will also be inherited by any resource within that container

 # that doesn't have its own ACL.

 acl:default <https://alice.databox.me/docs/>.

**See also**

[Background on CORS](https://solid.github.io/web-access-control-spec/Background)

**Old discussion of access to group files**

**Group Listings - Authentication of External Requests**

*This section is not normative*

Group Listings via acl:agentGroup links introduce the possibility of an ACL checking engine having to make requests to other servers. Given that access to those external group listings can be protected, the question immediately arises: By what mechanism should the ACL checking engine authenticate its request to external servers?

For example: Alice sends a GET request to a resource on the server https://a.com. The ACL for that resource links to a group listing on an external server, https://b.com. In the process of resolving the ACL, a.com must send a request to b.com, to get that group listing. Note that it's not Alice herself (or her application) that is sending that request, it's actuallya.com sending it (as part of trying to resolve its own ACL). How should a.com authenticate itself? Does it have its own credentials, or does it have a way to say that it's acting on behalf of Alice? Or both?

There are several implementation possibilities:

**No authentication**. The ACL checking engine sends *un-authenticated* requests to external servers (to fetch group listings). This is the simplest method to implement, but suffers from the limitation that those external group listings need to be public-readable. THIS IS THE ONLY METHOD CURRENTLY IN USE

**WebID-TLS Delegation**. If your implementation uses the WebID-TLS authentication method, it also needs to implement the ability to delegate its requests on behalf of the original user. (No, the original requester may not be allowed access -- you don't have to able to access a group to be in it) For a discussion of such a capability, see the [Extending the WebID Protocol with Access Delegation](http://bblfish.net/tmp/2012/08/05/WebID_Delegation.pdf) paper. One thing to keep in mind is - if there are several hops (an ACL request chain across more than one other domain), how does this change the delegation confirmation algorithm? If the original server is explicitly authorized for delegation by the user, what about the subsequent ones?

**ID Tokens/Bearer Tokens**. If you're using a token-based authentication system such as OpenID Connect or OAuth2 Bearer Tokens, it will also need to implement the ability to delegate its ACL requests on behalf of the original user. See[PoP/RFC7800](https://tools.ietf.org/html/rfc7800) and [Authorization Cross Domain Code](http://openid.bitbucket.org/draft-acdc-01.html) specs for relevant examples.

**Infinite Request Loops in Group Listings**

Since Group Listings (which are linked to from ACL resources using the acl:agentGroup predicate) reside in regular documents, those documents will have their very own .acl resources that restrict which users (or groups) are allowed to access or change them. This fact, that .acls point to Group Listings, which can have .acls of their own, which can potentially also point to Group Listings, and so on, introduces the potential for infinite loops during ACL resolution.

Take the following situation with two different servers:

https://a.com https://b.com

------------- GET ---------------

group-listA <------ group-listB.acl

 | ^ contains:

 | | agentGroup <a.com/group-ListA>

 v GET |

group-listA.acl ------> group-listB

 contains:

 agentGroup <b.com/group-listB>

The access to group-listA is controlled by group-listA.acl. So far so good. But if group-listA.acl contains any acl:agentGroup references to *another* group listing (say, points to group-listB), one runs into potential danger. In order to retrieve that other group listing, the ACL-checking engine on https://b.com will need to check the rules in group-listB.acl. And if group-listB.acl (by accident or malice) points back to group-listA a request will be made to access group-listA on the original server https://a.com, which would start an infinite cycle.

To guard against these loops, implementers have several options:

**A) Do not allow cross-domain Group Listing resolutions**. The simplest to implement (but also the most limited) option is to disallow cross-domain Group Listings resolution requests. That is, the ACL-checking code could detect agentGroup links pointing to external servers during ACL resolution time, and treat those uniformly (as errors, or as automatic "access denied").

**B) Treat Group Listings as special cases**. This assumes that the server has the ability to parse or query the contents of a Group Listing document *before* resolving ACL checks -- a design decision that some implementations may find unworkable. If the ACL checking engine can inspect the contents of a document and know that it's a Group Listing, it can put in various safeguards against loops. For example, it could validate ACLs when they are created, and disallow external Group Listing links, similar to option A above. Note that even if you wanted to ensure that no .acls are allowed for Group Listings, and that all such documents would be public-readable, you would still have to be able to tell Group Listings apart from other documents, which would imply special-case treatment.

**C) Create and pass along a tracking/state parameter**. For each ACL check request beyond the original server, it would be possible to create a nonce-type tracking parameter and pass it along with each subsequent request. Servers would then be able to use this parameter to detect loops on each particular request chain. However, this is a spec-level solution (instead of an individual implementation level), since all implementations have to play along for this to work. See issue [solid/solid#8](https://github.com/solid/solid/issues/8) for further discussion).

**D) Ignore this issue and rely on timeouts.** It's worth noting that if an infinite group ACL loop was created by mistake, this will soon become apparent since requests for that resource will time out. If the loop was created by malicious actors, this is comparable to a very small, low volume DDOS attack, which experienced server operators know how to guard against. In either case, the consequences are not disastrous.

**Other ideas about specifying trusted apps**

A reader can ask to use a given app, by publishing the fact that she trusts a given app.

<#me> acl:trustsForUse [ acl:origin <https://calendar.example.com>;

 acl:mode acl:Read,

 acl:Append].

<#me> acl:trustsForUse [ acl:origin <https://contacts.example.com>;

 acl:mode acl:Read,

 acl:Write,

 acl:Control].

A writer could have also more sophisticated requirements, such as that any app Alice wants to use must be signed by developer from a given list, and so on.

Therefore, by pulling the profiles of the reader and/or the writer, and/or the Origin app itself, the system can be adjusted to allow new apps to be added without bad things happening

**Not Supported by Design**

This section describes some features or acl-related terms that are not included in this spec by design.

**Resource Owners**

WAC has no formal notion of a resource owner, and no explicit Owner access mode. For convenience/UI purposes, it may be assumed that Owners are agents that have Read, Write and Control permissions.

**acl:accessToClass**

The predicate acl:accessToClass is not supported.

**Regular Expressions**

The use of regular expressions in statements such as acl:agentClass is not supported.

**Content Representation**

Solid deals with reading and writing two kinds of resources:

Linked Data resources (RDF in the form of JSON-LD, Turtle, HTML+RDFa, etc)

Everything else (binary data and non-linked-data structured text)

While you can build Solid applications with non-linked data resources, using actual RDF-based Linked Data provides you with considerable benefits in terms of interoperability with the rest of the Solid app ecosystem.

Resources are grouped in directory-like **Containers** (currently conforming to the [LDP Basic Container spec](https://www.w3.org/TR/ldp/#ldpbc)).

**See component spec:**[**Solid Content Representation**](https://github.com/solid/solid-spec/blob/master/content-representation.md)

**Reading and Writing Resources**

**HTTPS REST API**

Solid extends the [Linked Data Platform spec](https://www.w3.org/TR/ldp/) to provide a simple REST API for CRUD operations on resources and containers.

**See component spec:**[**HTTPS REST API**](https://github.com/solid/solid-spec/blob/master/api-rest.md)

**WebSockets API**

Solid also provides a WebSockets based API for a PubSub (Publish/Subscribe) mechanism, through which clients can be notified in real time of changes affecting a give resource.

**See component spec: [WebSockets API](https://github.com/solid/solid-spec/blob/master/api-websockets.md)**

## Legal

What are the requirements to become a Pod Provider?

# Finding Solid Apps

|  |
| --- |
| As a person I want to search for applications to use information about me so that I can get stuff done efficiently to my own benefit |

## Ethics

What makes Solid Apps unique is that their functionality is independent of the data generated which is stored in the Pod. Separating the app functionality from the data generated means that a person can switch between apps and bring their data with them. This makes the decision to use an application or not independent of the convenience from having a collection of historical data such as contacts, or previous conversations.

Social Web App Protocols

In addition to read/write operations on resources, Solid provides a number of specs and recommendations to help developers achieve interoperability between various social web applications that are part of the ecosystem.

## Design

What are the design requirements for Solid App Providers to publish their app?

What are the design requirements for people to find Solid Apps?

## Technical

Notifications

See component spec: [Linked Data Notifications](https://www.w3.org/TR/ldn/)

Friends Lists, Followers and Following

API recommendations for managing subscriptions and friends lists are still being discussed. TBD.

Recommendations for Server Implementations

See component spec: [Recommendations for Server Implementations](https://github.com/solid/solid-spec/blob/master/recommendations-server.md)

Recommendations for Client App Implementations

See component spec: [Recommendations for Client Implementations](https://github.com/solid/solid-spec/blob/master/recommendations-client.md)

Examples

[User Posts a Note](https://github.com/solid/solid-spec/blob/master/examples/user-posts-note.md)

Current Implementations

Server Implementations: See [solid/solid-platform](https://github.com/solid/solid-platform#servers) for a list of Solid servers and developer tools. Note: The Solid team uses[node-solid-server](https://github.com/solid/node-solid-server) as its main server implementation.

Client App Implementations: See [solid-client](https://github.com/solid/solid-client) for the main client library, and [solid/solid-apps](https://github.com/solid/solid-apps) for an example list of Apps built using Solid.

[Contributing to Solid](https://github.com/solid/solid#contributing-to-solid)

[Pre-Requisites](https://github.com/solid/solid#pre-requisites)

[Solid Project Workflow](https://github.com/solid/solid#solid-project-workflow)

[Standards Used](https://github.com/solid/solid#standards-used)

[Platform Notes](https://github.com/solid/solid#solid-platform-notes)

[Solid Project directory](https://github.com/solid/solid#project-directory)

## Legal

Who verifies which Solid Apps are trustworthy?

Who governs and provides the service of being able to search for Apps?

What are the requirements to become a Solid App?

# Finding Contacts

|  |
| --- |
| As a person I want to search for the up to date contact details of people I know. As a person I want to share my up to date contact details with people I know.  |

## Ethics

Should there be a limitation about what a parent or guardian can include in the public facing profile?

## Design

What are the design requirements for a Solid Profile? How should someone be able to find another identity?

## Technical

Profiles: Online Publicly Searchable Identity

Each WebID can optionally be associated to a public facing identity.

The WebID URI's primary function is to point to the location of a public [WebID Profile document](https://github.com/solid/solid-spec#profiles) (see below).

Solid uses WebID Profile Documents for management of user identity and security credentials (such as public keys), and user preferences discovery.

Solid uses WebID Profile Documents for management of user identity and security credentials (such as public keys), and user preferences discovery.

Although here we mostly refer to them in the context of user profiles, other types of actors use these profiles as well, such as groups, organizations, devices, and software applications.

WebID Profile Documents

A WebID URI, when dereferenced, yields a WebID Profile Document in a Linked Data format ([Turtle](http://www.w3.org/TR/turtle/) by default, but often available as JSON-LD or HTML+RDFa). Parsing this document provides a client application with useful information, such as the user's name and profile image, links to user preferences and related documents, and lists of public key certificates or other relevant identity credentials.

See component spec: [Solid WebID Profiles Specification](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md)

[Overview](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#overview)

[Profile Representation Formats](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#profile-representation-formats)

[Required Profile Information](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#required-profile-information)

[Minimum Recommended Profile Information](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#minimum-recommended-profile-information)

[Recommendation for User Names in Profiles](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#recommendation-for-user-names-in-profiles)

[Public and Private Profiles](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#public-and-private-profiles)

[Extended Profile](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#extended-profile)

[Public Key Certificates](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#public-key-certificates)

[Account Resource Discovery](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#account-resource-discovery)

[Storage Discovery](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#storage-discovery)

[Inbox Discovery](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#inbox-discovery)

[Type Registry Index Discovery](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#type-registry-index-discovery)

Profile Representation Formats

From the [WebID Profile](https://www.w3.org/2005/Incubator/webid/spec/identity/#publishing-the-webid-profile-document) spec:

A WebID Profile Document is an RDF Web resource that MUST be available as [text/turtle](http://www.w3.org/TR/turtle/), but MAY be available in other RDF serialization formats (such as JSON-LD or [HTML+RDFa](http://www.w3.org/TR/rdfa-core/)) if requested through content negotiation.

Required Profile Information

There are only 3 statements required for a valid (though not very useful) WebID Profile Document:

Identifying the document as a foaf:PersonalProfileDocument instance

Having a foaf:primaryTopic predicate

Having that primary topic be a valid foaf:Agent type, such as foaf:Person

Here's an example of a minimum valid profile, in Turtle (text/turtle) format:

@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<https://alice.databox.com/profile/card>

 a foaf:PersonalProfileDocument ;

 foaf:primaryTopic <#me> .

<#me>

 a foaf:Person .

Same profile, in JSON-LD (application/ld+json) format:

{

 "@context": {

 "foaf": "http://xmlns.com/foaf/0.1/"

 },

 "@graph": [

 {

 "@id": "https://alice.databox.com/profile/card",

 "@type": "foaf:PersonalProfileDocument",

 "foaf:primaryTopic": {

 "@id": "#me"

 }

 },

 {

 "@id": "#me",

 "@type": "foaf:Person"

 }

 ]

}

Minimum Recommended Profile Information

The above minimal valid profile doesn't provide enough useful information for the purposes of building distributed read-write-web applications. In addition, Solid recommends that WebID profiles include the following statements:

A profile MUST include a foaf:name (see the discussion on [user names](https://github.com/solid/solid-spec/blob/master/solid-webid-profiles.md#recommendation-for-user-names-in-profiles) below). This does not have to be a real name, it can by any pseudonym, but a string provided for apps to use for representing the user, in chats, sharing etc etc.

A profile SHOULD include a public foaf:img of either a mugshot of the person or a chosen avatar to make the display of the user's contributions identifiable.

A profile MAY provide a foaf:nick nickname as a short string for use by user interfaces where space is limited.

A profile SHOULD include cert:key public key certificate information, for use with WebID+TLS (which is currently the primary Solid authentication mechanism).

A profile SHOULD point to the root storage location using pim:storage (so that applications will know where to read and write their data).

@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<https://alice.databox.com/profile/card>

 a foaf:PersonalProfileDocument ;

 foaf:primaryTopic <#me> .

<#me>

 a foaf:Person ;

 foaf:name "Alice" ;

 <http://www.w3.org/ns/auth/cert#key> <#key6b4c> ;

 <http://www.w3.org/ns/pim/space#storage> <../> ;

<#key6b4c>

 # ... certificate key statements go here, see Certificates section

Recommendation for User Names in Profiles

Client-side applications frequently need to know what to name the user, both when interacting with the user directly (such as displaying the currently logged in user in the navigation bar), or talking about users indirectly (an event manager app, when listing which users are invited to a meeting, needs to know how to display their names).

The Solid recommendation for client-side application code, when discovering what to name the user, is to perform the following steps:

An app SHOULD look in the user's WebID Profile for the foaf:name predicate, and use that as the name, if it's available.

If an app does not find a name in the user profile, it MAY fall back to using the WebID URL, or a part of it, as the username.

Public and Private Profiles

Solid servers must be able to support the separation of public and private data in a user's profile. As a result, Solid WebID profiles MAY be split into multiple RDF resources with different read/write permissions, linked together either via owl:sameAs and rdfs:seeAlso predicates, or via the Solid-specific predicate space:preferencesFile.

For example, a typical Solid user would have profile-related statements split across several RDF documents:

/profile/card - their primary (public-readable) WebID Profile. Which would contain a space:preferencesFile link to:

/settings/prefs.ttl - a private (only the user has read/write access) Preferences file which contains further profile-related statements.

Extended Profile

The combination of the main WebID Profile document, and all of the related profile documents is referred to as the **Extended Profile**.

Solid apps that interact anonymously with the WebID profile MUST also load and parse all of the related public RDF resources that are linked to from the main profile using any the following triples in the main profile document:

$webid  http://www.w3.org/2002/07/owl#sameAs ?public

$webid  http://www.w3.org/2000/01/rdf-schema#seeAlso ?public

Solid apps that interact as the user in question, logged in with their credentials, with their own WebID profile MUST also load and parse all of the related public resources above and also will normally load the user's preferences file.

Private preferences file

The private preferences file is part of the extended profile. It is found by following a triple in the main profile (the result of looking up the webid)

$webid  http://www.w3.org/ns/pim/space#preferencesFile ?preferences

Where the subject is the user's original webid.

It is the first private file that the app discovers in this process, and it is the place which either stores, or leads to, all of the data which is private to the user, including settings and preferences, language and display preferences, and so on and all the user's personal data, be it contacts, pictures or health data.

The solid:preferencesFile link is unusual then in that it is a link from public data to private data. Otherwise, discovery happens in two parallel but otherwise congruent ways, in a tree of public information starting from the extended profile, and a tree of private information starting from the private preferences file. Developers are urged to use common software for these cases, and also to make it extensible in future for when the congruent trees may be rooted in files corresponding to groups and organizations of which the user is a member.

Public Key Certificates

Solid currently uses WebID+TLS as its main Authentication mechanism. To enable this, WebID Profile documents on Solid-compliant servers MAY contain one or more Public Key Certificate sections, linked to from the main WebID subject via cert:key predicates.

Example profile with a public key certificate (created by LDNode):

@prefix foaf: <http://xmlns.com/foaf/0.1/>.

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

@prefix cert: <http://www.w3.org/ns/auth/cert#>.

@prefix dc: <http://purl.org/dc/terms/>.

@prefix XML: <http://www.w3.org/2001/XMLSchema#>.

<https://alice.databox.com/profile/card>

 a foaf:PersonalProfileDocument ;

 foaf:primaryTopic <#me> .

<#me>

 a foaf:Person ;

 foaf:name "Alice" ;

 <http://www.w3.org/ns/auth/cert#key> <#key6b4c> ;

 <http://www.w3.org/ns/pim/space#storage> <../> ;

<#key6b4c>

 dc:created

 "2016-02-12T15:07:46.916Z"^^XML:dateTime;

 dc:title

 "Created by ldnode";

 a cert:RSAPublicKey;

 rdfs:label

 "LDNode Localhost Test Cert";

 cert:exponent

 "65537"^^XML:int;

 cert:modulus

 "970E88..(many digits here)..167801"^^XML:hexBinary.

Account Resource Discovery

Solid WebID Profile documents MAY contain the following links, to support the discovery of resources that are of interest to client side applications.

Storage Discovery

A Solid WebID Profile SHOULD contain a link to one or more Solid Containers that act as Storage (a space for apps to read and write data).

Example link to Root Storage (gets created [by default](https://github.com/solid/solid-spec/blob/master/recommendations-server.md#default-containers) on account creation):

# ...

<#me>

 a foaf:Person ;

 <http://www.w3.org/ns/pim/space#storage> <../> .

Inbox Discovery

A Solid WebID Profile MAY contain a link to the Solid Inbox container (gets created [by default](https://github.com/solid/solid-spec/blob/master/recommendations-server.md#default-containers) on account creation).

If an inbox link exists, there MUST be only one Inbox for the profile.

Example:

# ...

<#me>

 a foaf:Person ;

 <http://www.w3.org/ns/ldp#inbox> </inbox/> .

Type Registry Index Discovery

A Solid WebID Profile SHOULD contain one or more links to [Type Registry Index](https://github.com/solid/solid/blob/master/proposals/data-discovery.md) resources.

If links to type indexes exist, there MUST be only one link each to a private and a public type registry index file, respectively.

For example, a link to the Listed Type Index in the main profile document:

# ...

<#me>

 a foaf:Person ;

 <http://www.w3.org/ns/solid/terms#publicTypeIndex>

 </settings/publicTypeIndex.ttl> .

And an example corresponding link to the Unlisted Type Index, in a private resources of the Extended Profile, such as the Preferences file (in /settings/prefs.ttl):

# ...

<#me>

 <http://www.w3.org/ns/solid/terms#privateTypeIndex>

 </settings/privateTypeIndex.ttl> .

## Legal