Demanding Interoperability to Strengthen the Free (Libre) Web: Introducing Disfluid

for version 0.6.1, 21 October 2021

Vivien Kraus (vivien@planete-kraus.eu)

This is the manual of disfluid (version 0.6.1, 21 October 2021), an implementation of the Solid authentication protocol for guile, client and server.

Table of Contents

Disfluid is an independent implementation of a web stack focusing on interoperability. In this implementation, the users control what programs run in their computers. They also choose who to trust for online data storage and processing, without needing any permission, or can self-host their data.

The software is available at `https://labo.planete-kraus.eu/webid-oidc.git`. The latest commit is tracked in the Guix channel `https://labo.planete-kraus.eu/webid-oidc-channel.git`.

1 Decentralized Authentication on the Web

Authentication on the web is currently handled in the following way: anyone can install a server that will authenticate users on the web. The problem is interoperability. If a client (an application) wants to authenticate a user, it has to be approved by the authentication server. In other words, if *useful-program* wants to authenticate *MegaCorp* users, then *useful-program* has to register to *MegaCorp* first, and get approved. This goes against the principle of permission-less innovation, which is at the heart of the web.

In the decentralized authentication web, the best attempt so far is that of ActivityPub. All servers are interoperable with respect to authentication: if user A emits an activity, it is forwarded by A's server to its recipients, and A's server is responsible for A's identity.

The problem with that approach is that the data is tied to the application. It is not possible to use another application to process the data differently, or to use multiple data sources, in an interoperable way (without the ActivityPub server knowing). This means that on Activitypub, microblogging applications will not present different activities correctly. This also means that it is difficult to write a free replacement to a non-free application program, because it would need to manage the data.

In the Solid ecosystem, there is a clear distinction between servers and applications. An application is free to read data from all places at the same time, using a permission-less authentication system. Since the applications do not need to store data, the cost of having users is neglectible, so users do not need prior approval before using them (making captchas and the like a thing of the past). Servers do not have a say in which applications the user uses.

The authentication used is a slight modification of the well-established OpenID Connect. It is intended to work in a web browser, but this package demonstrates that it also works without a web browser.

2 Invoking disfluid

The '`disfluid`' program runs a server, if the user specifies a configuration file, or the graphical browser otherwise.

2.1 General options

The server will respond to '`-h`' and '`-v`' commands, to get the help output and the version information.

The server output (command-line, logs) are localized for the system administrator. You can control it with the '`LANG`' environment variable. So if your locale is not English, you can have the same commands as in this manual by running with `LANG=C`.

The programs respect the '`XDG_DATA_HOME`' (if not overriden by the server configuration) and '`XDG_CACHE_HOME`' to store persistent data and disposable data. The cache directory can be deleted at any time. If one of these variables is not set, its value is computed from the '`HOME`' environment variable.

2.2 Running a server

The disfluid code is published under the Affero GPL, which means that the service provider needs to publish all changes made to the program to users over the network. The '`disfluid`' command provides a '`--complete-corresponding-source`' option so that the system administrator can specify a means to download the source.

The servers will add a '`Source:`' header in each response, containing the value of this configuration option. It can be, for instance, an URI where to download the modified source code.

The servers can be configured to redirect output and errors to a log file and an error file, with the '`--log-file`' and '`--error-file`' options.

The server will listen to port 8080 by default, but this may be configured with '`--port`'. Since the servers do not support TLS, and they only support HTTP/1.1, they are intended to run behind a reverse proxy (even for the authenticating reverse proxy).

Finally, you configure the server by passing the '`--configuration`' parameter pointing to a configuration file. The configuration file is plain guile code, that must evaluate to an `<endpoint>`.

Here is an example configuration that runs a resource server with an identity provider:

```
(use-modules (webid-oidc server endpoint)
             (webid-oidc server endpoint resource-server)
             (webid-oidc server endpoint identity-provider)
             (webid-oidc server endpoint authentication)
             (webid-oidc oidc-configuration)
             (oop goops))

(make <identity-provider>
  #:host "example.com"
  #:oidc-discovery
  (make <oidc-discovery>
```

```
  #:path "/.well-known/openid-configuration"
  #:configuration
  (make <oidc-configuration>
    #:jwks-uri "https://example.com/keys"
    #:authorization-endpoint "https://example.com/authorize"
    #:token-endpoint "https://example.com/token"))
#:authorization-endpoint
(make <authorization-endpoint>
  #:path "/authorize"
  #:subject "https://example.com/profile/card#me"
  #:encrypted-password (crypt "secretpassword123" "$6$secret.salt")
  #:key-file "/var/lib/disfluid/key-file.jwk")
#:token-endpoint
(make <token-endpoint>
  #:path "/token"
  #:issuer "https://example.com"
  #:key-file "/var/lib/disfluid/key-file.jwk")
#:jwks-endpoint
(make <jwks-endpoint>
  #:path "/keys"
  #:key-file "/var/lib/disfluid/key-file.jwk")
#:default
(make <authenticator>
  #:backend
  (make <resource-server>
    #:server-name "https://example.com"
    #:owner "https://example.com/profile/card#me")
  #:server-uri "https://example.com"))
```

The server will make requests on the world-wide web, for instance to download client manifests. The requests can be redirected with XML Catalog, by setting the '`XML_CATALOG_FILES`' to a space-separated list of URIs (can be `file:` URIs). The requests cannot be directed to the file system.

3 Running disfluid with GNU Guix

The channel at `https://labo.planete-kraus.eu/webid-oidc-channel.git` can be used with guix. It defines the package at the latest commit, and a service definition in *(vkraus services disfluid)*.

`disfluid-service-type` [service type]

This service runs a disfluid server with the *disfluid* system user. The value it takes is a service configuration.

`<disfluid-configuration>` [*disfluid*] [configuration record]
complete-corresponding-source port configuration-file [*extra-options*]

The configuration for the identity provider. The optional *disfluid* argument is the package containing the binary to run, if you want to apply some patches, and *extra-options* is an empty list by default.

configuration-file is a file-like object or a file name.

4 Common parameters

The *(webid-oidc parameters)* module provides a set of Guile parameter to control the program behavior.

`data-home` [parameter]
: This parameter controls the location where the program stores persistent data. By default, it is located in `XDG_DATA_HOME`.

`cache-home` [parameter]
: This parameter controls the location where the program stores data that might get deleted at any time. By default, it uses `XDG_CACHE_HOME`.

`current-date` [parameter]
: This parameter is a thunk similar to SRFI-19 `current-date`, except it can be set with a thunk returning a date, time or number of seconds, or a date, time or number of seconds.

`anonymous-http-request` [parameter]
: This parameter is a function similar to the `http-request` function in *(web client)*.

`authorization-code-default-validity` [parameter]
: This parameter controls the number of seconds for which an authorization code is valid at creation time.

`oidc-token-default-validity` [parameter]
: This parameter controls the number of seconds for which an ID token or access token is valid at creation time.

`dpop-proof-validity` [parameter]
: This parameter controls the number of seconds for which a DPoP proof is valid after it has been issued.

5 Managing keys

Some functions require a key, or a key pair, to operate. The *(webid-oidc jwk)* module provides you with everything required to manage keys.

`<private-key>` () *alg* [Class]
: This is the base class for a private key. You need it to issue signatures. Signatures issued with this key will use *alg* for the signature algorithm, but the public key associated with this private key will verify signatures in any compatible algorithm, not just *alg*.

  alg is a symbol, for instance `'RS256`.

`<public-key>` () [Class]
: This is the base class for a public key. You need it to check signatures.

`<key-pair>` () *public-key private-key* [Class]
: A key pair contains a *public-key* and a matching *private-key*. You use this form for keys you own.

`<rsa-key-pair>` () (`<key-pair>`) [Class]
: This key pair contains matching RSA keys.

`<ec-key-pair>` () (`<key-pair>`) *crv* [Class]
: This key pair contains matching elliptic curve keys. *crv* is a symbol identifiying the curve.

`<rsa-private-key>` (`<`*private-key*`>`) *d p q dp dq qi* [Class]
`<rsa-public-key>` (`<`*public-key*`>`) *n e* [Class]
`<ec-scalar>` (`<`*private-key*`>`) *crv z* [Class]
`<ec-point>` (`<`*public-key*`>`) *crv x y* [Class]
: All fields are strings, base64 encoding the parameters, except *crv*, which is a symbol.

`<jwks>` () *keys* [Class]
: An identity provider may use different keys that are in validity to sign different access tokens. The JWKS encapsulates many public *keys*.

`<public-key> public-key` (*key* `<key-pair>`) [Generic method]
`<public-key> public-key` (*key* `<public-key>`) [Generic method]
: Return the public part of *key*, which may either be a key pair or a public key.

`<private-key> private-key` (*key* `<key-pair>`) [Generic method]
`<private-key> private-key` (*key* `<private-key>`) [Generic method]
: Return the private part of *key*.

`<string> rsa-d` (*key* `<rsa-key-pair>`) [Generic method]
`<string> rsa-d` (*key* `<rsa-private-key>`) [Generic method]
`<string> rsa-p` (*key* `<rsa-key-pair>`) [Generic method]
`<string> rsa-p` (*key* `<rsa-private-key>`) [Generic method]
`<string> rsa-q` (*key* `<rsa-key-pair>`) [Generic method]
`<string> rsa-q` (*key* `<rsa-private-key>`) [Generic method]

`<string> rsa-dp (key <rsa-key-pair>)` [Generic method]
`<string> rsa-dp (key <rsa-private-key>)` [Generic method]
`<string> rsa-dq (key <rsa-key-pair>)` [Generic method]
`<string> rsa-dq (key <rsa-private-key>)` [Generic method]
`<string> rsa-qi (key <rsa-key-pair>)` [Generic method]
`<string> rsa-qi (key <rsa-private-key>)` [Generic method]
`<string> rsa-n (key <rsa-key-pair>)` [Generic method]
`<string> rsa-n (key <rsa-public-key>)` [Generic method]
`<string> rsa-e (key <rsa-key-pair>)` [Generic method]
`<string> rsa-e (key <rsa-public-key>)` [Generic method]
`<symbol> ec-crv (key <ec-key-pair>)` [Generic method]
`<symbol> ec-crv (key <ec-point>)` [Generic method]
`<symbol> ec-crv (key <ec-scalar>)` [Generic method]
`<string> ec-x (key <ec-key-pair>)` [Generic method]
`<string> ec-x (key <ec-point>)` [Generic method]
`<string> ec-y (key <ec-key-pair>)` [Generic method]
`<string> ec-y (key <ec-point>)` [Generic method]
`<string> ec-z (key <ec-key-pair>)` [Generic method]
`<string> ec-z (key <ec-scalar>)` [Generic method]
`<symbol> alg (key <key-pair>)` [Generic method]
`<symbol> alg (key <private-key>)` [Generic method]

Key parameter getters.

`<list> keys (jwks <jwks>)` [Generic method]

Return all the public keys used by *jwks*.

`<undefined> check-key (key <public-key>)` [Generic method]
`<undefined> check-key (key <private-key>)` [Generic method]
`<undefined> check-key (key <key-pair>)` [Generic method]

Check that the *key* parameters are consistent.

When exchanging keys, maybe you will have them in the form of a JWK: an alist from symbols to strings, as a representation for a JSON object.

`<list> key->jwk (key <public-key>)` [Generic method]
`<list> key->jwk (key <private-key>)` [Generic method]
`<list> key->jwk (key <key-pair>)` [Generic method]

Return an alist with known parameter names for JSON.

`jwk->key jwk` [function]

Parse *jwk* as a key or a key pair.

`<symbol> kty (key <rsa-key-pair>)` [Generic method]
`<symbol> kty (key <rsa-public-key>)` [Generic method]
`<symbol> kty (key <rsa-private-key>)` [Generic method]
`<symbol> kty (key <ec-key-pair>)` [Generic method]
`<symbol> kty (key <ec-point>)` [Generic method]
`<symbol> kty (key <ec-scalar>)` [Generic method]

Return `'RSA` for RSA keys, or `'EC` for elliptic curve keys.

`<string> jkt (key <key-pair>)` [Generic method]
`<string> jkt (key <public-key>)` [Generic method]
Hash the *key* parameters in a reproducible order to get the hash of a key.

`generate-key [#:n-size] [#:e-size] [#:e=\"AQAB\"] [#:crv]` [function]
Generate a new key pair.

`<values> serve (jwks <jwks>) expiration-date` [Generic method]
Return a response and response body for serving *jwks*. Client-side caching is very much necessary for a JWKS, so pass *expiration-date* as a SRFI-19 date to define a maximum date for caching. It should be in the future, for instance in 1 hour.

`get-jwks uri [#:http-request]` [function]
Download a JWKS on the web at *uri*. Use *http-request*, with the same interface as that of *(web client)*, to actually get the JWKS.

`&not-a-jwk` [Exception type]
If the key parameters are incorrect, this exception is raised.

`&not-a-jwks` [Exception type]
If the JWKS cannot be downloaded, or is incorrect, this exception is raised.

6 OIDC discovery

An identity provider is known by its server name. The different endpoints can be discovered from there.

`<oidc-configuration>` () *jwks-uri authorization-endpoint token-endpoint* [Class]

The OIDC configuration for an identity provider. *jwks-uri*, *authorization-endpoint* adn *token-endpoint* are all URIs.

You can construct an OIDC configuration two different ways:

- by passing `#:jwks-uri`, `#:authorization-endpoint` and `#:token-endpoint` to the constructor;
- by passing `#:server`, and optionally `#:http-request` to the constructor, to query the *server* for its configuration.

`&invalid-oidc-configuration` [Exception type]

This exception is raised when the configuration is unusable or incomplete.

`make-invalid-oidc-configuration` [function]

Constructor for the `&invalid-oidc-configuration` exception type.

`invalid-oidc-configuration?` *exception* [function]

Check whether *exception* was raised because of an invalid OIDC configuration.

`jwks-uri` *oidc-configuration* [Generic]

Return the JWKS uri of *oidc-configuration*.

`jwks` *oidc-configuration* [Generic]

Query the JWKS uri of *oidc-configuration*.

`authorization-endpoint` *oidc-configuration* [Generic]

Return the authorization endpoint of *oidc-configuration*.

`token-endpoint` *oidc-configuration* [Generic]

Return the token endpoint of *oidc-configuration*.

`serve` *configuration expiration-date* [Generic]

Return 2 values: the response, and response body, needed to serve *configuration*. It is very much recommended to let clients cache this value. They will not revalidate it until after *expiration-date*, a SRFI-19 date.

7 Client manifest

To make sure that a client application is legitimate, it is mandated that it serves a public document under its ID URI, and that document should confirm the URI and the redirection URI, where the client application gets the authorization code.

`<client-manifest>` () *client-id redirect-uris* [Class]
: This is the class encapsulating a very basic client manifest. *client-id* is an URI, and *redirect-uris* is a list of URIs.

  You can construct one by providing both `#:client-id` and `#:redirect-uris`, or by providing only `#:client-id`, in which case it will be downloaded from the web.

Clients that cannot serve pages should use the anonymous client ID, that accepts all redirect URIs.

`client-id` *manifest* [Generic]
: Return the client ID of *manifest*.

`redirect-uris` *manifest* [Generic]
: Return the list of accepted redirection URIs for *manifest*.

`->json-data` *manifest* [Generic]
: Convert *manifest* to JSON data (alists for objects, vectors for arrays). You should override this method if you design an extended client manifest class.

`check-redirect-uri` *manifest uri* [Generic]
: Check that *manifest* controls *uri*, where to send the authorization code. Raises an exception if that's not the case.

`&invalid-client-manifest` [Exception type]
: This exception is raised when the client manifest is invalid.

`make-invalid-client-manifest` [function]
: Constructor for the `&invalid-client-manifest` exception type.

`invalid-client-manifest?` *exception* [function]
: Check whether *exception* was raised because of an invalid client manifest.

`&unauthorized-redirect-uri` [Exception type]
: This exception is raised when the requested authorization URI is unauthorized.

`make-unauthorized-redirect-uri` [function]
: Constructor for the `&unauthorized-redirect-uri` exception type.

`unauthorized-redirect-uri?` *exception* [function]
: Check whether *exception* was raised because of an unauthorized redirection URI.

`&inconsistent-client-manifest` [Exception type]
: This exception is raised when the client ID does not match what the client manifest says.

`make-inconsistent-client-manifest` [function]
Constructor for the `&inconsistent-client-manifest` exception type.

`inconsistent-client-manifest?` *`exception`* [function]
Check whether *exception* was raised because of an inconsistent client manifest.

`&cannot-serve-public-manifest` [Exception type]
This exception is raised when the manifest to serve has the public client URI as ID.

`make-cannot-serve-public-manifest` [function]
Constructor for the `&cannot-serve-public-manifest` exception type.

`cannot-serve-public-manifest?` *`exception`* [function]
Check whether *exception* was raised because the server wants to serve a public manifest.

`&cannot-fetch-client-manifest` [Exception type]
This exception is raised when the server does not behave correctly when fetching the manifest.

`make-cannot-fetch-client-manifest` [function]
Constructor for the `&cannot-fetch-client-manifest` exception type.

`cannot-fetch-client-manifest?` *`exception`* [function]
Check whether *exception* was raised because we could not fetch a client manifest.

8 The Json Web Token

The Json Web Token, or *JWT*, is a terse representation of a pair of JSON objects: the *header*, and the *payload*. The JWT can be *encoded* as a Json Web Signature (*JWS*), in which case the header is encoded to base64 with the URL alphabet, and without padding characters, the payload is also encoded to base64, and the concatenation of the encoding of the header, a dot, and the encoding of the payload is signed with some cryptography algorithm. In the following, we will only be interested by public-key cryptography. The concatenation of header, dot, payload, dot and signature in base64 is the encoding of the JWT.

8.1 Tokens

The *(webid-oidc jws)* implements some functionality for tokens.

`<token>` () *alg* [Class]
: The base class for all tokens. It only knows the signature *alg*orithm. You can construct one in different ways:
  - the `#:alg` construct keyword supports a string or a keyword as a value, containing a valid JWA identifier, such as `RS256`;
  - the `#:signing-key` keyword defines the key that will serve to sign the token. The signature algorithm is set to the default of *signing-key*;
  - the `#:jwt-header` and `#:jwt-payload` keywords let you pass two alists, following the JSON representation from srfi-180: objects are alists of **symbols** to values, arrays are vectors.

`&invalid-jws` [Exception type]
: This exception is raised when a JWT cannot be parsed or constructed as a JWS.

`make-invalid-jws` [function]
: Construct an exception of type `&invalid-jws`.

`invalid-jws?` *exception* [function]
: Check whether *exception* was raised because of an invalid JWS.

There are multiple things you can do with a token.

`alg` *token* [Generic]
: Return the signature algorithm used for *token*, as a symbol.

`token->jwt` *token* [Generic]
: Return two alists, following the JSON representation from srfi-180: one for the header, and then one for the payload.

`lookup-keys` *token args* [Generic]
: Return the set of keys that could be used to sign *token*, as a public key, a list of keys, or a JWKS. *args* is a list of keyword arguments for specific implementations.

`verify` *token args* [Generic]
: Suppose that the *token* signature has been checked, perform some additional verifications. This function should raise exceptions to signal an invalid token.

`decode` *`expected-token-class encoded . args`* [function]
: Parse *encoded* as a token from the *expected-token-class*, check its signature against the key obtained by `(lookup-keys` *`token args`*`)` where *token* is the parsed token, and perform additional verifications with `(verify` *`token args`*`)`.

`encode` *`token key`* [function]
: Encode and sign *token* with *key*, returning a string.

`issue` *`token-class issuer-key . args`* [function]
: Construct a token of *token-class* and *args* and sign it with *issuer-key*. Since we know the key to sign it, it is not necessary to pass either `#:signing-key` nor `#:alg` to the constructor.

8.2 Tokens issued by an OIDC provider

OIDC tokens are those signed by an OIDC identity provider. This kind of token knows its issuer, and getting the keys to check the token signature is done by OIDC discovery.

`<oidc-token>` (`<`*token*`>`) *`iss`* [Class]
: The base class for tokens which are issued by an identity provider. It knows the issuer (*iss*, an uri from *(web uri)*), and can query it to check the token signature.

  Similarly to the base token type, you can construct one by specifying its arguments, or create one from a pair of alists.

  - `#:`*`alg`* or `#:`*`signing-key`* is required to construct the base token;
  - `#:`*`iss`* specifies the issuer.

  The main point of this class is to provide a method for the `lookup-keys` generic. This method accepts one keyword argument, `#:`*`http-request`*, a function that behaves like the web client in *(web client)*. You can set this value as a keyword argument in the `decode` function.

`iss` *`token`* [Generic]
: Return the issuer of *token*, as an URI.

`&cannot-query-identity-provider` *`identity-provider`* [Exception type]
: This exception is raised when the OIDC discovery fails. *identity-provider* is an URI.

`make-cannot-query-identity-provider` *`identity-provider`* [function]
: Construct an exception of type `&cannot-query-identity-provider`.

`cannot-query-identity-provider?` *`exception`* [function]
: Check whether *exception* was raised because an identity provider could not be queried.

`cannot-query-identity-provider-value` *`exception`* [function]
: Return the faulty identity provider for *exception*.

8.3 Date verification for tokens

Different kinds of tokens have a requirement for a limited time window for which the signature should be valid.

`<time-bound-token>` (`<`*token*`>`) `iat` *exp* [Class]

The base class for tokens which are issued for a limited time window. It knows the issuance date (*iat*, a date from *(srfi srfi-19)*), and the expiration date (*iat*, a date from *(srfi srfi-19)*).

Similarly to the base token type, you can construct one by specifying its arguments, or create one from a pair of alists.

- `#:alg` or `#:signing-key` is required to construct the base token;
- `#:iat` specifies the issuance date. It defaults to the current date;
- `#:exp` specifies the expiration date. If it is not set, the value will be computed from *iat* and *validity*;
- `#:validity` is used when the expiration date is not known in advance. It is a number of seconds. For a DPoP proof, the value should be around 30 seconds. For an access token, a good value is in the ballpark of 3600 seconds (an hour). Defaults to 3600 seconds, but be aware that for single-use tokens, this value will be ignored and replaced with a much shorter time.

The main point of this class is to provide a stricter token validation function. You can customize the current date by passing `#:current-date ...` as keyword arguments to `decode`. `...` would be replaced with a time or date.

`default-validity` *token* [Generic]

Return the default validity as a number of seconds to construct *token*, or `#f` if an explicit `#:validity` is required.

`has-explicit-exp?` *token* [Generic]

Check whether we should trust the JWT exp field when constructing *token*. DPoP proofs should not be able to fill our cache with infinitely-valid proofs, so it is disabled for DPoP proofs.

`iat` *token* [Generic]

Return the signature date of *token*, as a srfi-19 date.

`exp` *token* [Generic]

Return the expiration date of *token*, as a srfi-19 date.

`&signed-in-future` *signature-date current-date* [Exception type]
`&expired` *expiration-date current-date* [Exception type]

An exception of type `&signed-in-future` is raised when the current date is before the alleged signature date. Since the signing entity and the verifier entity may not be on the same system, the clocks may be slightly out of synchronization, so a margin of 5 seconds is usually accepted.

An exception of type `&expired` indicates that the signature is no longer valid.

`make-signed-in-future` *`signature-date current-date`* [function]
`make-expired` *`expiration-date current-date`* [function]
Constructors for the `&signed-in-future` and `&expired` exception types.

`signed-in-future?` *`exception`* [function]
`expired?` *`exception`* [function]
Check whether *exception* was raised because of a date mismatch.

`error-signature-date` *`exception`* [function]
`error-expiration-date` *`exception`* [function]
`error-current-date` *`exception`* [function]
If *exception* was raised because of a date mismatch, return the signature, expiration or current date.

8.4 Single-use tokens

To prevent replay attacks, you might want to assign an unique identifier to each token of some kind. If you have an expiration date, you could remember that this identifier has been seen, and forget about it as soon as the token expires. For this to work, you would need an expiration date for your single-use token: this is why we only support it for time-bound tokens, and the validity is reduced down to 2 minutes.

`<single-use-token>` (`<`*time-bound-token*`>`) *`nonce`* [Class]
The base class for tokens which are intended to be decoded only once. The unique identifier string *nonce* will be remembered as long as the program is running and the token is not expired.

Similarly to the base token type, you can construct one by specifying its arguments, or create one from a pair of alists.

- `#:alg` or `#:signing-key` is required to construct the base token;
- `#:iat` and `#:exp` or `#:validity` is required to construct the time-bound token;
- `#:nonce` specifies the unique identifier. It defaults to a random string of base64 data encoding 96 bits of entropy.
-

The main point of this class is to provide an even stricter token validation function, that can only be run once for a given token (with reasonable limits: if the program is killed, it won't remember the tokens from before). You can customize the current date by passing `#:current-date ...` as keyword arguments to `decode`, just as you do for regular time-bound tokens. `...` would be replaced with a time or date.

`nonce-field-name` *`token`* [Generic]
When constructing *token* from an existing JWT, this method gives the field name in the JWT payload that represents the nonce. DPoP proofs use `'jti`, so they override this value.

`nonce` *`token`* [Generic]
Return the unique identifier of *token*, as a string.

`&nonce-found` *nonce* [Exception type]
: If a token with the same nonce has already been decoded during its life time, this exception is raised with the duplicated *nonce*.

`make-nonce-found` *nonce* [function]
: Construct an exception of type `&nonce-found`.

`nonce-found?` *exception* [function]
: Check whether *exception* was raised because a single-use token was already parsed.

`nonce-found-nonce` *exception* [function]
: Return the faulty nonce in *exception*.

8.5 ID tokens

The *(webid-oidc oidc-id-token)* module contains a definition for the OIDC ID token.

`<id-token>` (`<`*single-use-token*`>` `<`*oidc-token*`>`) `webid sub aud` [Class]
: The ID token is issued by an identity provider, and is intended to be used by the client only. It gives information about the user identified by a *webid*, an URI from *(web uri)*, and the client ID, *aud*, an URI too. Since the client should not communicate this token, it is reasonable to think that the client will deccode the token as soon as it gets it, and then forget the now useless signature. This is why this token is considered single-use. The *sub* field should store a username as a string, but if it is missing, the webid (as a string) will be used.

  To construct an ID token, you would either need `#:jwt-header` and `#:jwt-payload`, as for any token, or a combination of parameters:

  - `#:alg` or `#:signing-key`, to initialize a JWT;
  - `#:iat` and `#:exp` or `#:validity`, because it is issued for a limited time window (around an hour);
  - `#:nonce` to define its identifier (defaults to a random one);
  - `#:iss`, the issuer URI, because it is an OIDC token;
  - `#:webid`, an URI identifying the user;
  - `#:sub`, a string that defaults to the webid;
  - `#:aud`, an URI identifying the application.

`webid` *token* [Generic]
: Return the user identifier in *token*, as an URI.

`sub` *token* [Generic]
: Return the username in *token*, as a string.

`aud` *token* [Generic]
: Return the client identifier in *token*, as an URI.

`&invalid-id-token` [Exception type]
: This exception is raised when the ID token is invalid.

`make-invalid-id-token` [function]
Construct an exception of type `&invalid-id-token`.

`invalid-id-token?` *exception* [function]
Check whether *exception* was raised because of an invalid ID token.

8.6 Access tokens

The *(webid-oidc access-token)* module contains a definition for the OIDC access token.

`<access-token>` (`<`*time-bound-token*`>` `<`*oidc-token*`>`) *webid aud client-id cnf/jkt* [Class]
The access token is issued by an identity provider for a client, and is intended to be used by the resource servers. It indicates that the agent possessing a key hashed to *cnf/jkt* (a string) is identified by *client-id* (an URI) and is authorized to act on behalf of the user identified by *webid* (an URI). For compatibility, *aud* should be set to the literal string `"solid"`. The agent demonstrates that it owns this key by issuing a DPoP proof alongside the access token.

To construct an access token, you would either need `#:jwt-header` and `#:jwt-payload`, as for any token, or a combination of parameters:

- `#:alg` or `#:signing-key`, to initialize a JWT;
- `#:iat` and `#:exp` or `#:validity`, because it is issued for a limited time window (around an hour);
- `#:iss`, the issuer URI, because it is an OIDC token;
- `#:webid`, an URI identifying the user;
- `#:client-id`, an URI identifying the client;
- `#:cnf/jkt`, the hash of a public key whose private key is owned by the client, or `#:client-key`, the client key itself;
- `#:aud`, literally `"solid"`, optional, defaults to the correct value.

Since the same access token is presented on each request, it is not single-use.

`webid` *token* [Generic]
Return the user identifier in *token*, as an URI.

`client-id` *token* [Generic]
Return the client identifier in *token*, as an URI.

`cnf/jkt` *token* [Generic]
Return the hash of the client key, as a string.

`aud` *token* [Generic]
Return `"solid"`.

`&invalid-access-token` [Exception type]
This exception is raised when the access token is invalid.

`make-invalid-access-token` [function]
Construct an exception of type `&invalid-access-token`.

`invalid-access-token?` *exception* [function]
Check whether *exception* was raised because of an invalid access token.

8.7 DPoP proofs

The *(webid-oidc dpop-proof)* module contains a definition for the DPoP proof token.

`<dpop-proof>` (`<`*single-use-token*`>`) *typ jwk htm htu ath* [Class]
The DPoP proof is a token that is issued by the client, and presented to the resource server along with an access token. It is only valid for one request, and for one use. So, it should have a very short validity frame, for instance 30 seconds, and should only be valid for a specific request method *htm* and a specific request URI *htu*, down to the path, but ignoring the query and fragment.

The DPoP proof is the proof of possession of *jwk*, a public key. It should always have a *typ* field set to `"dpop+jwt"`.

To construct a DPoP proof, you would either need `#:jwt-header` and `#:jwt-payload`, as for any token, or a combination of parameters:

- `#:alg` or `#:signing-key`, to initialize a JWT;
- `#:iat` and `#:exp` or `#:validity`, because it is issued for a limited time window (around 30 seconds);
- `#:nonce`, because it is single-use;
- `#:jwk`, the public key whose possession we demonstrate by signing the proof;
- `#:htm`, the HTTP method used (as a symbol);
- `#:htu`, the HTTP URI used (as an URI);
- `#:ath`, the hash of the access token that goes with this proof, or `#:access-token`, the encoded access token itself, if the proof goes with an access token. Otherwise, pass `#f`. Defaults to `#f`;
- `#:typ`, literally `"dpop+jwt"`, optional, defaults to the correct value.

This token class makes a stricter verification function. It requires you to set as a keyword argument in `decode` the following parameters:

`#:access-token`
: set the access token that should go with the proof, defaults to `#f` (no access token);

`#:method` set the method used for the proof;

`#:uri` set the URI used for the proof;

`#:cnf/check`
: set the expected hash of the key used by the DPoP proof, or a function taking a public key hash. If this is a function, it should raise an exception if the hash is invalid, because its return value is ignored.

`jwk` *proof* [Generic]
Return the public key whose possession *proof* demonstrates.

`htm` *proof* [Generic]
Return the HTTP method in *proof*, as a symbol.

`htu` *proof* [Generic]
Return the HTTP URI in *proof*, as an URI.

`ath` *proof* [Generic]
Return the hash of the access token that should go with *proof*, or `#f` if *proof* is not used with an access token.

`typ` *proof* [Generic]
Return `"dpop+jwt"`.

`&invalid-dpop-proof` [Exception type]
This exception is raised when the DPoP proof is invalid.

`make-invalid-dpop-proof` [function]
Construct an exception of type `&invalid-dpop-proof`.

`invalid-dpop-proof?` *exception* [function]
Check whether *exception* was raised because of an invalid DPoP proof.

`&dpop-method-mismatch` *advertised actual* [Exception type]
This exception is raised when the *advertised* method is not what is *actual*ly used in the request (both symbols).

`make-dpop-method-mismatch` *advertised actual* [function]
Construct an exception of type `&dpop-method-mismatch`.

`dpop-method-mismatch?` *exception* [function]
Check whether *exception* was raised because of a difference between the advertised and actual HTTP methods used.

`dpop-method-mismatch-advertised` *exception* [function]
In case of a DPoP method mismatch causing *exception*, return the method used in the proof signature.

`dpop-method-mismatch-actual` *exception* [function]
In case of a DPoP method mismatch causing *exception*, return the method that the server received.

`&dpop-uri-mismatch` *advertised actual* [Exception type]
This exception is raised when the *advertised* URI is not what is *actual*ly used in the request (both URIs).

`make-dpop-uri-mismatch` *advertised actual* [function]
Construct an exception of type `&dpop-uri-mismatch`.

`dpop-uri-mismatch?` *exception* [function]
Check whether *exception* was raised because of a difference between the advertised and actual HTTP URIs used.

`dpop-uri-mismatch-advertised` *exception* [function]
In case of a DPoP URI mismatch causing *exception*, return the URI used in the proof signature.

`dpop-uri-mismatch-actual` *exception* [function]
In case of a DPoP URI mismatch causing *exception*, return the URI that the server received.

`&dpop-invalid-ath` *hash access-token* [Exception type]
This exception is raised when the DPoP proof is intended for use along with an access token identified by *hash*, but is actually used along with *access-token*.

`make-dpop-invalid-ath` *hash access-token* [function]
Construct an exception of type `&dpop-invalid-ath`.

`dpop-invalid-ath?` *exception* [function]
Check whether *exception* was raised because the DPoP proof was not used with the correct access token.

`dpop-invalid-ath-hash` *exception* [function]
In case of a DPoP presented with the wrong access token, causing *exception*, return the hash of the intended access token.

`dpop-invalid-ath-access-token` *exception* [function]
In case of a DPoP presented with the wrong access token, causing *exception*, return the actual access token.

`&dpop-unconfirmed-key` [Exception type]
This exception is raised when the DPoP proof does not demonstrate the possession of the correct key.

`make-dpop-unconfirmed-key` [function]
Construct an exception of type `&dpop-unconfirmed-key`.

`dpop-unconfirmed-key?` *exception* [function]
Check whether *exception* was raised because the DPoP proof demonstrated the possession of an incorrect key.

8.8 Authorization codes

(webid-oidc authorization-code) defines an authorization code type.

`<authorization-code>` (`<`*single-use-token*`>`) *webid client-id* [Class]
While it is not necessary that an authorization code is a JWT, it is easier to implement that way. It is an authorization for *client-id*, an URI identifying a client, to access the data of the user identified by *webid*, an URI too. It should only be valid for a limited amount of time, and used once only.

The DPoP proof is a token that is issued by the client, and presented to the resource server along with an access token. It is only valid for one request, and for one use. So, it should have a very short validity frame, for instance 30 seconds, and should

only be valid for a specific request method *htm* and a specific request URI *htu*, down to the path, but ignoring the query and fragment.

The DPoP proof is the proof of possession of *jwk*, a public key. It should always have a *typ* field set to `"dpop+jwt"`.

To construct an authorization code, you would either need `#:jwt-header` and `#:jwt-payload`, as for any token, or a combination of parameters:

- `#:alg` or `#:signing-key`, to initialize a JWT;
- `#:iat` and `#:exp` or `#:validity`, because it is issued for a limited time window (around 30 seconds);
- `#:nonce`, because it is single-use;
- `#:webid`, the user identifier;
- `#:client-id`, the client identifier.

The authorization code is signed and verified by the same entity. So, the key lookup function is tuned to always return the issuer key. You need to set it as the `#:issuer-key` keyword argument of the `decode` function.

`webid` *token* [Generic]
Return the user identifier in *token*, as an URI.

`client-id` *token* [Generic]
Return the client identifier in *token*, as an URI.

`&invalid-authorization-code` [Exception type]
This exception is raised when the authorization ccode is invalid.

`make-invalid-authorization-code` [function]
Construct an exception of type `&invalid-authorization-code`.

`invalid-authorization-code?` *exception* [function]
Check whether *exception* was raised because of an invalid authorization code.

9 Caching on server side

Both the identity provider and the resource server need to cache things. The identity provider will cache application webids, and the resource server will cache the identity provider keys, for instance.

The solution is to use a file-system cache. Every response (except those that have a cache-control policy of no-store) are stored to a sub-directory of *XDG_CACHE_HOME*. Each store has a 5% chance of triggering a cleanup of the cache. When a cleanup occurs, each cached response has a 5% chance of being dropped, including responses that are indicated as valid. This way, a malicious cache response that has a maliciously long validity will not stay too long in the cache. A log line will indicate which items are dropped.

The *(webid-oidc cache)* module exports two functions to deal with the cache.

`clean-cache` [`#percents`] [function]
: Drop *percents*% of the cache right now.

`use-cache` *f* [function]
: Call *f* with no arguments, with the default HTTP request method set to a function that tries to use the cache first.o

  The cache will be read and written in the '`web-cache`' subdirectory of the cache home. To check the time window validity, the *current-date* parameter is used.

  The back-end function, *http-get*, defaults to that of *(web client)*.

`cache-home` [parameter]
: This parameters sets the cache directory. By default, it is *XDG_CACHE_HOME*.

10 The HTTP Link header

The HTTP Link header lets you attach metadata about a resource, directly in the HTTP protocol. It is used to link resources to their auxiliary resources, for instance.

The following API is defined in *(webid-oidc http-link)*:

`<link>` *`target-iri relation-type target-attributes`* [Class]
: The link refers to the *target-iri* that is being linked to the requested resource, with a given *relation-type* (a string), and optional additional *target-attributes*.

  When constructing a `<link>`, you should use the `#:target-iri`, `#:relation-type` and `#:target-attributes` keyword arguments (`#:target-attributes` defaults to the empty list) to initialize the link. For convenience, the `#:anchor`, `#:hreflang`, `#:media`, `#:title`, `#:title*` and `#:type` keyword arguments can be passed to add well-known target attributes.

`<target-attribute>` *`key value`* [Class]
: If you wish to add new extension target attributes, you can create an ad-hoc target attribute with *key* and *value* (initialized as `#:key` and `#:value` constructor keyword arguments).

`target-iri` *`link`* [Generic]
`relation-type` *`link`* [Generic]
`target-attributes` *`link`* [Generic]
: Getters for the `<link>` class.

`key` *`target-attribute`* [Generic]
`value` *`target-attribute`* [Generic]
: Getters for the `<target-attribute>` class.

`target-attribute` *`link key`* [Generic]
: Return the value of the first target attributet with *key*.

`anchor` *`link`* [Generic]
`hreflang` *`link`* [Generic]
`media` *`link`* [Generic]
`title` *`link`* [Generic]
`title*` *`link`* [Generic]
`type` *`link`* [Generic]
: Convenience attribute lookup functions. `anchor` returns an URI referencce, the others return a string.

`declare-link-header!` [function]
: Declare functions to parse, validate and print HTTP Link headers with the Guile web request / response API.

`request-links` *`request`* [function]
`response-links` *`response`* [function]
: Return the list of links in *request* or *response*.

11 Content negociation

There are a number of different available syntaxes for RDF, some being simple and human readable like *turtle*, and others more adapted to the JavaScript ecosystem like *json-ld*. To help clients both from and outside of the JS ecosystem, the server needs to perform *content negociation*, i.e. convert from one content-type to another.

`convert` *`client-accepts server-name path content-type content`* [function from `(webid-oidc serve)`]

Convert the resource representation under *path* on *server-name*, which has a given *content-type* and *content*, to a content-type that the *client accepts*.

Return 2 values:

1. the accepted content-type;
2. the content in the given content-type.

Currently, the only conversions are from and to *Turtle* and *N-Quads*.

12 Server endpoints

The disfluid server consists of a set of endpoints that handle requests. The *(webid-oidc server endpoint)* module defines the base building blocks.

`<endpoint>` () *host path* [Class]
: All endpoints define a *host* for which they are relevant, and an absolute *path*. If a request comes with a matching host and a matching path prefix, then it will be handled by this endpoint.

  If *host* is `#f`, then this endpoint will be used for all hosts.

  You can construct an endpoint with the `#:host` and `#:path` keyword arguments: the former is a string (defaults to `#f`), and the latter is a string starting with `"/"` (defaults to `"/"`).

`handle` *endpoint request request-body* [Generic]
: Handle *request* with *endpoint*.

  request is the request, in the form of an object from *(web request)*. *request-body* is `#f`, a string or a bytevector.

  Return 3 values: a response, a response body (a string, bytevector, input port, or `#f` if no body is expected), and some response meta-data.

`host` *endpoint* [Generic]
: Return the host name *endpoint* is configured to respond to.

`path` *endpoint* [Generic]
: Return the path prefix *endpoint* is configured to respond to.

`relevant?` *endpoint request* [Generic]
: Check if *endpoint* is configured to respond to *request*.

The handler may throw exceptions to signal errors. Exception messages will be printed to the log file, and user messages will be passed to the user.

12.1 Error signalling

The *(webid-oidc server endpoint)* module defines exception types that can be emitted to abort the computation in a handler. If an exception of a different kind is raised, this will lead to a 500 Internal Server Error response.

`&web-exception` *code reason-phrase* [Exception type]
: The request failed, with *code* and *reason-phrase*.

`make-web-exception` *code reason-phrase* [function]
: Create an exception with *code* and *reason-phrase*.

`web-exception?` *exn* [function]
: Check if *exn* was thrown because the request failed.

`web-exception-code` *exn* [function]
`web-exception-reason-phrase` *exn* [function]
Return the code and reason-phrase for when *exn* was thrown, if it was thrown because of a failing request.

`&caused-by-user` *webid* [Exception type]
If a web exception is raised, maybe it is caused by some user identified by *webid* (an URI, or `#f`.

`make-caused-by-user` *webid* [function]
Constructor for `&caused-by-user`.

`caused-by-user?` *exn* [function]
Check if *exn* was caused by the user.

`caused-by-user-webid` *exn* [function]
Return the webid of the user that caused *exn*.

`&user-message` *sxml* [Exception type]
An exception containing a message that is safe to show to the user, as an SXML fragment of XHTML. Typically, this would be a `<p/>`, or a `<div/>`.

You can set a user-message multiple times. The occurences will be concatenated in the response, in the order they appear in the composite exception.

`make-user-message` *sxml* [function]
Create a new user message containing the *sxml* fragment.

`user-message?` *exn* [function]
Check if there is at least one user message in *exn*.

`user-message-sxml` *exn* [function]
Return all user messages in *exn*, as a `<div/>` SXML fragment.

12.2 Router endpoint

The first non-trivial handler is for the router endpoint, defined in *(webid-oidc server endpoint)*.

`<router>` (`<`*endpoint*`>`) *routed* [Class]
The router has a list of endpoints, and chooses which one will handle an incoming request based on the request fields. The *routed* endpoints is a list of endpoints. You can set it at construction time with `#:routed`.

The router will check if the *routed* endpoints are relevant, in turn, or return a 404 Not Found response if no endpoint is relevant.

`routed` *router* [Generic]
Return the list of endpoints for *router*.

12.3 Request authentication

The *(webid-oidc server endpoint authentication)* defines an endpoint that authentifies the user and passes the annotated request to a backend endpoint.

`<authenticator>` (`<`*endpoint*`>`) `backend server-uri` [Class]
: The authenticator calls the *backend* endpoint once it has authentified the user. If the authentication is successful, the request is annotated with a `'user` entry in the alist table containing the URI of the user. Otherwise, it is passed as is.

  To check the validity of the DPoP proof, the endpoint must know the public name of the server that is running, *server-uri*.

  It can be constructed with the `#:backend` and `#:server-uri` keyword arguments, respectively an endpoint and an URI.

`backend` *`authenticator`* [Generic]
: Return the backend endpoint of *authenticator*.

`server-uri` *`authenticator`* [Generic]
: Return the public server URI of *authenticator*.

12.4 Hello world

The *(webid-oidc server endpoint hello)* module defines an endpoint that will greet the user, to check that Solid authentication worked. It is intended to be a backend for an authenticator.

`<greeter>` (`<`*endpoint*`>`) [Class]
: An endpoint that will greet anonymous users and authenticated users.

12.5 Reverse proxy

The *(webid-oidc server endpoint reverse-proxy)* module defines a *reverse proxy*, an endpoint that passes the incoming request to a backend server with added metadata.

`<reverse-proxy>` (`<`*endpoint*`>`) `backend-uri authentication-header` [Class]
: This endpoint will handle the incoming requests by adding a header, named *authentication-header* (a symbol), to hold the webid of the authentified user, and passing it to the server listening at *backend-uri* (an URI).

  You can construct it with `#:backend-uri` and `#:authentication-header`.

`backend-uri` *`reverse-proxy`* [Generic]
: Return the URI where requests are passed.

`authentication-header` *`reverse-proxy`* [Generic]
: Return the header set by the reverse proxy to hold the authenticated webid.

12.6 Client pages

The *(webid-oidc server endpoint client)* module defines an endpoint to serve the public pages for a client application.

`<client-id>` (`<`*endpoint*`>`) *`client-id redirect-uris client-name client-uri grant-types response-types`* [Class]

During the OIDC authorization process, the identity provider must check some things against the public URI of a client application. This endpoint will respond to this query.

You can construct it with `#:`*`redirect-uris`* (a list of URIs), `#:`*`client-id`* (an URI, or string encoding an URI), `#:`*`client-name`* (a string), `#:`*`grant-types`* (a list of symbols or strings), `#:`*`response-types`* (a list of symbols or strings).

`redirect-uris` *`client-id`* [Generic]

Return the list of approved redirection URIs.

`client-id` *`client-id`* [Generic]

Return the URI where the application can be queried by the identity provider.

`client-name` *`client-id`* [Generic]

Return the associated name. Please note that the companion implementation of the identity provider in this package will not display the name to the user, because it can be misleading.

`client-uri` *`client-id`* [Generic]

Return the URI where people can find information about the application. Also not hidden by the identity provider.

`<redirect-uri>` (`<`*endpoint*`>`) [Class]

This endpoint receives an authorization code, and display it to the user, asking to paste it in the application.

12.7 Identity provider

The *(webid-oidc server endpoint identity-provider)* module defines endpoints that are required for an identity provider.

`<oidc-discovery>` (`<`*endpoint*`>`) *`configuration`* [Class]

Serve the OIDC *configuration*.

You can construct it with `#:`*`configuration`*.

`configuration` *`endpoint`* [Generic]

Return the OIDC configuration served by *endpoint*.

`<authorization-endpoint>` (`<`*endpoint*`>`) *`subject encrypted-password key-file`* [Class]

The authorization endpoint prompts the user for a password, and then grants an authorization code. It is defined for one particular user, whose webid is *subject*, and who knows the password. The authorization endpoint signs authorization codes with the key under *key-file*. If this file does not exist, a new key will be generated.

The constructor expects keyword arguments `#:subject`, `#:encrypted-password` and `#:key-file`.

`subject` *authorization-endpoint* [Generic]
Return the webid of the user authorized by *authorization-endpoint*.

`encrypted-password` *authorization-endpoint* [Generic]
Return the encrypted password used to authentify the user at *authorization-endpoint*.

`key-file` *authorization-endpoint* [Generic]
Return the file name where the key to sign authorization codes in *authorization-endpoint* is stored.

`<token-endpoint>` (`<`*endpoint*`>`) *issuer key-file* [Class]
The token endpoint exchanges authorization codes or refresh tokens for new access tokens. The access token is signed with the key loaded from *key-file*, and the access token is bound to the *issuer* URI (host name).

You can construct a token endpoint with the `#:issuer` and `#:key-file` keyword arguments.

`issuer` *token-endpoint* [Generic]
Return the issuer (URI with no path) that this *token-endpoint* operates for.

`key-file` *token-endpoint* [Generic]
Return the file name where the key to sign access tokens in *token-endpoint* is stored.

`<jwks-endpoint>` (`<`*endpoint*`>`) *key-file* [Class]
The JWKS endpoint returns the list of valid public keys used by the identity provider. For now, only the public part of the key under *key-file* is served.

You can construct one with the `#:key-file` header argument.

`<identity-provider>` (`<`*router*`>`) *oidc-discovery authorization-endpoint token-endpoint jwks-endpoint default* [Class]
An identity provider is the sum of an *OIDC discovery* endpoint, an *authorization-endpoint*, an *token-endpoint* and an *jwks-endpoint*, and a *default* endpoint that gets all the requests that aren't handled by the identity provider.

You can construct one with the following keyword arguments: `#:authorization-endpoint`, `#:token-endpoint`, `#:jwks-endpoint` and `#:default`.

`oidc-discovery` *identity-provider* [Generic]
Return the OIDC discovery endpoint of the *identity-provider*.

`authorization-endpoint` *identity-provider* [Generic]
Return the authorization endpoint of the *identity-provider*.

`token-endpoint` *identity-provider* [Generic]
Return the token endpoint of the *identity-provider*.

`jwks-endpoint` *identity-provider* [Generic]
Return the JWKS endpoint of the *identity-provider*.

`default` *`identity-provider`* [Generic]
: Return the endpoint where all requests that aren't handled by any element of the *identity-provider* go.

12.8 Resource server

The resource server is a read-write server with fine-grained authorizations. You can create one in the *(webid-oidc server endpoint resource-server)* module.

`<resource-server>` (`<`*endpoint*`>`) *`server-uri owner data-home`* [Class]
: Create a resource server endpoint. To manage RDF data, and in particular to identify owned resources, it is necessary that the server knows its public *server-uri*. *owner* is the webid of someone that has total control.

  If you want to manage multiple resource servers, you must make sure that each one of them has a separate *data-home* directory.

  You can construct one with `#:`*`server-uri`* (an URI), `#:`*`owner`* (an URI) and `#:`*`data-home`* (a directory file name or a thunk returning a file name; it may exist or not, defaults to `$XDG_CACHE_HOME`).

`server-uri` *`resource-server`* [Generic]
: Return the public URI of the *resource-server*.

`owner` *`resource-server`* [Generic]
: Return the webid of a user that has full control over *resource-server*.

`data-home` *`resource-server`* [Generic]
: Return the directory where *resource-server* stores persistent data.

13 Resources stored on the server

To store and serve resources, the server has two distinct mechanisms. A *content* is a read-only possible value for a resource, indexed by etags, and a *path* is a mutable value that indicates the etag of the resource, and of the auxiliary resources (description and ACL). With this separation, it is possible to atomically delete a resource and all associated auxiliary resources, by unlinking the corresponding *path*. It is also possible to mutate separately the ACL and the resource itself without writing a copy for both.

The *content* API is contained in the `(webid-oidc server resource content)` module.

`<content>` () *`etag content-type contained static-content`* [Class]
: This class encapsulate a static resource content linked to a particular *etag*.

  The *content-type* is a symbol, and *static-content* is a bytevector, although a string will be encoded to UTF-8 at construction time. *contained* is either `#f`, if the resource is not a container, or a list of resource paths (each one is a string) identifying contained resources.

  You can construct a content in two ways.

  If you pass `#:`*`etag`*, it will be loaded from the file system under the *etag* index, or if `#:`*`cache`* is passed or the `current-content-cache` is set to *cache*, it will try to load from *cache* first. If you define a cache, the result will also be added to *cache*.

  If you pass `#:`*`content-type`*, `#:`*`contained`* and `#:`*`static-content`*, but not `#:etag`, it will be created and saved to disk, and optionally added to the `#:`*`cache`* or the current content cache.

`<content-cache>` () *`cache`* [Class]
: Since the contents are read-only, it is possible to cache the values in memory to avoid reading the same file more than once. This is how the session works.

  cache is a hash table for string etag values to cached content values. It is initialized as an empty hash table.

`etag` *`content`* [Generic]
: Return the ETag of *content*, as a string.

`content-type` *`content`* [Generic]
: Return the Content-Type of *content*, as a symbol.

`contained` *`content`* [Generic]
: Return the contained paths of *content*, as a list of strings, or `#f` if it is not a container.

`static-content` *`content`* [Generic]
: Return the static content of *content*, as a bytevector.

`delete-content` *`content`* [Generic]
`delete-content` *`etag`* [Generic]
: Remove [*content*'s] *etag* from the file system. If it is cached in *session*, also remove the cached value. Otherwise, other sessions can still access it.

`current-content-cache` [parameter]
: A guile parameter indicating a cache where loaded contents should be added and preferably fetched. By default, no caching is performed. You need to set this parameter to benefit from it.

The *path* API is defined in `(webid-oidc server resource path)`.

`read-path` *path* [function]
: Read the resource at *path*, and return 2 values:

  1. the content of the main resource;
  2. an alist where keys are auxiliary resource type URIs (the type is from `(web uri)`), and the values are contents of the corresponding resource.

  If the resource is not found, raise an exception with type `&path-not-found`, and maybe `&uri-slash-semantics-error` if a resource with a different ending-in-slash exists.

  If the `current-content-cache` parameter is set to a cache, it will be used to load the content and auxiliary contents.

  This function is safe to call when the path is being modified, either by another thread, process or else, as the returned values will always be consistent. However, once the function returns, an updating process may have deleted the returned content. If this is the case, then you must call this function again to read the updated path.

`update-path` *path f* [`#:`*create-intermediate-containers?*`=#f`] [function]
: Read *path*, call *f* with two values: the main content and the auxiliary contents (as returned by *read-path*), and update the path accordingly. If *path* does not exist, then the first argument is `#f` and the second one is the empty list.

  If *f* returns `#f`, then the resource is deleted.

  If *f* returns two values: a content as the first and an alist of auxiliary types (as URIs) to auxiliary contents as the second, then the resource is updated.

  This function uses the `current-content-cache` parameter to load contents. If a resource is created or deleted, the parent's containment triples will be modified, so they will also be loaded in the cache.

  Some operations should create the intermediate containers for a given path, this is the case for the `PUT` HTTP verb. For `POST`, the parent should exist. The *#:create-intermediate-containers?* switch lets you change the behavior. In any case, it is an error to delete a non-empty container.

  The update is atomic, meaning that at any point in time the file is fully written out. Concurrent access to the same resource is performed by locking the lock file named X/.lock, where X is the first character of the base64-url sha-256 hash of the path. **The lock file is not meant to be removed** when the resource is unlocked. It should be locked with `flock` instead. **Like other forms of lock-based synchronization, this function is not composable**. This means that you cannot call this function within *f*, otherwise a deadlock may ensue.

  If the resource is created or deleted, then the parent resource is updated as well. To avoid deadlocks with other processes, please follow the following rules: lock the path,

then lock the parent path, then update the parent, then unlock the parent, and finally unlock the child path.

The Web Access Control specification defines an RDF vocabulary to check whether a given user is allowed to perform some operations. The `(webid-oidc server resource wac)` helps you do that.

`wac-get-modes` *`server-name path user`* [`#:http-get`] [function]
: Return the list of modes that are allowed for *user* accessing *path*. The *server-name* URI is required to find the relevant triples in the ACL. If *user* is unauthenticated, pass `#f`.

  Please note that in any case, the data owner should have all rights whatsoever, bypassing WAC. Otherwise, it is possible to steal control away from the data owner.

`check-acl-can-read` *`server-name path owner user`* [`#:http-get`] [function]
`check-acl-can-write` *`server-name path owner user`* [`#:http-get`] [function]
`check-acl-can-append` *`server-name path owner user`* [`#:http-get`] [function]
`check-acl-can-control` *`server-name path owner user`* [function]
[`#:http-get`]
: Assert that the resource at *path* on *server-name* is owned by *owner*, and check that *user* has the proper authorization. Otherwise, raise an exception of type `&forbidden`.

14 Running a client

The job of the client is to use accounts to fetch private resources on the web. The *(webid-oidc client)* defines the `<client>` class.

`<client>` *client-id key-pair redirect-uri* [Class]

In OIDC, a client is an application that does not hold the resources. It may in fact be a network server available on the web, or a program that you run on your machine. Being a network server or not is irrelevant.

The `<client>` class is designed with immutability in mind. You can create a client with the `make` generic method, using these keywords to initialize values:

`#:client-id`
: to set the public client identifier (this endpoint should be available on the world-wide web), as a string representing an URI or an URI from `(web uri)`;

`#:key-pair`
: to use a specific key pair. If not set, a new key pair will be generated;

`#:redirect-uri`
: to set the redirect URI that the application controls. It may just be a page showing the authorization code, with instructions on how to paste this code into the application. It should match one of the authorized redirect URIs in the client identifier endpoint.

  If you want to set a state parameter for the redirection, you can do it by setting the guile parameter `authorization-state`.

`uri client-id` (*client* `<client>`) [Generic method]
`key pair key-pair` (*client* `<client>`) [Generic method]
`uri redirect-uri` (*client* `<client>`) [Generic method]

Slot accessors for *client*.

`client` [Parameter]

Define this parameter to set the client to use to access private data.

To access private data, you must identify yourself. The *(webid-oidc client accounts)* module lets you define accounts.

`<account>` *subject issuer id-token access-token refresh-token key-pair* [Class]

Encapsulate an account. *subject* is your webid, while *issuer* is a host name. *id-token* is the *decoded* OIDC ID token, i.e. a pair of `(header . payload)`, because we don't need to show it to any other party, so its authenticity needs not be demonstrated. However, *access-token* is an *encoded* access token (into a string), because we don't need to worry about its internals on client side.

There are different ways to initialize an account. First, you can save all parameters to some form of storage, and restore it by using the associated keyword arguments at construction time:

`#:subject`
`#:issuer`

`#:id-token`
`#:access-token`
`#:refresh-token`
`#:key-pair`

If you want to make a new account, you would ask the user for an identity provider, and pass it with `#:issuer` as the only initialized value. The constructor will log you in, using the `authorization-process` and `anonymous-http-request` function parameters.

If you want to refresh an access token, you would also set `#:refresh-token`.

In any case, when you don't specify a value, it's as if you passed `#f`.

`authorization-process` [Parameter]
This function is called when an explicit user authorization is required, for instance because there is no refresh token and the access token expired. The function takes an URI as argument, with an additional `#:reason` keyword argument containing the reason for the authorization as a string. In this function, you should present the reason to the user and ask the user to browse this URI so that your application gets the authorization code.

`anonymous-http-request` [Parameter]
This function is used as a back-end for private resource access, and to query the server configuration. It defaults to `http-request` from *(web client)*.

`uri subject (account <account>)` [Generic method]
`<account> set-subject (account <account>) (uri string or URI)` [Generic method]
`uri issuer (account <account>)` [Generic method]
`<account> set-issuer (account <account>) (uri string or URI)` [Generic method]
`optional decoded ID token id-token (account <account>)` [Generic method]
`<account> set-id-token (account <account>) (id-token optional ID token)` [Generic method]
`optional encoded access token access-token (account <account>)` [Generic method]
`<account> set-access-token (account <account>) (access-token optional access token)` [Generic method]
`optional <string> refresh-token (account <account>)` [Generic method]
`<account> set-refresh-token (account <account>) (refresh-token optional <string>)` [Generic method]
`key pair key-pair (account <account>)` [Generic method]
`<account> set-key-pair (account <account>) (key-pair optional key pair)` [Generic method]
Slot accessors and functional setters for *account*.

If you intend to run a public network server as a client application, you may have multiple different users, but you should not let any user use any account. If this is the case, you can either store the accounts on the user agent storage (for instance, as a cookie), or store all

of them on the server. If you choose to store the accounts on the user agent, at least use a new key pair for each of them. If you want to store the user database on the server side, be aware that no entity other than yourself will check that your user abides by any term of service, so it is possible that a single user makes a lot of accounts to annoy you and fill your hard drive with key pairs. If your application does not let random people to use it, you might want to use *protected accounts*, to help you check that the users cannot impersonate each other.

`<protected-account>` (`<account>`) *`username encrypted-password`* [Class]
This superclass of `<account>` is protected by a username and password. It is constructed with the initializer keywords `#:username` and `#:encrypted-password`.

`<string> username` (*`protected-account`* `<protected-account>`) [Generic method]
`<protected-account> set-username` (*`protected-account`* `<protected-account>`) (*`username`* `<`*string*`>`) [Generic method]
`<string> encrypted-password` (*`protected-account`* `<protected-account>`) [Generic method]
`<protected-account> set-encrypted-password` (*`protected-account`* `<protected-account>`) (*`encrypted-password`* `<`*string*`>`) [Generic method]
Slot accessors and functional setters for *protected-account*.

`<account> invalidate-access-token` (*`account`* `<account>`) [Generic method]
Indicate that the access token in *account* cannot be used. Before using *account* again, you will need to refresh the access token. This function does not mutate *account*.

`<account> invalidate-refresh-token` (*`account`* `<account>`) [Generic method]
Indicate that the refresh token has been revoked for *account*. This is usually an indication that the user don't want your application to access her private data. This function does not mutate *account*.

`<account> refresh` (*`account`* `<account>`) [Generic method]
Refresh the access token.

`&authorization-code-required` *`uri`* [Exception type]
If the login process requires the user to send an authorization code, an exception of this type will be raised, with an implicit invitation for the user to browse *uri* and follow the instructions.

The instructions will be handled by the *redirect-uri* in the `login` function. If your client is a traditional web application, the user will be redirected to this URI with an authorization code. If your client is a native application, then maybe that redirection URI should display the authorization code and invite the user to paste it in the appropriate place in the application.

When an exception of this type is raised during the `login` function, it is continuable, meaning that the login function will resume. You need to create an exception handler for an exception of this type, look up the *uri*, direct the user to browse it, get the authorization code back, and *return* the authorization code *from the exception handler*.

`make-authorization-code-required` *uri* [function]
`authorization-code-required?` *error* [function]
`authorization-code-required-uri` *error* [function]
Constructor, predicate, and accessor for the `&authorization-code-required` exception type.

`&refresh-token-expired` [Exception type]
The refresh token can be used to still perform requests on behalf of the user when perse is offline. However, if the refresh token expires while the user is offline, it is not possible to log in again, because it requires a new authorization code. So, it is not possible to recover from this error, and the refresh token is immediately discarded.

`make-refresh-token-expired` [function]
`refresh-token-expired?` *error* [function]
Constructor and predicate for the `&refresh-token-expired` exception type.

`invalidate-access-token` *account* [function]
Discard the access token for *account*. It is not saved in the user database yet. This is roughly equivalent to log out.

`invalidate-refresh-token` *account* [function]
Discard the refresh token for *account*. You still need to save the *account*.

`&token-request-failed` *response response-body* [Exception type]
If the token endpoint is unable to deliver an identity token and an access token, this exception is raised with the identity provider *response* and *response body*. This exception cannot be continued.

`make-token-request-failed` *response response-body* [function]
`token-request-failed?` *error* [function]
`token-request-response` *error* [function]
`token-request-response-body` *error* [function]
Constructor, predicate, and accessors for the `&token-request-failed` exception type.

The *(webid-oidc client)* module provides the most useful function for a client.

`request` *account uri . args* [function]
Perform a request on behalf of *account*, with the current value of the *client* parameter as the client, using as a backend the current value of *anonymous-http-request*.

Finally, to implement your application, there needs to be a public endpoint for the resource server to check that you are not impersonating another application. This endpoint can be served by any web server, but a convenience procedure is made available here:

`serve-application` *id redirect-uri* [*#client-name*] [*#client-uri*] [function]
Return a handler for web requests to serve the application manifest and the redirection to transmit the authorization code. You should set the *client-name* to your application name and *client-uri* to point to where to a presentation of your application.

15 Serialization to (S)XML

The *(webid-oidc serializable)* module provides tools to have serialization to SXML and deserialization from XML.

`<plugin-class>` (`<`*class*`>`) *module-name direct-name* [Class]

This metaclass permits to register plugins. *module-name* is the name of a module that defines the class, and *direct-name* is the class name without the surrounding angle brackets. Please note that all plugin classes should be surrounded by angle brackets.

Most GOOPS classes defined in this program are actually plugin classes.

Serialization works for each slot by serializing other plugin classes the normal way, and other values are simply represented as strings with `display`.

Deserialization works by loading the module containing the target class, collecting a value for each slot (a string for non-plugin-class-valued slots), and making an instance of that class with all collected values. The initialization function should accept strings values, for objects that are not of a plugin class.

Since most scheme data types written by `display` cannot be read in a meaningful way, you may add a `#:->sxml` slot option with a function taking the slot value and either returning a string that the initialization function can parse, or an SXML fragment. For instance, if a slot should contain an URI value, you would pass `#:->sxml uri->string` as options to the slot definition, and accept a string value in the initialization function, that you would convert to an URI with `string->uri`.

Sometimes slots contain functional data that cannot be serialized. In this case, pass `#:->sxml 'ignore` to avoid serialization.

`read/xml` *port* [function]

Read the XML document at *port* and deserialize it.

`->sxml` *object* [function]

Convert *object* to an SXML fragment.

16 Exceptional conditions

The library will raise an exception whenever something fishy occurs. For instance, if a signature is invalid, or the expiration date has passed.

When the client is responsible for an error, such as presenting an invalid access token, a compound exception is raised. It is sometimes useful for the user to understand what happened, because it could indicate a problem in a part of the web they need to change. For instance, if the access token cannot be decoded because the identity provider is down, then maybe informing the user of that fact is useful.

However, presenting too much information is a security risk. For instance, if the system administrator also runs a private server on the same machine, and a malicious client tries to pretend that this private server is an identity provider, then the public server will try to query the private server. If an error happens and the public server displays some information to the client, then a part of the information comes from the private server. Thus, a balance needs to be found so that not too much is revealed.

The module *(webid-oidc errors)* defines an exception type that indicates a message that is safe to display to the user.

`&message-for-the-user` *`message`* [Exception type]

Indicate that *message* can be safely displayed to the user. It is an XHTML paragraph (or equivalent), presented as SXML.

`make-message-for-the-user` *`message`* [function]
`exception` [user-message]

Constructor and accessor for the `&message-for-the-user` exception type.

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/licenses/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

&

—

<

A

B

C

D

E

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W