

FastFed

Simplifying the adoption of Single-Sign-On

Introduction

SAML and OpenID Connect (OIDC) are two popular standards for Federated Single Sign-On. While both are widely deployed, there remains significant friction in setting up new relationships between identity and service providers. As a result of the friction, many service providers who support federation are seeing low adoption of the feature. The vast majority of customers create yet-another username/password.

FastFed is a proposed standard which seeks to reduce the onboarding friction for common uses of federation. An ecosystem of “FastFed Compliant” identity and service providers will enable end-users to instantiate new federation relationships with a few clicks, and without needing to understand the underlying technologies. Importantly, FastFed will not change the pre-existing standards, nor will it break existing implementations. Instead, it defines additional metadata, APIs, and flows to make existing standards easier to adopt.

This document describes the current barriers to adoption for federation, how FastFed seeks to eliminate them, and enumerates the major open questions to be addressed for FastFed to become a reality.

Barriers to Adoption of Federation

The barriers are best illustrated by describing the typical experience of an enterprise administrator who is configuring single sign-on for a new SaaS application.

SAML and OpenID Connect are examined in sequence.

SAML Experience

SAML is a widely-adopted authentication protocol, especially for SSO to SaaS applications. A typical ecosystem that leverages SAML might include an on-premises directory, such as Microsoft AD or LDAP. Alongside this, an “identity provider” service is run which performs the SAML communication exchanges. Two common implementations are Active Directory Federation Server (ADFS) and Shibboleth. Many other providers exist in the marketplace.

To configure single sign-on to a new SaaS application, the administrator typically takes the following actions:

- First, they sign-up for the SaaS app by visiting the provider’s website, creating an account, and configuring any necessary settings (such as billing, or registering namespaces).
- Next, the admin searches the service provider documentation for how to set up single-sign-in. This typically leads them to a web form that asks for values. There is little consistency in how these values are specified or labeled. Some applications want a metadata file. Others ask for individual fields to be input one-by-one, such as “Entity

ID”, “Sign-In URLs”, and certificates. The administrators must search their own identity provider’s documentation to find these values and manually copy and paste them over. Someone unfamiliar with SAML will likely become confused and frustrated by the unknown terminology.

- At this point, the administrator likely has 4 browser windows open: 2 showing documentation from each party, and 2 for the various forms into which they are copying-and-pasting values.
- Next, they must do the same in the reverse direction, copying information about the service provider that is needed by their identity provider. While the SAML spec defines a standard metadata format for this exchange, few services provide it. Instead, admins manually copy things like “ACS URL”, “Entity ID”, “Relay State”, and “Name ID Format”. Instructions will give generic pointers such as “ACS URL should be the value https://<your_domain.example-saas-app.com/tenant_id=?<your_tenant_id>. Please look for this value in your service provider.”
- Some attributes aren’t provided by the service provider, such as name, description, and logo. The admin must provide their own definitions and/or find an image to use. They may have to manually resize and format the image to meet the requirements of their identity provider.
- After this is done, the administrator must decide how to format the information about their users in order to send it to the service. For example, “FirstName” and “LastName” are commonly exchanged attributes, but the service provider might require the former to be labeled as “FName” “GivenName”, “first-name”, or something else. The administrator is responsible for defining any translations in order to publish their employee information using the naming formats expected by the service provider.
- Finally, some applications require that users be “pre-provisioned” (i.e. preregistered) with the service before they can log in. The administrator must determine how to execute this provisioning. This might require the exchange of more user attributes and more custom mappings. Sometimes, they need to implement systems to do this provisioning with all the burden that entails, such as writing software scripts, managing software credentials, and system monitoring.

Throughout this process, mistakes inevitably happen; steps are missed, typos occur. Often, the documentation is unclear or misleading. As a result, the administrator experiences a frustrating sequence of unexpected failures and confusing error messages. Debugging can require the user to become an expert in Identity technologies and/or contact technical support. As a result, a typical enterprise will budget 2 weeks for a new SAML integration.

Adding to the pain, the administrator will often discover that their configuration stops working later when time-bound credentials and certificates expire. The parties have no long-lived communication channel in order to automatically rotate credentials or publish updates to each other. Instead, the administrator must manually rotate the credentials for each service, one-by-one.

OpenID Connect (OIDC) Examples

OIDC is a newer authentication protocol, commonly seen by consumers via the “Login with Facebook/Google/Amazon/Other” buttons that appear on websites and apps. These companies

This document is for discussion purposes only, and not an Implementers Draft or Final Specification

vend toolkits for app developers to embed their SSO solutions. The result is a simple experience for end-users to reuse their existing accounts on a wide range of applications.

OIDC has less adoption in the business-to-business and SaaS marketplace. (This is mainly due to being a newcomer in a domain in which SAML was a well-entrenched solution.) As a result of limited adoption, there are fewer examples of real OIDC friction to point to.

Nonetheless, if an administrator wishes to use OIDC for SaaS applications, they would likely encounter many of the same issues currently experienced with SAML. For example:

- Relying parties must register with a provider to receive a clientID and clientSecret. While the specs do define a programmatic registration procedure, few providers implement it, and hence the registration is likely to require a human visit a UI and manually submit a form. A human will be copying-and-pasting values between UIs, much like the SAML experience.
- OIDC has a limited set of predefined user attributes, which it calls claims. These claims include values that are common in the social networking ecosystem, such as Name and Profile URL. But, it lacks extended user information needed for enterprises, governments, or educational institutions. As a result, each application owner may define their own extended attribute names. Much like the SAML experience, integration with these applications would require a custom mapping of attribute names between what the provider vends, and what the service expects.

How FastFed Seeks to Minimize the Barriers

A rational eye could look at the barriers to adoption and see that a significant contributor is that existing standards are not leveraged. For example, SAML defines a metadata file format, but not all parties use them. OIDC defines a dynamic registration flow, but few providers implement it. And, while there are plenty of existing schemas for modeling user attributes, many service providers choose to ignore them and define their own attribute names.

At the cynical extreme, one could perceive FastFed as taking the perspective that “Nobody is fully leveraging the existing standards, so let’s create a new standard that instructs people to use the existing standards.”

To address that concern, it is worthwhile to enumerate what FastFed brings to the table and how it addresses the problems.

- First, the specification will define the term “FastFed Compliant”. While these are only words, having a common vocabulary can be a powerful driver of change. It is a shorthand reference to a desirable user experience. Technology providers who enable this experience can promote themselves as compliant. Others who neglect this experience can be called out for incompliance.
- Next, it will define the portions of existing specifications which must be implemented to be “FastFed Compliant”. For example, the SAML spec is large and much of the

functionality is not necessary for Single-Sign-On use cases. FastFed will help implementers zoom into the portions that are relevant to them.

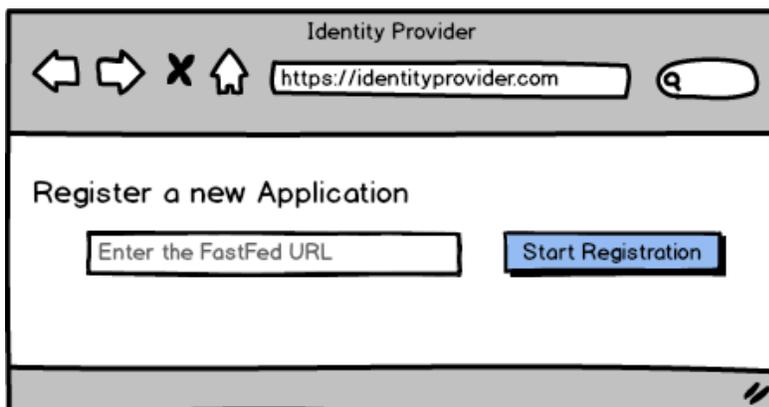
- In places where existing specifications are insufficient, FastFed will fill in the blanks. It will do this with a minimal amount of additional metadata. Some of the extensions include:
 - Specifying which protocols are supported (e.g. OIDC, SAML, or both)
 - Declaring which user attributes are supported, and how to map those attributes into the SAML/OIDC protocols.
- Finally, FastFed will define the user experience flows and any new APIs necessary to achieve them. Some of the issues addressed by these flows include:
 - OIDC Dynamic Registration can require an Initial Access Token. The OIDC spec does not define how to generate this token and share it with the RP. FastFed defines this exchange.
 - In multi-tenant systems, the configuration files (such as SAML Metadata and OIDC Discovery Docs) may be custom-generated for each tenant, rather than being a static file in a well-known location. FastFed will specify how parties find and share these configuration files.

User Experience

Because FastFed is predicated on providing a better user experience, it is helpful to clarify this experience.

The following example illustrates a potential FastFed flow for an enterprise administrator who is enabling SSO to a 3rd party SaaS application.

An example implementation of the FastFed flow begins with the administrator visiting a UI provided by their identity provider. With FastFed, only one piece of information is needed from the administrator: the FastFed URL for the new application.



The image shows a browser window titled "Identity Provider". The address bar contains "https://identityprovider.com". The main content area has the heading "Register a new Application". Below the heading, there is a text input field labeled "Enter the FastFed URL" and a blue button labeled "Start Registration".

Next, the admin is redirected to the service provider.

This document is for discussion purposes only, and not an Implementers Draft or Final Specification



The service provider may ask a few questions about sign-on preferences (not shown). This is at the discretion of the service provider. When the service provider is finished, the administrator is redirected back home.

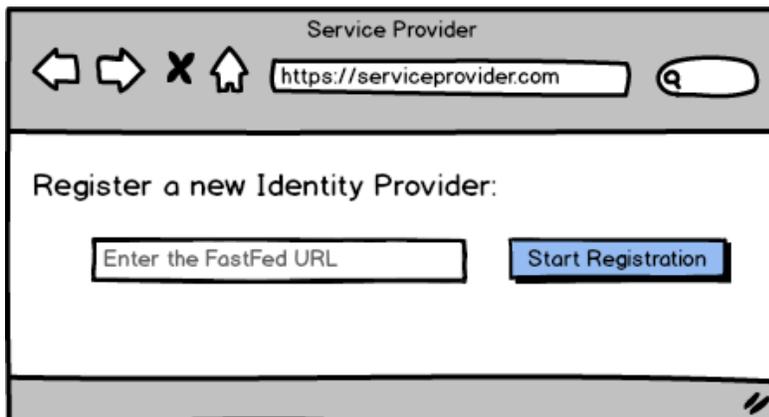


After completing the registration, the configuration is complete. Users can sign-in.

Most identity providers are likely to include an additional screen to ask who should be permitted to sign-in to the application. This is outside the scope of the FastFed spec.



As an additional exemplary implementation of the FastFed flow, the registration experience can also be initiated from the Service Provider. This is simply another entry point into the same flow described above.



Key Concepts

To deliver the customer experience, FastFed has a few needs. Understanding these needs can illuminate why certain activities occur within the specification.

Common Language for User Attributes

Today, there is no shared agreement on how to represent user attributes across various SSO implementations. The lack of consistency results in each integration being a “one-off” where someone must define a mapping between what the identity provider vends and what the service expects.

To remove this friction, FastFed requires a *lingua franca* for user attributes. SCIM is the chosen language.

Throughout the specification, all parties communicate their needs for user attributes in the form of SCIM schema references. Any transformations to/from alternative formats are specified as transformations to/from SCIM.

Credential Exchange

One of the key needs of FastFed is that the identity provider and service provider be able to communicate with one another, programmatically. Communication occurs for several purposes. It allows the exchange of registration materials, the ongoing rotation of keys, and the provisioning of users. Because many of these communications involve private information, there must be a mechanism to demonstrate that the end-user has approved access to the information.

FastFed uses OAuth 2.0 access_tokens and refresh_tokens for this purpose. The specification describes how these tokens are distributed to each party.

FastFed Configuration Files

Both parties need to understand the capabilities of the other in order to establish a working relationship, such as whether to use SAML or OIDC.

The establishment of this relationship occurs in a process named the FastFed handshake. The handshake relies on the exchange of configuration information.

To enable this communication, FastFed defines a standard configuration file format. However, rather than a single file, FastFed divides the configuration into two parts: a public file and a private file. The split arises because some multi-tenant services will generate different SSO configurations for each tenant. In addition, multi-tenant services often have privacy requirements that preclude exposing tenant information without approval.

To meet these goals, the private file (“FastFed Metadata”) allows the vending of unique, access-controlled configuration for each tenant. The public file (“FastFed Discovery”) allows the handshake to bootstrap so that each party can acquire permissions to the private files.

Security

TBD

Note – a threat model is still todo. It is expected that the specification will evolve to address the threats. Mitigations may be worth explaining here if they are non-intuitive.

Detailed Flows

With the foundations in place, we are ready for details. This section steps through the FastFed registration process and describes the detailed activities at each step.

Note on terminology:

This document is for discussion purposes only, and not an Implementers Draft or Final Specification

FastFed spans multiple standards. A single actor could be referred to as the “SAML Service Provider”, the ”OpenID Connect Relying Party”, or the “SCIM Server”. To minimize verbosity, this document uses the SAML vocabulary:

- Identity Provider (IdP)- the authoritative user directory
- Service Provider (SP) – the application provider

When applied to non-SAML specifications, the appropriate translations apply. **(TODO – OK for a strawman document, but need to nail down the real vocabulary before drafting the specification.)**

(Step 1) Publication of Discovery Configuration

FastFed begins with the publication of “Discovery” configuration files. Both parties host their configuration at a publicly accessible URL. The configuration contains a minimal amount of information necessary to bootstrap the registration handshake.

Example:

```
GET /.well-known/fastfed-discovery HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "identity_provider":
    {"handshake_endpoint": "https://idp.example.com/fastfed/handshake/start"},
  "service_provider":
    {"handshake_endpoint": "https://sp.example.com/fastfed/handshake/receive",
     "auth_protocols_supported": ["OIDC", "SAML"]}
}
```

The Discovery configuration can contain a block for “identity_provider”, “service_provider”, or both, depending on what role(s) the entity plays. A system which acts solely as an IdP, for example, would only include the “identity_provider” block. A system which supports both capabilities would include both blocks.

The configuration includes a “handshake_endpoint”, which is the location where the user_agent should be directed in order to perform the FastFed registration handshake.

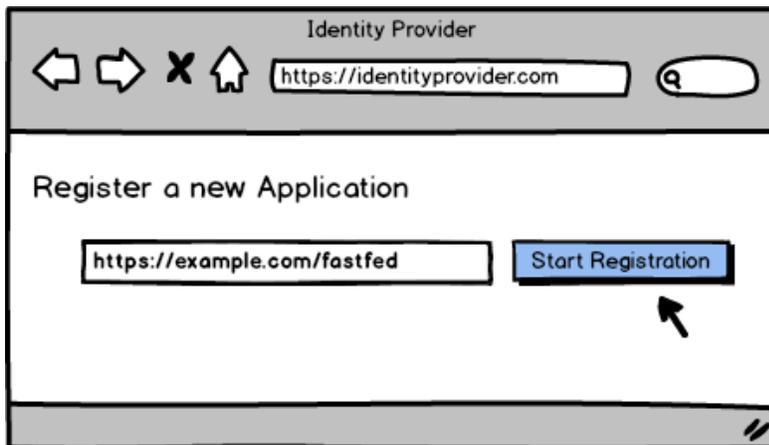
In addition, service providers declare the supported authentication protocols. The IdP will examine the list and decide which method(s) to use. The decision is purely at the discretion of the IdP and can be based on the customer preferences, the capabilities of the IdP, or any other factor. The IdPs chosen protocol will be expressed in the private files (later).

(Open Question – Should the Discovery configuration also include the application name, logo URL, and other displayable information?)

(Step 2) IdP Allows Administrator to Start Process

The administrator chooses an application and initiates the registration process.

This could be done by a UI and the contents of the UI are at the discretion of the identity provider and outside the scope of the FastFed specification. In the most basic implementation, an end-user manually enters a URL, such as “https://example.com/.well-known/fastfed” or simply “https://example.com/” when using the “.well-known” location.



The IdP is responsible for authenticating the end-user and ensuring they are authorized to register new applications. This is outside the scope of the specification.

(Step 3) IdP Creates a Confirmation

The next step is to confirm the user wishes to proceed with the registration. The contents are at the discretion of the identity provider and outside the scope of the FastFed specification. Since the FastFed handshake may result in private information being exposed to the service provider, this page may include notifications, or terms and conditions, that are applicable to the situation. Alternatively, a provider may choose to display nothing and immediately redirect the user to the next step.

The confirmation step is initiated by making an HTTP POST to the IdPs handshake endpoint and passing the SP’s FastFed Discovery URL. **(Open Question – Need to nail down the semantics of whether this is a GET or POST...)**

Example:

```
POST /fastfed/idp/handshake/start
Content-Type: application/x-www-form-urlencoded
Content-Length: 50
Host: idp.example.com

sp= https%3A%2F%2Fsp.example.com%2F.well-known%2Ffastfed
```



(Step 4) IdP generates tenant-specific information

Upon receiving confirmation from the user to initiate the registration, the identity provider performs a number of actions behind the scenes. These actions include the following:

- Download the Service Provider's FastFed Discovery file and extract the following values:
 - Supported Authentication Protocols (e.g. SAML, OIDC)
 - Handshake URL
- Validate the Service Provider. At minimum, this requires validating that the IdP is compatible with the authentication protocols offered by the SP. E.g. if one party supports SAML, and the other only supports OIDC, the registration cannot proceed. The IdP may perform additional validation beyond this, such as ensuring the application is part of a whitelisted collection of trusted apps. This is outside the scope of FastFed.
- Examine the Supported Authentication Protocols to determine which to use. If the SP supports multiple protocols, choose one.
- Create an `initial_access_token` and nonce that will be given to the SP. These values will allow the SP to call an OAuth endpoint and attain an `access_tokens` and `refresh_tokens`, so they can interact with the IdP programmatically.
- Generate a ReturnTo URL. This tells the service provider how to redirect the user back to the identity provider to finish the registration handshake.
- (Optional) Generate a `state` attribute which encodes any state that the identity provider wishes to preserve across the handshake. This will be passed to the service provider and echoed back to the ReturnTo URL.
- Generate (or otherwise make available) a private FastFed Metadata file to share with the SP. Access to the file can be restricted to holders of the `initial_access_token`. The file contains the following information:
 - Name (*Required*)
 - A displayable name for the Identity Provider (**Open Question – limits on name length?**)
 - Logo URI (*Optional*)
 - A logo that represents the Identity Provider (**Open Question – limits on size & image format?**)
 - Auth Protocols (*Required, at least one*)

This document is for discussion purposes only, and not an Implementers Draft or Final Specification

- Describes whether the IdP will use SAML, OIDC, or both.
- SAML Metadata URI (*Required if the AuthProtocols include SAML*)
 - A URL for a standard SAML Metadata document containing an IDPSSODescriptor. Access to this file may require the access_token.
- OIDC Configuraiton URI (*Required if the AuthProtocols include OIDC*)
 - A URL for a standard OpenID Provider configuration document. Access to this file may require the access_token.
- OAuth Token Endpoint (*Required*)
 - A URL for a standard OAuth token exchange using the refresh_token.
- SCIM Endpoint (*Required*)
 - A URL for standard SCIM operations exposed by identity providers, such as fetching a User by Id. This endpoint may not be usable by the service provider until after the handshake completes.
- Supported Attributes (*Required*)
 - Describes the collection of SCIM user attributes that the IdP can make available (presuming the administrator approves their release to the service provider). These are expressed as a collection of SCIM attribute names. (**Open Question – any prior implementations here? Closest example was OIDC claims_supported, but SCIM needs a richer expressiveness.**)

At the end of this process, the IdP is capable of giving access to a configuration file, for example, like this:

```
GET /fastfed/metadata HTTP/1.1
Host: tenant12345.example.com
Authorization: Bearer kdDS42K12ojk
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "identity_provider": {
    "name": "Awesome IdP"
    "logo_uri": "https://example.com/images/idp_logo.png",
    "auth_protocols": ["SAML","OIDC"], #In practice, only 1 protocol typically
chosen.
    "saml_metadata_uri": "https://tenant12345.example.com/saml-metadata.xml",
    "oidc_configuration_uri": "https://tenant12345.example.com/oidc-configuration",
    "token_endpoint": "https://tenant12345.example.com/token",
    "scim_endpoint": "https://tenant12345.example.com/scim",
    "supported_attributes": {
      "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"],
      "id",
      "userName",
      "name": {
        "familyName",
        "givenName",
      },
      "displayName",
      "emails",
      "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
        "employeeNumber",
        "costCenter",
```

```
    "manager": {  
      "value"  
    }  
  }  
}
```

(Step 5) IdP redirects the user to the SP

At this point, the IdP has initialized the registration and is ready to redirect the user to the SP to execute a similar sequence of steps. This redirection is accomplished by issuing an HTTP 302 to the location that the SP has specified in the FastFed Discovery File.

Example:

```
HTTP/1.1 302 Found  
Location: https://sp.example.com/fastfed/handshake/receive?  
  initial_access_token=kj2kj3ujwldnl  
  &nonce=bpqolop4u4pjoe  
  &fastfed_metadata_uri=https%3A%2F%2Ftenant12345.example.com%2Ffastfed%2Fmetadata  
  &return_to=https%3A%2F%2Fidp.example.com%2Ffastfed%2Fhandshake%2Ffinish  
  &state=vk2j35ijlkt2j2oij3ti2jtkl1tkl2n4k12n
```

(Step 6) SP registers the relationship

The service provider receives the request and begins its registration process.

When the service provider has finished any custom configuration, which are outside the scope of the FastFed specification, it then performs the following FastFed actions behind the scenes:

- Download the Identity Provider's FastFed Metadata and extract the attributes from it, using the initial_access_token to access the file.
- Examine the metadata to ensure compatibility. This may include examining the authentication protocols, the supported attributes, and specific details within the SAML/OIDC configuration files. If incompatibilities exist, the service provider stops and displays an error message.
- Uses the OAuth endpoint from the Metadata to convert the initial_access_token + nonce into an access_token + refresh_token. These will be used for future communications where the SP must initiate communication with the IdP. **(Open Question: This is an unorthodox OAuth flow. Doesn't match any existing grant type. Is there a better approach? Needs a deep security review.)**
- Generate a corresponding initial_access_token and nonce that will be given to the IdP. This will allow the IdP to complete its side of the handshake later.
- If using OIDC, perform OIDC Dynamic Registration with the IdP as per the standard OIDC specifications. The registration endpoint can be discovered via the oidc-configuration provided within the IdP's FastFed metadata. (Note – To be FastFed compliant, IdPs must support dynamic registration if they support OIDC).

This document is for discussion purposes only, and not an Implementers Draft or Final Specification

- Generate (or otherwise make available) the service provider's FastFed metadata to share with the IdP. Access to the file can be restricted to holders of the `initial_access_token`. The file contains the following information:
 - Name (*Required*)
 - A displayable name for the Service Provider (**Open Question – limits on name length?**)
 - Logo URI (*Optional*)
 - A logo that represents the Service Provider (**Open Question – limits on size & image format?**)
 - Auth Protocols (*Required, at least one*)
 - Describes whether the SP will support SAML, OIDC, or both for the given identity provider.
 - SAML Metadata URI (*Required if the AuthProtocols include SAML*)
 - A URL for a standard SAML Metadata document containing an SPSSODescriptor. Access to this file may require the `access_token`.
 - OAuth Token Endpoint (*Required*)
 - A URL for a standard OAuth token exchange using the `refresh_token`.
 - SCIM Endpoint (*Required*)
 - A URL for standard SCIM operations exposed by service providers, such as those necessary for user provisioning.
 - Provisioning Mode (*Required*)
 - Describes how the IdP should provision users into the service. Options include: None, JIT, Preprovision... (**Open Question: what are the right values here?**)
 - Desired Attributes (*Required*)
 - Describes which user attributes the SP would like to receive from the IdP, and whether they are essential or optional.
 - Attribute Mapping (*Optional*)
 - Describes how to map SCIM attributes into SAML Attributes or OIDC Claims. This mapping can be necessary to maintain compatibility with existing infrastructure.

At the end of this process, the SP is capable of giving access to a configuration file like this:

```
GET /fastfed/metatdata HTTP/1.1
Host: tenant56789.example.com
Authorization: Bearer Pk39smWk2L9s6
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "service_provider": {
    "name": "Awesome Service (Tenant 56789)",
    "logo_uri": "https://example.com/images/sp_logo.png",
    "auth_protocols": ["SAML", "OIDC"],
    "saml_metadata_uri": "https://tenant56789.example.com/saml-metadata.xml",
    "token_endpoint": "https://tenant56789.example.com/token",
    "scim_endpoint": "https://tenant56789.example.com/scim",
```

```
"provisioning_mode": "None",
"desired_attributes": {
  "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User"],
  "userName": {"essential": true}
  "name": {
    "formatted": null
  },
  "emails[primary == true]": {
    value: {"essential": true}
  }
},
"saml_attribute_map": {
  "name_id": {
    "format": "urn:oasis:names:tc:SAML:2.0:nameid-format:persistent",
    "value": "{$user.userName}",
  },
  "attributes": [
    {"name": "name",
     "format": "urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified",
     "value": "{$user.name.formatted}"
    },
    {"name": "email",
     "format": "urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified",
     "value": "{$user.email[primary == true].value}"
    }
  ]
},
"oidc_claim_map": {
  "sub": "{$user.userName}",
  "name": "{$user.name.formatted}",
  "email": "{$user.email[primary == true].value}"
}
}
```

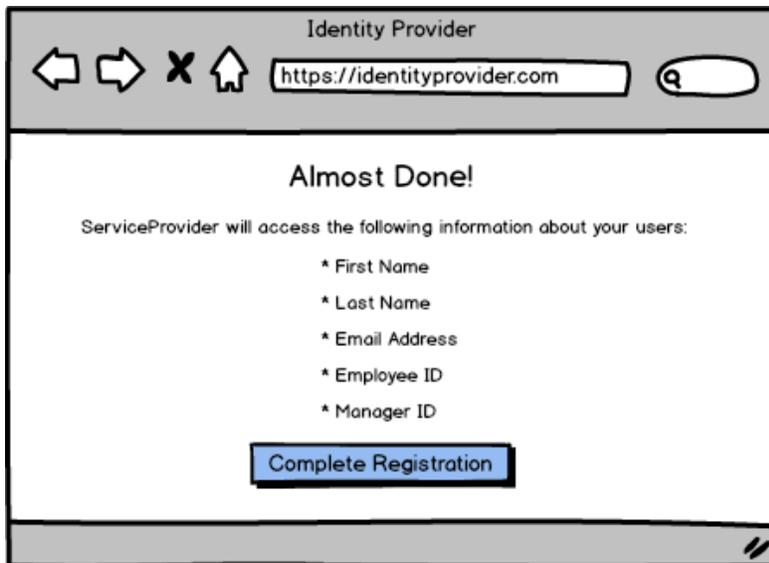
(Step 7) SP redirects the user back to the IdP

The last remaining step is to redirect the end-user back to the IdP.

The redirection is accomplished by issuing an HTTP 302 to the location originally provided by the IdP in the *return_to* URL. As part of the return parameters, the SP provides access tokens and the location of the SP's FastFed Metadata. It also echoes back the *state* originally provided by the IdP.

Example:

```
HTTP/1.1 302 Found
Location: https://ipd.example.com/fastfed/handshake/finish?
  initial_access_token=po23opm2d90S3K3q
  &nonce=9K2j3oH83jwi39Nc
  &fastfed_metadata_uri=https%3A%2F%2Ftenant56789.example.com%2Ffastfed%2Fmetadata
  &state=vk2j35ijlkt2j2oij3ti2jtkl1tkl12n4kl2n
```



(Step 8) IdP completes the registration

The identity provider uses the information provided by the SP to complete the registration. This involves the following steps:

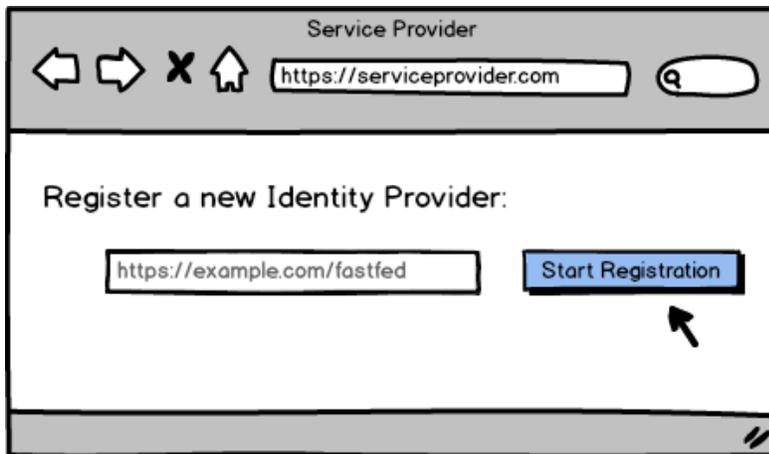
- Download the Service Provider's FastFed Metadata, using the `initial_access_token` to access the file.
- Uses the OAuth endpoint from the Metadata to convert the `initial_access_token` + `nonce` into an `access_token` + `refresh_token`. These will be used for future communications where the IdP must initiate communication to the SP. **(Open Issue: This is the same unorthodox OAuth flow as in Step 6.)**
- Finalize the registration and "turn-on" SSO to the service provider for end-users. The mechanisms of doing this are specific to each IdP, but will typically involve persisting the SSO information into some form of database, granting permissions, and scheduling recurring activities, such as key rotations.

SP-Initiated Registration

The previous flows described an IdP-initiated registration. The same flow can be initiated from the service provider.

To start this flow, as an example, the SP allows the end-user to choose an existing IdP, or provides the FastFed URL for a custom provider.

This document is for discussion purposes only, and not an Implementers Draft or Final Specification



Upon submission, the SP loads the FastFed Discovery configuration from the specified URL.

```
GET /fastfed HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: application/json

{
  "identity_provider":
    {"handshake_endpoint": "https://idp.example.com/fastfed/handshake/start"}
}
```

It extracts the handshake endpoint from the configuration and sends the user to “Step #3” in the previously described flow by redirecting them to the handshake endpoint, including the SP’s Discovery URL as a parameter.

Example:

```
HTTP/1.1 302 Found
Location: https://ipd.example.com/fastfed/handshake/start?sp=https%3A%2F%2Fsp.example.com%2F.well-known%2Ffastfed
```

From this point, the registration flow proceeds as previously described.

Post-Registration Activities

After registration is complete, there remain ongoing activities as part of the SSO relationship, including::

Sign-In

When an end-user signs in to an application, the normal SAML/OIDC flows occur. However, the IdP must apply any FastFed attribute mappings defined by the SP when performing these flows. **(See Open Issues at end of document for discussion on whether attribute mappings are needed.)**

User Provisioning

Some applications require ongoing user provisioning. Such applications declare the *provisioning_mode* in their FastFed Metadata and, if necessary, provide a SCIM endpoint for provisioning. The IdPs must respect and execute the declared provisioning mode.

Key Rotation

Both SAML and OIDC rely on key materials. All key materials must be rotated. **(Open Question – Is this enforceable? Should FastFed recommend best practices for key rotation?)** .

Open Issues

Throughout this document, a number of questions and issues were highlighted. This section consolidates the issues and can serve as a reference point for discussions.

(Issue #1) Names, Names, Names

Every design requires the obligatory hand-wringing over terminology. Many names in this document may change.

In particular, FastFed is unique in that it spans three existing specs, each with unique terminologies. Does FastFed foist a 4th terminology on the world?

(Issue #2) Balancing Ease-of-Use with Feature Richness

FastFed sets a high bar for user simplicity. Delivering this experience necessitates restrictions in the functionality that IdPs and SPs expose. How restrictive can FastFed be without crippling adoption of the specification? Ideally, the familiar 80/20 rule can be achieved; solving 80% of the integrations with this simple experience.

For example, can we mandate that:

- SAML NameID must always be populated with the SCIM *userName* and be of type “unspecified”. Otherwise, administrators must define a mapping, which fails the usability goal.
- No extended attributes are allowed, because this will require end-users to define more attribute mappings. FastFed provides a predefined list of supported SCIM schemas. (*Note – this would probably require a new blessed schema for the educational sector, to substitute for the gaps between eduPerson and SCIM User/EnterpriseUser.*)
- Key rotation specifics.
- Provisioning, if required, must use SCIM. There will be a small number of allowed provisioning modes (e.g. JIT, PreProvisioning). Modes must be easily understandable by non-experts in Identity.

(Issue #3) Attribute Mappings

This document proposes SCIM as the *lingua franca* for user attributes. However, neither SAML nor OIDC describe how to bind SCIM attributes into the protocol.

This document is for discussion purposes only, and not an Implementers Draft or Final Specification

One option is to define “the one true way” that all FastFed compliant systems adhere to for using SCIM in these protocols. For example:

- http://openid.net/specs/openid-connect-scim-profile-1_0.html
- <https://www.ietf.org/mail-archive/web/scim/current/msg01141.html>

Another option is to allow service providers to define custom attribute mappings. These would be defined once, by the SP, and reused across all integrations.

Attribute mappings could ease adoption by allowing service providers to avoid making changes to their existing SAML/OIDC endpoints. They may continue using whatever attribute naming scheme exist today. However, it also increases the complexity of the FastFed specification. It is undetermined whether the benefits outweigh the complexity.

If decided to support attribute mappings, another challenge is the lack of an existing standard for attribute mappings syntax. (Unless someone knows one?) Various providers have invented their own syntax to fill the gap. Examples:

- <https://docs.microsoft.com/en-us/azure/active-directory/active-directory-scim-provisioning>
- <https://developers.onelogin.com/scim/define-user-schema>
- <https://docs.wso2.com/display/IS600/Managing+Claim+Dialects>

Many of these rely on some form of “JSON pointer” to reference a SCIM attribute within a JSON document. There is no applicable standard for this, either. There is a JSONPointer specification (<https://tools.ietf.org/html/rfc6901>) but it is not rich enough to support SCIM multivalued attributes. There is also JSONPath which can express the richness, and is widely adopted, but the specification consists solely of user docs on a personal blog. (<http://goessner.net/articles/JsonPath/>)

(Issue #4) Credential Exchange

FastFed needs to distribute credentials to the IdP and SP so they can communicate with each other. OAuth tokens are nice, but there is no existing OAuth grant type that aligns with the FastFed needs. An unorthodox flow was proposed in this document. However, the security implications of this proposal have not been examined, and there may be better approaches to credential exchange.

(Issue #5) Endpoints on the Public Internet

The flows described in this document require that an SP be able to communicate programmatically with the IdP. For example, to load the FastFed Metadata configuration, refresh OAuth tokens, and rotate keys.

Many installations of SAML IdPs exist in private networks with firewall restrictions. They may not be able to call (or be called) via endpoints on the public Internet. This has not mattered in the past because SAML relied on the user_agent as a data carrier.

This document is for discussion purposes only, and not an Implementers Draft or Final Specification

If this is a constraint for enough potential users, it may be necessary to define a FastFed flow that performs the registration handshake through the user_agent via HTTP POSTS, and negates the need for public endpoints. This profile may only support a limited subset of FastFed functionality.

(Issue #6) Minimal Set of SAML/OIDC functionality to be compliant

Both SAML and OIDC define a variety of capabilities that may not be necessary for the FastFed experience. To reduce the burden on FastFed implementers, and encourage compatibility, should FastFed define the minimum set of capabilities within these specifications that must be implemented to be FastFed Compliant?

(Issue #7) Entitlements

Some applications offer different profiles/roles that users can attain. The permissions to do so are sometimes represented in extended attributes on the user profile, inside the Identity Provider.

Custom attributes are generally in conflict with the FastFed goals because they require an administrator to define/configure additional data. However, the need for profile management still exists. Can FastFed support these custom attributes without sacrificing usability?

If there is enough commonality in the representation of entitlements, it might be possible for FastFed to define a mechanism for Service Providers to declare the set of profiles/roles that can be attained.

However, this is a slippery slope. Authorization rules become complicated very quickly. It is undetermined if this is possible or appropriate for FastFed to address.

(Issue #7) Duplicate Registrations

How does FastFed identify a unique registration between an IdentityProvider and a ServiceProvider?

The answer is not clear-cut because some applications allow multiple instances. For example, there might be “Production” and “Sandbox” instances of the same SaaS application. Or, different departments in a company might create instances for “Sales”, “Finance”, “Vendors”, etc...

What is the mechanism by which FastFed implementers can recognize that a registration to a given instance of an app already exists?

(Issue #8) Evolution of IdP and SP configurations

How does the relationship evolve over time? For example, perhaps the service provider wishes to update their logo. Or, the IdP administrator wishes to switch from SAML to OIDC.

Should the IdP and SP poll the other’s FastFed Metadata files on a recurring cadence? Alternatively, should administrators “re-register” to make changes? The use cases still need to be enumerated and solutions proposed.

This document is for discussion purposes only, and not an Implementers Draft or Final Specification

(Issue #9) Other Edge Cases

There are various edge cases that were not considered by this document. For example, what happens if an administrator abandons a FastFed registration midway? Does information get cleaned up or expired eventually? What happens if the administrator later attempts to retry the registration? These edge cases still need to be identified and addressed.