	P. Dingle
	Ping Identity
	T. Bray
	Google Inc.
	May 27, 2016

OpenID AccountChooser Basic API Profile 1.0

draft-accountchooser-basic-profile-08

Abstract

AccountChooser is a facility where users can store basic identifying information for accounts that they use for signing in to Web sites (Sites). Once stored, users are able to use the AccountChooser user interface to transmit simple account hint information such as email addresses or federated issuer identifiers to sites rather than typing that information manually into login or signup forms.

This specification standardizes a simple JavaScript interface that can be offered by an AccountChooser, enabling Sites to embed AccountChooser functionality into login and signup pages, and allowing Sites to push new Account Records into the Chooser. By manipulating a JSON object passed as a javascript variable to an API maintained by the Chooser, Sites may interact with the Chooser without writing code.

For developers wishing to build or deeply integrate with an AccountChooser programatically, or for a deeper understanding of hot and cold choosers, storage best practices and privacy recommendations, see the OpenID AccountChooser Developer Specification 1.0 document.

Table of Contents

1. Introduction

The most common representation of a relationship between a user and a web site today is an account - a collection of attributes and settings, organized around a unique identifier such as an email address, and protected by a locally stored credential, most often a password. Creation, use and management of accounts are tedious and repetitive activities, as users end up typing the same information over and over again. Often users can feel challenged to recall which set of identity information has been supplied to which site.

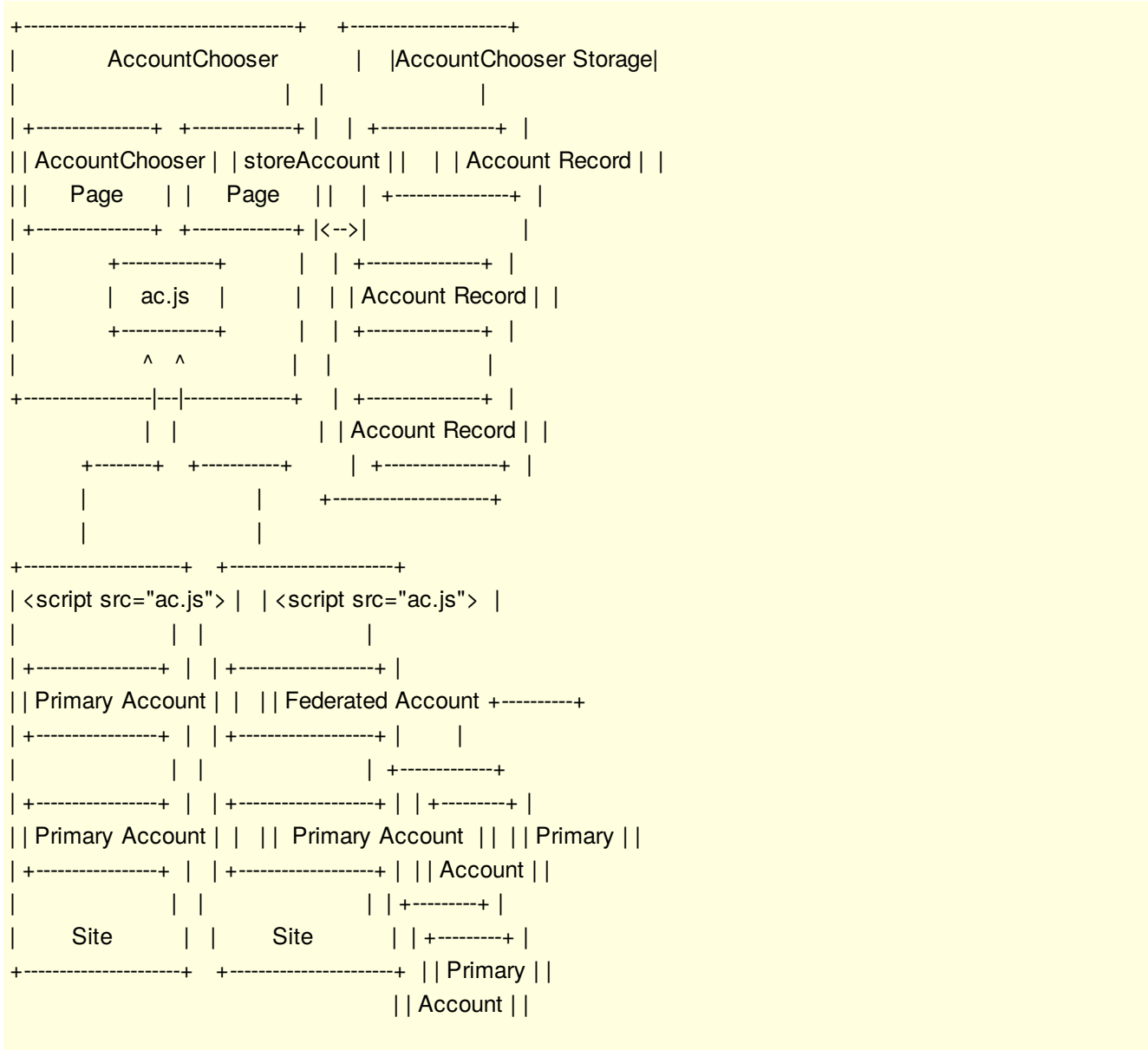
In addition to direct account management, sites may optionally offer federated account options, where the site does not store a credential for the account directly, but instead relies on another service (such as a social network or corporate federation service) to validate the identity of the user. In such a case, a user

returning to a site may not only need to remember which username they used with a site, but also which federated identity provider they linked to the site.

From the perspective of a user, an AccountChooser is a place where the user can store records of their different accounts, and reuse those records easily across multiple sites. The visual nature of an AccountChooser provides additional cues to remember which identities are stored where, and the act of "choosing" rather than typing means that login and sign-up are faster and less error-prone. The AccountChooser can be queried by sites, and while the user sees all the account records they might want to pass back to the site for the purposes of login or sign-up, the site only receives the identity information that the user chooses.

From the perspective of a site, the chooser is a way to help users remember their login information, thus reducing the likelihood of account abandonment and decreasing drop-off that may occur during login and signup. By embedding the AccountChooser Javascript into login and signup pages, the site is able to populate those pages with Account Record information received from the AccountChooser rather than directly from the user. For sites that support federated authentication, an AccountChooser performs both provider discovery and user discovery. Newer federation standards such as OpenID Connect support the inclusion of a "login hint" in the authentication request, which enables the provider in turn to operate with greater understanding of user and possibly session context.

This document standardizes a design pattern for behavior of an AccountChooser and for a javascript-specific interaction between a Site and an Account Chooser. A production instance of an AccountChooser exists today at <https://accountchooser.com>. All examples derive from this reference implementation.



```
| +---------+ |
|           |
|    Site   |
| (Identity |
|  Provider)|
+-----------+
```

1.1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2. Terms

For the purpose of this specification, the following terms are defined.

1.2.1. Account

A collection of information related to an individual user, containing at least one unique identifier.

1.2.2. Primary Account

An account that is validated directly at the site, for example by prompting the user for a password.

1.2.3. Federated Account

An account that is not validated directly at the site, but instead validated by an external entity such as a social network or a corporate IAM system.

1.2.4. Account Record

A tightly limited set of account attributes stored by an AccountChooser and passed to a Site.

1.2.5. Site

A website or other service at which users have accounts. Accounts may be primary accounts or federated accounts.

1.2.6. AccountChooser (AC)

A mechanism to remember, on behalf of a person using a browser, the Accounts they used to sign in to Web sites so that rather than typing in usernames, they need only pick an account from a list.

NOTE: The user is presented a list of her own Accounts so that she need only pick one from the list rather than typing in a username to login.

1.2.7. AccountChooser Storage

A persistent database maintained by the AccountChooser, containing zero or more Account Records.

1.2.8. AccountChooser JavaScript (ac.js)

Javascript file invoked by the Account Holder that performs AccountChooser tasks.

1.2.9. AccountChooser JavaScript File Reference

The script element used at the Site to invoke ac.js. The example file reference used in this document is <script type="text/javascript" src="https://www.accountchooser.com/ac.js" />

1.2.10. AccountChooser CONFIG Object

A JSON object that accompanies an ac.js call and passes parameters from the site to the AccountChooser.

1.2.11. Discovery Context

A JSON array of URNs that describe the context of the account record and can be used to filter account records displayed by the chooser.

1.2.12. Login Hint

An attribute within an account record that helps to identify the user trying to login or sign up. Login hints can either be email addresses or tokens.

1.2.13. Login Hint Description

An attribute used to explain the origins or context of a login hint. Required for Login Hint Tokens.

1.2.14. Login Hint Token

An attribute containing information identifying a user, which may not be human-readable. Examples of Login Hint Tokens might include JWT tokens or OpenID Connect id_tokens.

1.2.15. User Status Object

A JSON object that represents the current state of the account at the site.

1.2.16. User Claimed Identifier

The account record attribute that uniquely identifies the user. This specification uses the email field as the User Claimed Identifier.

1.2.17. Display Name

A human-readable name for the person associated with the user claimed identifier.

1.2.18. Provider ID

A URL identifying the service capable of validating the account referenced in the account record. In the case where the service specified by the Provider ID is not the Account Holder service, the Account Holder will likely make a federated call to the service indicated in order to validate the account, rather than locally validating the account.

2. Account Records

An account record is a collection of attributes that are passed between Sites and AccountChoosers during invocation of the AccountChooser JavaScript. Examples of account record operations include requesting an account record to be chosen, checking the status of a chosen account record against an existing account at a site, or requesting that a new account record be stored. The AccountChooser negotiates with the user to complete those requests. There are two ways that account record attributes can be grouped, primary account

records and federated account records. Primary account records contain only user information, which can be used at the Site for login or sign-up. Federated Account records additionally contain an affiliation to a remote domain, which a site could use to initiate a federated authentication request.

2.1. Primary Account Record Object

A primary account record object contains only user information, with no affiliation to a provider.

name	type	default value	maximum size	description
email	string	N/A	128 chars	Email address and user claimed identifier.
discoveryContext	array of string	N/A	8 array elements, 128 chars each	JSON array of URNs used for filtering results.
displayName	string	N/A	128 chars	Display Name for the user.
photoUrl	string	N/A	2048 chars	A URI from which a photo of the person can be fetched. Must use the HTTPS scheme.

The account record storage object is a JSON object. The email attribute MUST be present in a primary account record. When a providerId attribute is included, the object is no longer considered to be primary.

email

REQUIRED. This string is REQUIRED and MUST be considered as the user claimed identifier in the case where no providerId is present. This string SHOULD be formatted as an email address. Federated providers whose user claimed identifiers are not email addresses MAY store a non-email-format string in this field.

discoveryContext

OPTIONAL. A JSON array containing one or more string values, representing properties of the account record used to filter which records to display to the user.

displayName

OPTIONAL. This string SHOULD contain a short, human-readable description of the user and/or account.

photoUrl

OPTIONAL. If populated, this value MUST describe an absolute URI containing an HTTPS scheme. The domain portion of the URI MUST match the referring domain at time of account storage.

An example of a primary account record object with an opaque login hint token, formatted as a JSON object, is shown below:

```
<script type="text/javascript">
   accountchooser.CONFIG.storeAccount = {
      email: "nikhil_corlett@yahoo.com",
      displayName: "Nikhil Corlett",
      photoUrl: "https://example.com/nikhil_corlett.jpg"
      };
</script>
```

2.2. Federated Account Record Object

A federated account record contains both user information and an affiliated provider identifier.

name	type	default value	maximum size	description
email	string	N/A	128 chars	Email address. Possible user claimed identifier.
discoveryContext	string	openid:ac:login_hint_email	8 array elements, 128 chars each	JSON array of URNs used for filtering results.
displayName	string	N/A	128 chars	Display Name.
loginHintDescription	string	N/A	128 chars	Login Token Description.
loginHintToken	string	N/A	2048 chars	Login Token. Possible user claimed identifier.
photoUrl	string	N/A	512 chars	A URI from which a photo of the person can be fetched. Must use the HTTPS scheme.
providerId	string	N/A	128 chars	Domain name identifying a federated Identity Provider.

The account record storage object is a JSON object. At least one user claimed identifier MUST be present in every account record. When a providerId attribute is absent, the object is no longer considered to be federated.

email

OPTIONAL. This string is OPTIONAL if a Login Hint Token is present, but is REQUIRED and MUST be considered as the user claimed identifier in the absence of a Login Hint Token. This string SHOULD take the form of an email address. Federated providers whose user claimed identifiers are not email addresses MAY store a non-email-format string in this field.

discoveryContext

OPTIONAL. A JSON array containing one or more string values, representing properties of the account record used to filter which records to display to the user.

displayName

OPTIONAL. This string SHOULD contain a short, human-readable description of the user and/or account.

loginHintDescription

OPTIONAL. This string MAY contain a further description of the Account Record. Login Hint Description is REQUIRED if loginHintToken is present.

loginHintToken

OPTIONAL. A string expected to be opaque to the user, that can be used by a provider to identify the user attempting to login. For example, a Site could pass the loginHintToken to a provider during a federated authentication request. If present, loginHintToken SHOULD be considered as user claimed identifier.

photoUrl

OPTIONAL. If populated, this value MUST describe an absolute URI containing an HTTPS scheme.

The domain portion of the URI MUST match the referring domain during account record storage.

providerId

REQUIRED. A string representing the domain of the provider that has created the entry.

Two non-normative and fictional examples of federated account records are shown below (for further discussion of use cases, see Section 10):

```
<script type="text/javascript">
  accountchooser.CONFIG.storeAccount = {
  displayName: "Nikhil Corlett",
  loginHintDescription: "Identity Example",
  loginHintToken: "FGRTSDH6sdflkj3jg.45fgljsd.ey0",
  photoUrl: "https://identity.example.com/nikhil_corlett.jpg"
  providerId: "identity.example.com" };
</script>
```

```
<script type="text/javascript">
  accountchooser.CONFIG.storeAccount = {
  discoveryContext: any:twitterhandle
  displayName: "Mars Rover",
  loginHintDescription: "@marsrover Twitter Handle",
  loginHintToken: "@marsrover",
  photoUrl: "https://twitter.com/photos/marsrover"
  providerId: "twitter.com" };
</script>
```

2.3. Account Record Object Handling

When an account record is passed to ac.js in the storeAccount variable, the following restrictions MUST be met for the storage operation to succeed. The store account context is described in Section 8.

- The account record MUST contain at least one user claimed identifier.
- The account record object MUST NOT contain attributes beyond those listed in Section 2.
- The loginHintToken and loginHintDescriptions MUST NOT be present unless the providerId attribute is also present.

When an account record is passed to ac.js in the storeAccount variable, the following restrictions MUST be met for the specified attribute to be included in the stored record:

- The photoUrl attribute MUST have a URI scheme of HTTPS.
- The domain portion of the photoUrl URI MUST match the domain of the site executing the storeAccount page.
- The discoveryContext attribute MUST NOT contain elements prefixed with openid:ac.

All attribute values supplied in the storeAccount variable SHOULD be sanitized according to OWASP recommendations.

3. AccountChooser JavaScript

The method by which a site interacts with AccountChooser is a JavaScript file named ac.js, which performs a variety of tasks depending on where it is called from, and can be customized with configuration arguments.

The ac.js file may either be hosted with the site domain, or referenced remotely from the AccountChooser domain. This guide uses the default convention of a remotely referenced file from

https://www.accountchooser.com/ac.js.

Sites trigger user interaction with the AccountChooser by invoking the ac.js file from three different user contexts: user signup, user login, and after successful user authentication (generally from the authenticated home page). In addition, a programmatic status checker URL can be implemented by the Site.

3.1. AccountChooser JavaScript Variables

The AccountChooser ac.js file establishes the following javascript variables:

accountchooser

A javascript variable defined by ac.js that can be manipulated by the site.

accountchooser.CONFIG

A javascript variable which is a child element of the accountchooser variable. Used to pass configuration requests. It contains an AccountChooser CONFIG Object.

accountchooser.CONFIG.uiConfig

A javascript variable which is a child element of the accountchooser.CONFIG variable. Used to pass branding information. It contains an AccountChooser Branding Object (Section 3.4).

3.2. AccountChooser CONFIG Object

Values of the accountchooser.CONFIG variable represent configuration parameters sent from the Site to the AccountChooser at the time the ac.js file is invoked. The contents of accountchooser.CONFIG must conform to the elements listed below:

Note: The value of a URL parameter (loginUrl, signupUrl, userStatusUrl, or homeUrl) can be an absolute URL or a relative reference [RFC3986]. Relative references are resolved against the URL of the web page.

Following is the list of the configuration parameters:

name	type	default value	description
loginUrl	string	'account-login'	Location of the primary login form
signupUrl	string	'account-create'	Location of the user registration page.
userStatusUrl	string	'account-status'	Location of the home web page.
homeUrl	string	'/'	Location of the user registration page.
siteEmailId	string	'email'	Email input field ID (HTML id= attribute value) in the signup and login forms.
sitePasswordId	string	'password'	Password input field ID in the signup and login forms.
siteDisplayNameId	string	'displayName'	Display name input field ID in the signup form. Can be set to null or empty if your signup form does not need the user's name.
sitePhotoUrlId	string	'photoUrl'	Profile photo URL input field ID in the signup form. Can be set to null or empty if your signup form does not need the user's profile picture.

name	type	default value	description
mode	string	N/A	Mode for the current page. The possible values are 'login' and 'signup'. ac.js normally sets the mode by checking the current URL against the loginUrl and signupUrl parameters, but this parameter may be used to override that behavior.//TODO: are absolute URLs outside of the current origin allowed?
uiConfig	object	N/A	Interface customization object (defined in Section 3.2)
language	string	'en'	The display language of the AccountChooser page.
providers	JSON array	N/A	List of supported federation providers, usually given by top-level domain name, for example facebook.com. //TODO add xref to place where providers list is defined
discoveryContext	JSON array	N/A	List of URNs restricting type of account records to display, see Section 3.3 for more information.
storeAccount	object	N/A	Triggers the storage of an Account Record Object (defined in Section 2).

3.3. Filtering by Discovery Context

Population of the accountchooser.CONFIG.discoveryContext variable during login and signup can be optionally used to filter which account records are shown to the user. The discoveryContext variable may contain either reserved contexts that are specified below or custom contexts supplied by the provider at time of account record storage.

URN	description
openid:ac:login_hint_email	Describes account records containing an email address attribute
openid:ac:login_hint_token	Describes account records containing a login hint token
openid:ac:federated	Describes account records containing a provider identifier
openid:ac:primary	Describes account records with no provider identifier

If accountchooser.CONFIG.discoveryContext contains more than one array element, the resulting filter MUST constitute the intersection of all array members. For example, if accountchooser.CONFIG.discoveryContext is set to ["openid:ac:federated","openid:ac:login_hint_email"], the chooser will only show account records with both a providerId attribute and an email attribute, and will not show any federated account records that only contain loginHintToken attributes. Note that the providers variable may also affect returned account records.

3.4. AccountChooser Branding Object

A site MAY choose to populate the uiConfig parameter of the AccountChooser CONFIG Object in order to customize some of the AccountChooser page shown to end users. If set, the uiConfig parameter MUST contain a JSON object with at least one of the following members:

name	type	default value	description
title	string	N/A	Suggested title for AccountChooser Window/Tab (note this is window title not page title).
favicon	string	N/A	URI identifying a favicon to display in the address bar of theAccountChooser Page.
branding	string	N/A	URI identifying a resource with media-type text/html to be displayed on the AccountChooser page as visual identification of the site. MUST be HTTPS.

An example interface configuration object:

```
accountchooser.CONFIG.uiConfig = {
    title: 'Sign in to Example.com',
    favicon: 'https://example.com/favicon.ico',
    branding: 'https://example.com/branding/ac-blurb.html'
}
```

4. Overview of Page Interaction

To successfully integrate AccountChooser into a site, the site will modify two existing pages, and create two new pages that execute transparently to the user.

The site MUST modify:

- The login page
    - page where users normally authenticate. AccountChooser does not replace this page, but instead populates values within the page if an account record is chosen.
- The signup page
    - page where users normally register. AccountChooser does not replace this page but instead populates values within the page if an account record is chosen.

The site MUST create:

- The user status page
    - page called by AccountChooser to determine where to redirect the user. Different contexts will result in different results.

The site MAY create:

- The store account page //TODO: diagram shows storeAccount Page as being at Chooser not site
    - page called by a site to create or update an account record at AccountChooser.

Other pages are specified by the site as redirection targets, however these pages do not contain references to ac.js and do not interact with AccountChooser.

- Home URL
    - primary page of the site. Used as a final redirection after completed AccountChooser events such as account storage.
- Auth URI
    - Location for initiation of a federated identity request. This URI could represent a "startSSO" link at the site where an authentication request is constructed, or constitute an already constructed authentication request URI containing everything a federated identity provider needs.

5. Login Page Requirements

When ac.js is embedded into a login page and configured with the correct CONFIG object, the user experience is only interrupted in the case where account records exist in the user's AccountChooser. In the case where there are no records, ac.js immediately exits, leaving the user to authenticate without assistance. Once that authentication occurs, the site can then make an Account Record Insertion request, such that the next time the user visits the Login Page, they will have an account record populated.

5.1. Login Page Required Elements

The following <script> objects MUST be placed into the web page such that they will be executed prior to the population of the returned page:

- The AccountChooser Javascript file reference (Section 1.2.9).
- The accountchooser.CONFIG variable, populated with a JSON object containing values described in Section 3.2.

The site SHOULD place the objects in the HEAD element of the web document. The site MAY place the object in the BODY element of the document, but the scripts MUST occur prior to display of any page elements.

5.2. AccountChooser CONFIG Object Values

The site MUST perform the following configurations:

- Specify a login context
    - The site MAY indicate login mode by setting the loginUrl variable. When ac.js compares the URL of the currently loaded login page to the loginURL value, the values MUST match for ac.js functionality to run.
    - The site MAY indicate login mode by setting the mode variable to the string login. If both loginUrl and mode are set, the value of the mode variable takes precedence.
    - The site SHOULD NOT populate the storeAccount variable. If the storeAccount variable is populated, ac.js will execute an Account Record Insertion request rather than a Login request.

- Indicate the location of the User Status Checker
    - The site MUST implement an user status checker.
    - The site MAY specify the location of the user status checker by setting the userStatusUrl variable. If the userStatusUrl is not set, ac.js will assume that the user status checker can be found at the default relative reference of account-status.

The site SHOULD place the objects in the HEAD element of the web document. The site MAY place the object in the BODY element of the document, but the scripts MUST occur prior to display of any page elements.

The site MAY perform the following configurations:

- Specify a Signup URL
    - The site MAY indicate the location of the signup page by setting the signupUrl variable. In the case where a user chooses an account record but the status check on the record is negative, ac.js will redirect the browser to value of signupUrl if it is set.

- Specify Login Form DOM Elements
    - The site MAY indicate the DOM element that represents the site login form username element by setting the siteEmailId variable. If this variable is set, ac.js will populate the value of the corresponding DOM element in the case that an account record is successfully selected by the user.
    - The site MAY indicate the DOM element that represents the site login form password element by setting the sitePasswordId variable. If this variable is set, ac.js will set the focus of the form to the password element in the case that an account record is successfully selected by the user. The site SHOULD only set sitePasswordId in conjunction with siteEmailId.

- Recommend a Preferred Language to be used by the AccountChooser User Interface
    - The site MAY indicate a non-standard language preference, by setting the language variable to an [ISO639] compliant string.

- Specify HTML to be used to brand the AccountChooser
    - See AccountChooser Branding (Section 9) for more information.

- Indicate a set of supported federated providers or discovery contexts
    - The site MAY indicate a collection of providers for which account records should be returned. See Section 10.1.
    - The site MAY indicate a collection of discovery contexts for which account records should be returned. See Section 3.3

A non-normative example of a fully specified login page:

```
<html>
  <head>
    <script type="text/javascript"
        src="https://www.accountchooser.com/ac.js" />
    <script type="text/javascript">
      accountchooser.CONFIG={
        loginUrl: "utils/mysitelogin",
        mode: "login",
        siteEmailId: "form_username",
        sitePasswordId: "form_password" };
    </script>
  </head>
  <body>
    <form>
      <input id=&amp;quot;form_username&amp;quot; type=&amp;quot;text&amp;quot;/>
      <input id="form_password" type="password" />
      <input id="submit" type="submit">Login</input>
    </form>
    ...
```

6. Signup Page Requirements

When ac.js is embedded into a signup page and configured with the correct CONFIG object, the user experience is only interrupted in the case where account records exist in the user's AccountChooser. In the case where there are no records, ac.js immediately exits, leaving the user to sign up without assistance. Once the registration is complete, the site can then redirect to the store account page, ensuring that the next time the user visits the site, an account record will be populated.

6.1. Signup Page Required Elements

The following elements MUST be placed into the web page such that they will be executed prior to the population of the returned page:

- The AccountChooser Javascript file reference.
- The accountchooser.CONFIG variable, populated with a JSON object containing values described in Section 3.2.

The site SHOULD place the objects in the HEAD element of the web document. The site MAY place the object in the BODY element of the document, but the scripts MUST occur prior to display of any page elements.

6.2. AccountChooser CONFIG Object Values

The site MUST perform the following configurations:

- Specify a signup context
    - The site MAY indicate signup mode by setting the signupUrl variable. When ac.js compares the URL of the currently loaded login page to the signupUrl value, the values MUST match according to [RFC3986] for ac.js functionality to run.
    - The site MAY indicate signup mode by setting the mode variable to the string signup. If both signupUrl and mode variables are set, the value of the mode variable takes precedence.
    - The site SHOULD NOT populate the storeAccount variable. If the storeAccount variable is populated, ac.js will execute within the store account page context rather than the signup context.

- Indicate the location of the User Status Page
    - The site MAY specify the location of the user status page by setting the userStatusUrl variable. If the userStatusUrl is not set, ac.js will assume that the user status page can be found at the default relative reference of account-status.

The site MAY perform the following configurations:

- Specify a Login URL
    - The site MAY indicate the location of the login page by setting the loginUrl variable. In the case where a user chooses an account record but the status check on the record is positive, ac.js will redirect the browser to value of loginUrl if it is set.

- Specify Signup Form DOM Elements
    - The site MAY indicate the DOM element that represents the signup form identifier element by setting the siteEmailId variable. If this variable is set, ac.js will populate the value of the corresponding DOM element in the case that an account record is successfully selected by the user.
    - The site MAY indicate the DOM element that represents the signup form password element by setting the sitePasswordId variable. If this variable is set, ac.js will set the focus of the form to the password element in the case that an account record is successfully selected by the user. The site SHOULD only set sitePasswordId in conjunction with siteEmailId.
    - The site MAY indicate DOM elements representing signup form input boxes by populating any one of the following variables. If the variable is populated, the DOM element listed will be populated by ac.js with values from the account record chosen by the user:
        - siteDisplayNameId
        - sitePhotoUrlId

- Recommend a Preferred Language to be used by the AccountChooser User Interface
    - The site MAY indicate a non-standard language preference, by setting the language variable to an [ISO639] compliant string.

- Specify HTML to be used to brand the AccountChooser
    - See Section 9, AccountChooser Branding for more information.

- Indicate a set of supported federated identity providers
    - The site MAY indicate a collection of providers for which account records should be returned. See Section 10, Federated Accounts.

A non-normative example of a signup page:

```
<html>
```

```
<head>
  <script type="text/javascript"
        src="https://www.accountchooser.com/ac.js" />
  <script type="text/javascript">
  accountchooser.CONFIG={
    loginUrl: "utils/register_now",
    mode: "signup",
    siteEmailId: "form_username",
    sitePasswordId: "form_password",
    siteDisplayNameId: "form_displayame",
    sitePhotoUrlId: "form_avatar_location" };
  </script>
</head>
<body>
  <form>
    <input id=&amp;quot;form_username&amp;quot; type=&amp;quot;text&amp;quot; />
    <input id="form_displayname" type="text" />
    <input id="form_avatar_location" type="text"/>
    <input id="form_password" type="password" />
    <input id="form_confirmpassword" type="password" />
    <input id="submit" type="submit">Login</input>
  </form>
  ...
```

7. User Status Page

The User Status Page is invoked by the AccountChooser after a user chooses an account record. The outcome of the User Status check determines which page the user is redirected to (the Login Page, the Signup Page, or possibly a federated identity provider).

The site MUST implement a user status page.

7.1. User Status Page Request

The AccountChooser will perform an HTTP POST to the URL specified in the userStatusUrl. If no value is set for userStatusUrl, the AccountChooser will POST to the relative reference noted as the default value for userStatusUrl in Section 3.2. Parameters returned in the HTTP POST will correspond to the populated attributes of the account record, defined in Section 2.1 and Section 2.2. At a minimum, the AccountChooser MUST return either an email or loginHintToken parameter.

7.2. User Status Page Response

The following objects MUST be placed into the web page header such that they will be executed prior to the population of the returned page:

- The AccountChooser Javascript file reference.
- A mime type of "application/json".

A non-normative example of a User Status Page with AccountChooser integration:

```
<html>
  <head>
    <script type=&amp;quot;text/javascript&amp;quot;
          src=&amp;quot;https://www.accountchooser.com/ac.js&amp;quot; />
    <meta http-equiv="content-type" content="application/json">
```

```
</head>
<body>
...
```

The response body MUST be a JSON object conforming to [RFC4627] containing ONE of the following parameters:

registered

> The value is set to true if the site has registered the user with password. If not, false is returned. This value is JSON Boolean.

authUri

> The value is set to the URI at which the appropriate federation protocol with that IDP starts. In this case ac.js will dispatch to that URI, and the subsequent login path depends on how that provider works. You can be guided by the IDP selection given by ac.js, but it is not compulsory. This value is JSON String.

The site MUST NOT return content other than the User Status object. Following are non-normative examples of the User Status object.

```
{ "registered":true }
```

Example 1: The user is registered with a password

```
{ "registered":false }
```

Example 2: The user is not currently registered at the site

```
{ "authUri":"https://idp.example.com/auth/" }
```

Example 3: The user is registered with a federated identity

//TODO: what about error conditions? What does the Account Status Page do if the account record is malformed?

8. Store Account Page

The store account page is hosted at the Site, and invoked to create or update a user's record in the AccountChooser. The site SHOULD implement the store account page, unless that function is known to be delegated to a Federated Identity Provider. Once the account record has been stored, the AccountChooser will redirect the user to the value specified in the homeUrl variable.

The site SHOULD invoke the StoreAccount Page under two circumstances:

1. Upon successful user login;
2. Upon successful creation of a new user account.

8.1. Store Account Page Requirements

The following elements MUST be placed into the web page such that they will be executed prior to the population of the returned page:

- The AccountChooser Javascript file reference.
- The accountchooser.CONFIG variable, populated with a JSON object containing values described in Section 3.2.

The site MAY populate the accountchooser.homeUrl variable to alter the location where the AccountChooser will redirect after account storage is complete. The Site MUST NOT place any other content into body of the web page, as control will not be returned to the page.

If accountchooser.CONFIG.discoveryContext is populated, array members MUST NOT begin with the openid:ac: prefix.

```
<html>
  <head>
    <script type="text/javascript"
          src="https://www.accountchooser.com/ac.js" />
    <script type="text/javascript">
      accountchooser.CONFIG.homeUrl = "./welcomepage.html";
      accountchooser.CONFIG.storeAccount = {
        email: "nikhil_corlett@yahoo.com",
        displayName: "Nikhil Corlett",
        photoUrl: "https://example.com/nikhil_corlett.jpg"
        providerId: "identity.example.com" };
    </script>
  </head>
  <body>
  ...
```

A non-normative example of a Store Account Page:

8.2. Storing Account Records containing Login Hint Tokens

If accountchooser.CONFIG.loginHintToken is present in the storeAccount object, accountchooser.CONFIG.loginHintDescription and accountchooser.CONFIG.providerId MUST also be present. It is RECOMMENDED that any PII or other sensitive information present in the loginHintToken be encrypted when described in the loginHintDescription. If PII or sensitive information is necessary in the loginHintDescription to help a user choose the correct record, the site SHOULD partially obscure the value where possible.

```
<html>
  <head>
    <script type="text/javascript"
          src="https://www.accountchooser.com/ac.js" />
    <script type="text/javascript">
      accountchooser.CONFIG.homeUrl = "./welcomepage.html";
      accountchooser.CONFIG.storeAccount = {
        loginHintToken="G4SGdldpp.NSdorF4S.ey0",
        loginHintDescription="Phone Number ending in 1234",
        discoveryContext="[&lquot;openid:modrna:sso&rquot;]",
        providerId: "mobile.mno.com" };
    </script>
  </head>
  <body>
          ...
```

A non-normative example of a Store Account Page with a LoginHintToken:

9. AccountChooser Branding

Sites MAY configure the Interface Configuration Object and pass it to the AccountChooser to brand content

displayed by the AccountChooser. For security reasons, The AccountChooser will not load any content containing executable code, or served over an unencrypted transport layer. At least one of the following json elements MAY be placed into the accountchooser.CONFIG.uiConfig variable:

title

If populated AccountChooser will replace the <title> element of the AccountChooser Page header with the specified value. This will modify the window or tab title.

favicon

This value indicates a replacement favicon for the Accountchooser default "keyhole" icon. If this value is populated, the following conditions MUST be met:

- The value MUST be a valid absolute URI.
- The scheme of the URI MUST be HTTPS.

branding

If this value is populated, the following conditions MUST be met:

- The value MUST be an absolute URI.
- The scheme of the URI MUST be HTTPS.
- The certificate used to encrypt the transport mode must be trusted by the browser.
- The html markup served at the URI MUST NOT contain javascript code.
- The HTTP response header for the markup served at the URI MUST have a Content-type of text/html.
- The response header must have correct cross-origin resource sharing headers.

A non-normative example of a branding page:

```
<?php
   header('Content-Type: text/html');
   header("Access-Control-Allow-Origin: *");
   header('Access-Control-Allow-Methods: GET, POST, OPTIONS');
   header('Access-Control-Max-Age: 86400');
   header('Access-Control-Allow-Credentials: true');
?>
<html>
   <body>
      <p>Branding Example text to be displayed</p>
   </body>
</html>
```

10. Federated Account Support

If a Site supports federated authentication protocols, AccountChooser can be used both to perform provider discovery, and to perform user discovery, if the federated protocol can support login hints within authentication requests.

Sites MAY implement federated account support. Any requirements noted in this section are only applicable to those sites that do implement federated support.

10.1. Filtering by Provider

Configuration of the login page AccountChooser script can be augmented to inform AccountChooser about which federated providers are supported by the site. This enables AccountChooser to hide account records

that would otherwise result in an error if chosen.

Sites MAY populate the providers variable in the login page or signup page accountChooser.CONFIG object. If the providers variable is populated, the value MUST be a JSON array containing one or more string values. The values populated within the providers value SHOULD directly correlate to the list of federated providers supported by the site. Note that account record display can be additionally filtered using discoveryContext (see Section 3.3).

If a site populates the providers variable, that site SHOULD also be capable of supplying a response to the AccountChooser containing an authentication URI for the provider when requested during the user status check.

```
<html>
  <head>
    <script type=&amp;quot;text/javascript&amp;quot;
          src="https://www.accountchooser.com/ac.js" />
    <script type="text/javascript">
      accountchooser.CONFIG.loginUrl="utils/mysitelogin";
      accountchooser.CONFIG.providers=["yahoo.com","facebook.com"];
    </script>
  </head>
  <body>
  ...
```

A non-normative example of a login page augmented with an identity provider filter list:

10.2. Initiating Federated Requests

AccountChooser is capable of initiating federated requests on behalf of the site. To initiate a federated request, the site passes a response to the AccountChooser during the user status check that instructs the AccountChooser as to where to redirect the browser.

Sites MAY initiate a federated request via the AccountChooser by returning a response body during user status check that contains the authUri element, rather than the registered element

```
{ "authUri":"https://idp.example.com/auth/"}
```

A non-normative example of an authUri returned from a user status check:

11. Privacy and Security Requirements

This section includes Privacy-Specific threats described in section 5.2 of [RFC6973].

11.1. Correlation

Whatever domain hosts the ac.js script has the opportunity to extract account records for the purposes of correlation outside of what is specified by this document. It is RECOMMENDED that Sites only include an AccountChooser file reference operated by a trusted third party domain with an appropriate privacy and security policy.

11.2. Identification by other Parties

Sites SHOULD ensure that all interactions with an AccountChooser, including the ac.js file reference all occur over a TLS-protected channel.

11.3. Requesting Account Records outside of Login/Signup Context

Sites MUST NOT request account records for any reason other than facilitation of login or sign up.

11.4. Code Injection Attempts

The branding field in the uiconfig Configuration Argument causes ac.js to fetch arbitrary HTML and attempt to display it in the context of the AccountChooser page. The site MUST NOT include any values in the HTML that contain executable commands, including but not limited to JavaScript, HTML, or CSS exploits.

11.5. Creation of Account Records not Affiliated to an Account

Sites MUST NOT attempt to create records in the AccountChooser that do not correspond to actual accounts.

12. IANA Considerations

This specification makes no request to IANA registry.

13. Normative References

[ISO639] International Organization for Standardization, "ISO 639-1:2002. Codes for the representation of names of languages -- Part 1: Alpha-2 code", 2002.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M. and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013.

Appendix A. Acknowledgements

The OpenID Community would like to thank the following people for the work they have done in the drafting and editing of this specification.

Nat Sakimura -- Nomura Research Institute

Appendix B. Notices

of the technology described in this specification or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any independent effort to identify any such rights. The OpenID Foundation and the contributors to this specification make no (and hereby expressly disclaim any) warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to this specification, and the entire risk as to implementing this specification is assumed by the implementer. The OpenID Intellectual Property Rights policy requires contributors to offer a patent promise not to assert certain patent claims against other contributors and against implementers. The OpenID Foundation invites any interested party to bring to its attention any copyrights, patents, patent applications, or other proprietary rights that may cover technology that may be required to practice this specification.

Authors' Addresses

Pamela Dingle
Ping Identity
EMail: pdingle@pingidentity.com

Tim Bray
Google Inc.
EMail: tbray@textuality.com