OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 06, 2014

P. Hunt, Ed.
Oracle Corporation
M. Ansari
Cisco
T. Nadalin
Microsoft

July 05, 2013

OAuth 2.0 SCIM Client Registration Profile
draft-hunt-oauth-scim-client-reg-00

Abstract

This specification defines a SCIM endpoint used to register and provision OAuth 2.0 clients to access a OAuth 2.0 protected service API in a just-in-time fashion. This draft profiles how a OAuth 2.0 client may use SCIM and OAuth 2.0 to manage its registration.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 06, 2014.

Copyright Notice

Table of Contents

1. Introduction

The OAuth 2.0 Authorization Framework [RFC6749] is a framework by which client applications are authorized by authorization servers to access software API servers using access tokens issued by a token server. As a framework, OAuth 2.0 enables many different flows by which a client application may obtain an access token including delegated authorization from a user. Most of these flows require that each client have a client identifier and some means of authentication.

In order for an OAuth 2.0 client to work with an OAuth authorization server, it must have previously obtained a client identifier (client_id) and a client credential (such as a password, secret, or authentication token). The OAuth 2.0 authorization framework does not define how the relationship between the client and the authorization server is initialized, or how a given client is assigned a unique client identifier. Further, because many clients (such as mobile applications) are copied for wide distribution, special security considerations are defined for those clients known as "public" clients. Public clients represent a high risk when hundreds to millions of clients share the same authentication credential and client_id. This draft provides a means by which public clients can be issued unique client identifiers to become confidential clients.

This draft profiles using SCIM [I-D.ietf-scim-api] as a just-in-time identity management system enabling every client to register itself with a designated SCIM endpoint to obtain a unique OAuth 2.0 [RFC6749] client_id and authentication credential. In addition to accepting a registration request, the profile specifies how the endpoint can be used to dynamically assign a client identifier, and optionally a client credential. This specification defines OAuth 2.0 client schema and objects accessible through SCIM. The OAuth 2.0 client schema includes metadata about the client software being registered as well as OAuth 2.0 protocol metadata required for the successful operation of a client in an OAuth 2.0 protected environment.

Finally, this specification also defines how a publisher of software services APIs deployed in multiple environments may issue a software assertion that may be used by the registration endpoint to recognize and accept client software. Additionally, the specification also defines a mechanism for pre-approval of client software within an administrative domain.

1.1. Notational Conventions

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].

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

1.2. Terminology

This specification uses the terms "Access Token", "Refresh Token", "Authorization Code", "Authorization Grant", "Authorization Server", "Authorization Endpoint", "Client", "Client Identifier", "Client Secret", "Protected Resource", "Resource Owner", "Resource Server", and "Token Endpoint" defined by OAuth 2.0 [RFC6749].

In RFC6749, a client identifier was typically given to a developer for inclusion in the software. As a result, client identifier often had two different meanings when comparing public and confidential clients. A confidential client tended to be deployed in limited locations on server platforms and thus a client identifier was associated with a client instance. Public clients on the other hand are deployed in many locations. In these cases, the client identifier has closer association to the software being used rather than the instance.

In this specification, "client identifier" is re-scoped more narrowly to imply an instance of client software (or a cluster). A new term, "software identifier" refers to a packaging of a software that is deployed in one or more location. Thus for every software identifier, there may be one or more client identifiers.

This specification defines the following additional terms:

Client Registration Endpoint A SCIM endpoint through which a client can be registered. The means by which the URL for this endpoint are obtained (discovery) are out of scope for this specification.

Client Configuration Endpoint An OAuth 2.0 protected SCIM Resource endpoint through which registration information for a registered client can be managed. This URL for this endpoint is returned by the authorization server in the client registration response or may be discovered by performing a SCIM GET query.

Deployment Organization An administrative security domain under which, a software API is deployed and protected by an OAuth 2.0 framework. In simple cloud deployments, the software API publisher and the deployment organization may be the same. In other scenarios, a Software Publisher may be working with many different deployment organizations.

Software API Deployment A deployment instance of a software API that is protected by OAuth 2.0 in a particular deployment organization domain. For any particular software API, there may be one or more deployments. A software API deployment typically has an associated OAuth 2.0 authorization server endpoint as well as a client registration endpoint. The means by which endpoints are obtained (discovery) are out of scope for this specification.

Software API Publisher The organization that defines a particular web accessible API that may deployed in one or more deployment environments. A publisher may be any commercial, public, private, or open source organization that is responsible for publishing and distributing software that may be protected via OAuth 2.0. A software API publisher may issue software assertions which client developers use to distribute with their software to facilitate registration.

Client Developer The person or organization that builds a client software package and prepares it for distribution. A client developer may obtain a software assertion from a software publisher for the purposes of facilitating client registration.

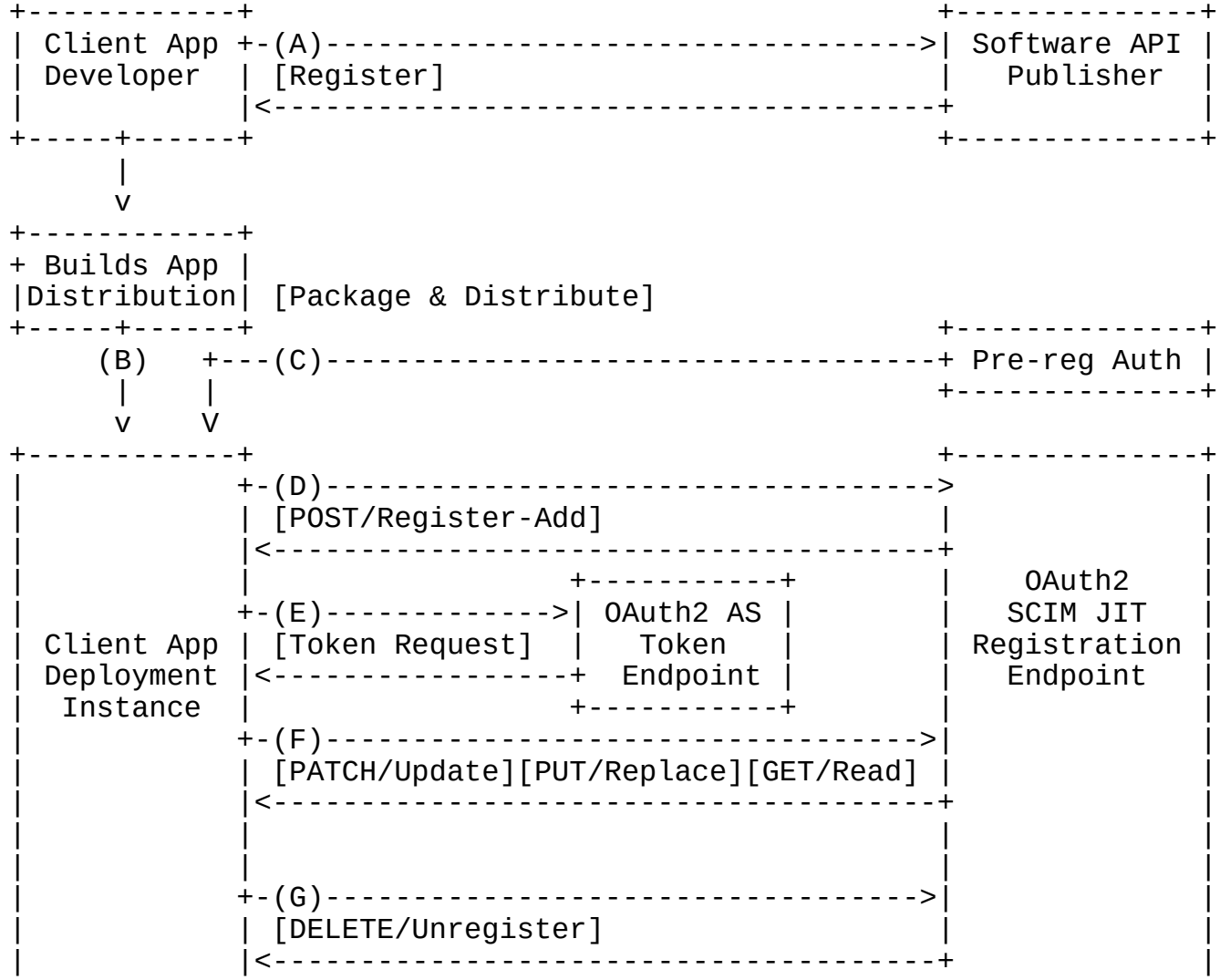
Software Assertion A signed OAuth 2.0 Bearer Token [OAuth.JWT] issued by an software API publisher that asserts information about the client software (see Section 4) that may be used by registration system to qualify clients for eligibility to register. Typically a client developer registers with a software API publisher to obtain a software assertion that will be distributed with all copies of a client application and may be used during the registration process to identify the client to the client registration endpoint.

Initial Access Token An OAuth 2.0 access token is typically issued by a software API deployment’s security domain and used by the client at the registration endpoint in order to register a client. In an authenticated registration, the token is usually issued by the same security domain as the Service API the client is registering for. The content, structure, generation, and validation of this token are out of scope for this specification. The SCIM registration endpoint security policy can use this token to verify that the presenter is allowed to dynamically register a

client. This token may be shared between multiple instances of a client to allow each client to register separately, thereby letting the authorization server use this token to tie multiple instances of registered clients (each with their own distinct client identifier) back to the party to whom the initial access token was issued.

Registration Access Token A registration access token is an optional token a registration server may issue for the purpose of supporting server initiated rotation of client credentials. If client credentials are revoked or expired, the registered client may use a provided registration access token to refresh its registration and obtain new client credentials. When doing this, the client does not need to obtain an access token.

1.3. Typical Registration Flow



+------------+ +--------------+

Figure 1: Figure 1: Registration Lifecycle Flow

The abstract OAuth 2.0 SCIM JIT Client Registration flow illustrated in Figure 1 describes the interaction between a software API publisher, a client developer, a deployed client software instance and the software API deployment registration services in this specification. This figure does not demonstrate error conditions. This flow includes the following steps:

(A) Optionally, a client developer registers the clent application with a software API publisher. The software publisher, upon approval, generates a signed software assertion that is returned to the developer. While the software assertion is defined by this specification, the process of issuance is out-of-scope and is likely a web workflow the developer follows with the publisher.

(B) The client developer packages the client software with the signed software assertion and distributes the client application. The method for doing this is out-of-scope of this specification.

(C) Upon receiving a client application software distribution, a deploying organization may assign an initial access token, that can be used to authenticate a particular distribution of client software for use within the deploying organization’s administrative domain. The mechanism for obtaining an initial access token and distributing it with a set of client software instances is out of scope of this application. One example might be a token supplied during the installation process of a client software, or it may inserted within a locally re-packaged distribution of the client application.

(D) A client application performs an HTTP POST of a "Client" of a JSON structured resource to the registration endpoint. If the client received an iInitial access token in (C), it includes it in the HTTP Authorization field, otherwise, the registration is treated as an "anonymous" POST. An optional software assertion, received in (A), is included in the SCIM Client resource JSON being posted.

Upon successful registration, the SCIM Server returns a copy of the successful registration, including an assigned "client_id" and client credential (e.g. "client_secret"). In order for the client to access and update its registration in the future, an HTTP "Location" (client configuration endpoint) is returned specifying the location of the created SCIM resource.

(E) As with any OAuth 2.0 protected resource, before a client may access and update its registration, the OAuth 2.0 client MUST obtain an access token. If the client received a "registration access token" in step (D), the client MAY use it. Alternatively, the client obtains an access token using the normal OAuth 2.0 a client credential flow section 4.3 [RFC6749] to obtain an access token with scope "urn:oauth:scim:api:scope:registration".

(F) The Client or Developer optionally calls the client configuration endpoint with a SCIM GET, PUT or PATCH request using the access token obtained in (E). Upon reading and or updating client registration data, a SCIM registration endpoint may choose to rotate the client credential.

(G) The Client or Developer optionally calls the client configuration endpoint with a Delete request using the access token issued in (E).

Clients that need to register for more than one service API should typically make a separate registration request for each API being registered.

2. SCIM OAuth Client Resource Schema

This specification defines a new SCIM resource type known as a "Client" identified using the URI "urn:scim:schemas:oauth:Client:1.0". [[Note include in IANA considerations]]

For each attribute defined, a qualifier (OPTIONAL, RECOMMENDED, REQUIRED) is included that indicates the usage requirement for the client. If the word "READ-ONLY" is used, it shall mean the attribute SHOULD be server generated. String attributes and mult-valued attributes are based on the attribute types defined in Section 3.1 and 3.2 [I-D.ietf-scim-core-schema]. Unless otherwise stated, ALL client schema attributes are String based values. For example, URIs, email addresses, identifiers, are all defined as SCIM String Attributes.

Extensions and profiles of this specification MAY expand this list. Authorization servers MUST accept all fields in this schema. The authorization server MUST ignore any additional parameters sent by the client that it does not understand. Clients MAY ignore additional parameters returned by the Server that the client does not understand.

2.1. Singular Attributes

The following is a list of attributes that MUST have only a SINGLE value.

client_id OPTIONAL. An identifier whose value SHOULD be unique for EACH instance registered to a registration server. The value SHOULD be assigned by the registration endpoint on successful completion of the registration. While many clients may share a single software_id, each client instance SHOULD have a unique client_id. If a client registration expires or is revoked, the client MAY provide its previous registration value for continuity purposes (see Section 3.5).

software_assertion
OPTIONAL. A value containing a signed sofware assertion (see Section 4) from a software API publisher. Values in the assertion MAY be used as DEFAULT values for other client registration metadata.

software_id
RECOMMENDED. A unique identifier that identifies the software such as a UUID. The identifier SHOULD NOT change when software version changes or when a new installation instance is detected. "software_id" is intended to help a registration endpoint recognize a client’s assertion that it is a prticular piece of software. Because of this, software identifier is usually associated with a particular client name. While "client_id"is linked to a client software deployment instance, the "software_id" is an identifier shared between all copies of the client software. Registration servers MAY use the supplied software identifier to determine whether a particular client software is approved or supported for use in the deployment domain.

software_version
RECOMMENDED. A version identifier such as a UUID or a number. Servers MAY use equality match to determine if a particular client is a particular version. "software_version" SHOULD change on any update to the client software. Registration servers MAY use the software version and identity to determine whether a particular client version is authorized for use in the deployment domain.

client_name
RECOMMENDED. A human-readable name of the client to be presented to the user. If omitted, the authorization server MAY display the raw "client_id" value to the user instead. It is RECOMMENDED

that clients always send this field. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.5).

client_secret READ-ONLY. A value that is the OAuth 2.0 "client_secret". This value SHOULD be assigned by the server. Any time the value is returned to the registered client, the server MAY rotate the value. Clients MUST check for any change in value returned and update their copy. The value is typically only used when the attribute "token_endpoint_auth_method" is set to "client_post" or "client_basic".

client_uri
RECOMMENDED. A URL of the homepage of the client software. If present, the server SHOULD display this URL to the end user in a clickable fashion. It is RECOMMENDED that clients always send this field. The value of this field MUST point to a valid Web page. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.5).

jwks_uri
OPTIONAL. A URL for the client's JSON Web Key Set [JWK] document representing the client's public keys. The value of this field MUST point to a valid JWK Set. These keys MAY also be used for higher level protocols that require signing or encryption.

logo_uri
OPTIONAL. A URL that references a logo image for the client. If present, the server SHOULD display this image to the end user during approval. The value of this field MUST point to a valid image file. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.5).

policy_uri
OPTIONAL. A URL that points to a human-readable policy document for the client. The authorization server SHOULD display this URL to the End-User if it is given. The Policy usually describes how an End-User's data will be used by the client. The value of this field MUST point to a valid Web page. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.5).

registration_token
READ-ONLY. An OAuth 2.0 Bearer Token [OAuth.JWT] known as the "registration access token". The token MAY be used by the client as a long-term access token used to update and manage the client's registration. The token is intended to allow the client to refresh its registration and obtain new client credentials in

the event of expiry or revocation of the client credentials by the service API deployment domain. When client credentials are rotated, the registration access token SHOULD also be rotated.

scope
: OPTIONAL. A space separated list of scope values (as described in Section 3.3 [RFC6749]) that the client can use when requesting access tokens. The semantics of values in this list is service specific. If omitted, an authorization server MAY register a client with a default set of scopes.

targetEndpoint
: RECOMMENDED. A URI of the service API the client is registering for. The server MAY provide this value in its response. Clients requesting access to more than one target endpoint should register once for each target.

token_endpoint_auth_method
: OPTIONAL. Value containing the requested authentication method for the Token Endpoint. The server MAY override the requested value. Clients MUST check for a change in value in the registration response. Values defined by this specification are:

  * "none": The client is a public client as defined in OAuth 2.0 and does not have a client secret.
  * "client_secret_post": The client uses the HTTP POST parameters defined in OAuth 2.0 section 2.3.1.
  * "client_secret_basic": the client uses HTTP Basic defined in OAuth 2.0 section 2.3.1

  Additional values can be defined via the IANA OAuth Token Endpoint Authentication Methods registry Section 6.1. Absolute URIs can also be used as values for this parameter. If unspecified or omitted, the default is "client_secret_basic", denoting HTTP Basic Authentication Scheme as specified in Section 2.3.1 of OAuth 2.0.

tos_uri
: OPTIONAL. A URL that points to a human-readable "Terms of Service" document for the client. The authorization server SHOULD display this URL to the End-User if it is given. The Terms of Service usually describe a contractual relationship between the End-User and the client that the End-User accepts when authorizing the client. The value of this field MUST point to a valid Web page. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.5).

2.2. Multi-valued Attributes

The following is a list of SCIM multi-valued attributes that MAY be part of a "Client" resource.

contacts
: OPTIONAL. One or more email addresses for people responsible for this client. The authorization server MAY make these addresses available to end users for support requests for the client. An authorization server MAY use these email addresses as identifiers for an administrative page for this client.

redirect_uris
: RECOMMENDED. One or more redirect URI values for use in redirect-based flows such as the Authorization Code and Implicit grant types. authorization servers SHOULD require registration of valid redirect URIs for all clients that use these grant types to protect against token and credential theft attacks.

grant_types
: OPTIONAL. One or more OAuth 2.0 grant types that the client may use. These grant types are defined as follows:

  * "authorization_code": The Authorization Code Grant described in OAuth 2.0 Section 4.1
  * "implicit": The Implicit Grant described in OAuth 2.0 Section 4.2
  * "password": The Resource Owner Password Credentials Grant described in OAuth 2.0 Section 4.3
  * "client_credentials": The "Client credentials Grant" described in OAuth 2.0 Section 4.4
  * "refresh_token": The Refresh Token Grant described in OAuth 2.0 Section 6.
  * "urn:ietf:params:oauth:grant-type:jwt-bearer": The JWT Bearer grant type defined in OAuth JWT Bearer Token Profiles [OAuth.JWT].
  * "urn:ietf:params:oauth:grant-type:saml2-bearer": The SAML 2 Bearer grant type defined in OAuth SAML 2 Bearer Token Profiles [OAuth.SAML2].

  Authorization servers MAY allow for other values as defined in grant type extensions to OAuth 2.0. The extension process is described in OAuth 2.0 Section 2.5, and the value of this parameter MUST be the same as the value of the "grant_type" parameter passed to the Token Endpoint defined in the extension.

response_types

OPTIONAL. One or more OAuth 2.0 response types that the client may use. These response types are defined as follows:

* "code": The Authorization Code response described in OAuth 2.0 Section 4.1.
* "token": The Implicit response described in OAuth 2.0 Section 4.2.

Authorization servers MAY allow for other values as defined in response type extensions to OAuth 2.0. The extension process is described in OAuth 2.0 Section 2.5, and the value of this parameter MUST be the same as the value of the "response_type" parameter passed to the Authorization Endpoint defined in the extension.

2.3. Client Representation

The following is a non-normative example of a fully populated SCIM OAuth 2.0 client registration in JSON format.

```
{
  "schemas":["urn:scim:schemas:core:1.0",
             "urn:scim:schemas:oauth:2.0:Client"],
  "id":"2060107e82-fbe3-42bd-b199-15df7081a8ae",
  "resourceType":"Client",
  "software_id":"5ed2dd14-3ef7-4655-a41d-b5bd4c5266cc",
  "software_version":"5.1.2.3.4",
  "client_name":"Example Social Client",
  "logo_uri":"https://client.example.org/logo.png",
  "jwks_uri":"https://client.example.org/my_public_keys.jwks",
  "token_endpoint_auth_method":"client_secret_post",
  "scope":"read write dolphin",
  "client_id":"2060107e82-fbe3-42bd-b199-15df7081a8ae",
  "client_secret":"Z7tk2XqLKo1CfE14374teR4V554e8JUS",
  "redirect_urls":[""https://client.example.org/callback",
    "https://client.example.org/callback2"],
  "targetEndpoint":"https://social.example.com/base"
}
```

Figure 2: Example Client Resource

2.4. Relationship Between Grant Types and Response Types

The "grant_types" and "response_types" values described above are partially orthogonal, as they refer to arguments passed to different endpoints in the OAuth protocol. However, they are related in that the "grant_types" available to a client influence the "response_types" that the client is allowed to use, and vice versa.

For instance, a "grant_types" value that includes "authorization_code" implies a "response_types" value that includes code, as both values are defined as part of the OAuth 2.0 Authorization Code Grant. As such, a server supporting these fields SHOULD take steps to ensure that a client cannot register itself into an inconsistent state.

The correlation between the two fields is listed in the table below.

grant_types value includes:	response_types value includes:
authorization_code	code
implicit	token
password	(none)
client_credentials	(none)
refresh_token	(none)
urn:ietf:params:oauth:grant-type:jwt-bearer	(none)
urn:ietf:params:oauth:grant-type:saml2-bearer	(none)

Extensions and profiles of this document that introduce new values to either the "grant_types" or "response_types" parameter MUST document all correspondences between these two parameter types.

2.5. Human Readable Client Metadata

[[This needs to be updated to be compatible with SCIM. There is a also a problem with how to limit the amount of localization data exchange for an instance registration. Note that mobile clients tend to only need one preferred language while web clients represent many clients and may have more than 20 languages to support.]]

Human-readable Client Metadata values and client Metadata values that reference human-readable values MAY be represented in multiple languages and scripts. For example, the values of fields such as "client_name", "tos_uri", "policy_uri", "logo_uri", and "client_uri" might have multiple locale-specific values in some client registrations.

To specify the languages and scripts, BCP47 [RFC5646] language tags are added to client Metadata member names, delimited by a # character. Since JSON member names are case sensitive, it is RECOMMENDED that language tag values used in Claim Names be spelled using the character case with which they are registered in the IANA Language Subtag Registry [IANA.Language]. In particular, normally language names are spelled with lowercase characters, region names are spelled with uppercase characters, and languages are spelled with mixed case characters. However, since BCP47 language tag values are case insensitive, implementations SHOULD interpret the language tag values supplied in a case insensitive manner. Per the recommendations in BCP47, language tag values used in Metadata member names should only be as specific as necessary. For instance, using "fr" might be sufficient in many contexts, rather than "fr-CA" or "fr-FR".

For example, a client could represent its name in English as ""client_name#en": "My Client"" and its name in Japanese as ""client_name#ja-Jpan-JP": "\u30AF\u30E9\u30A4\u30A2\u30F3\u30C8\u540D"" within the same registration request. The authorization server MAY display any or all of these names to the Resource Owner during the authorization step, choosing which name to display based on system configuration, user preferences or other factors.

If any human-readable field is sent without a language tag, parties using it MUST NOT make any assumptions about the language, character set, or script of the string value, and the string value MUST be used as-is wherever it is presented in a user interface. To facilitate interoperability, it is RECOMMENDED that clients and servers use a human-readable field without any language tags in addition to any language-specific fields, and it is RECOMMENDED that any human-readable fields sent without language tags contain values suitable for display on a wide variety of systems.

Implementer’s Note: Many JSON libraries make it possible to reference members of a JSON object as members of an Object construct in the native programming environment of the library. However, while the "#" character is a valid character inside of a JSON object’s member names, it is not a valid character for use in an object member name in many programming environments. Therefore, implementations will need to use alternative access forms for these claims. For instance, in JavaScript, if one parses the JSON as follows, "var j = JSON.parse(json);", then the member "client_name#en-us" can be accessed using the JavaScript syntax "j["client_name#en-us"]".

2.6. Registration Server Processing Rules

A registration server MAY override any value that a client requests during the registration process (including any omitted values) and replace the requested value with a server generated value or default at the server's discretion.

If a software assertion is provided the registration server SHOULD validate the assertion as per Section 4.

3. SCIM Interaction Profile

All calls to the registration endpoint follow the HTTP verbs defined in [I-D.ietf-scim-api].

All resources accessible through the SCIM client registration endpoint are OAuth 2.0 protected. Clients that require access to their own registration resource, MAY use the registration access token ("registration_token") returned after registration, OR obtain an access token from the token server endpoint using the client credentials flow Section 4.4 [RFC6749], and scope "urn:oauth:scim:api:scope:registration".

The initial access token used in the initial registration SHOULD NOT be used for this purpose of managing or updating client resources. Access to the client registration base for the purpose of adding a new registration MAY permit anonymous access as desribed in the next section.

Clients SHOULD NOT be able to access the registration resources of other clients.

Clients with expired client credentials SHOULD follow the procedures in Section 3.5.

3.1. Adding A Registration

Adding a registration follows the normal SCIM method for creating a resource (an HTTP POST) as described in Section 3.1 [I-D.ietf-scim-api].

3.1.1. Anonymous Registration

The SCIM registration endpoint MAY support anonymous registration for those clients that have not been issued an initial access token. For those clients that have been issued a software assertion, the clients SHOULD include this assertion in the software_assertion field of the client resource being created. The Server SHOULD validate the assertion and validate the issuer, subject, audience, and expiry fields as described in Section 4.1.

For a non-normative example, see Figure 2. In the anonymous case, the "HTTP Authorization" header is omitted.

3.1.2. Pre-Authorized Registration

A pre-authorized registration is where the client or the installer of the client has been issued an initial access token which may be used to call the SCIM registration endpoint. Clients should use the token by inserting the appropriate value in the HTTP Authorization header. Additionally, as with anonymous registrations, clients MAY include a software assertion. For example, a non-normative registration (line breaks inserted for readability):

```
POST /Users  HTTP/1.1
Host: example.com
Accept: application/json
Content-Type: application/json
Authorization: Bearer h480djs93hd8
Content-Length: ...

{
  "schemas":["urn:scim:schemas:core:1.0",
             "urn:scim:schemas:oauth:2.0:Client"],
  "software_id":"5ed2dd14-3ef7-4655-a41d-b5bd4c5266cc",
  "software_assertion":"eyJhbGciOiJSUzI1NiJ9.",
  "software_version":"5.1.2.3.4",
  "client_name":"Example Social Client",
  "logo_uri":"https://client.example.org/logo.png",
  "jwks_uri":"https://client.example.org/my_public_keys.jwks",
  "token_endpoint_auth_method":"client_secret_post",
  "scope":"read write dolphin",
  "redirect_urls":[""https://client.example.org/callback",
    "https://client.example.org/callback2"],
  "targetEndpoint":"https://social.example.com/base"
}
```

Figure 2: Figure 3: Client Registration Request

On successful processing, the SCIM endpoint would respond with:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location:
  https://example.com/v2/Clients/2060107e82-42bd-b199-15df7081a8ae
ETag: W/"e180ee84f0671b1"

{
  "schemas":["urn:scim:schemas:core:1.0",
```

```
            "urn:scim:schemas:oauth:2.0:Client"],
      "resourceType":"Client",
      "id":"2060107e82-42bd-b199-15df7081a8ae",
      "software_id":"5ed2dd14-3ef7-4655-a41d-b5bd4c5266cc",
      "software_assertion":"eyJhbGciOiJSUzI1NiJ9.",
      "software_version":"5.1.2.3.4",
      "client_name":"Example Social Client",
      "logo_uri":"https://client.example.org/logo.png",
      "jwks_uri":"https://client.example.org/my_public_keys.jwks",
      "token_endpoint_auth_method":"client_secret_post",
      "scope":"read write dolphin",
      "client_id":"2060107e82-fbe3-42bd-b199-15df7081a8ae",
      "client_secret":"Z7tk2XqLKo1CfE14374teR4V554e8JUS",
      "redirect_urls":[""https://client.example.org/callback",
        "https://client.example.org/callback2"],
      "targetEndpoint":"https://social.example.com/base"
    }
```

Figure 4: Client Registration Response

Clients SHOULD read the response and review and retain the following items:

- The client SHOULD remember the "HTTP Location" returned. This location is used for future registration updates and client credential rotation.
- The client MUST read the assigned "client_id" value and retain for use in all authorization server interactions with the associated target endpoint.
- If returned, the client SHOULD retain the "registration_token" as a credential that can be used to access and update the client's registration.
- The client MUST read the "token_endpoint_auth_method" to obtain the authentication method the server has chosen for the client.
- Based on the returned token endpoint authentication method returned, the client MUST read the registration for the appropriate client credential such as "token_secret".

3.2. Reading A Registration

When a server receives a GET request from an HTTP client whose access token is assigned to the OAuth 2.0 client that is the subject of the registration, the server MAY elect to rotate the client credential (e.g. client_secret) or other relevant attributes. If the access request is not from the registered client (such as an administrator), the server SHOULD NOT change any registration values. [[should the server mask anything?]]

Reading a client resource is done using the SCIM HTTP GET verb as defined in Section 3.2 [I-D.ietf-scim-api]

A client resource may be retrieved using the "location" returned from the original client registration. The registration MAY be retrieved directly using this URL. If the location is not available, the current values of the registration can be obtained by querying this URL, or by searching the "/Clients" with a query parameter for client_id that corresponds to the client's "client_id".

```
GET /Clients/?
  filter=client_id%20eq%2060107e82-fbe3-42bd-b199-15df7081a8ae

Host: example.com
Accept: application/json
Authorization: Bearer deadbeef
```

Figure 3: Figure 5: Retrieving a Registration

If found, the server will respond with a HTTP 200 message containing a single client resource, with one or more attributes:

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

{
  "totalResults":1,
  "schemas":["urn:scim:schemas:core:1.0",
             "urn:scim:schemas:oauth:Client:1.0"],
  "Resources":[
     {
      "id":"2060107e82-fbe3-42bd-b199-15df7081a8ae",
      "resourceType":"Client",
      "software_id":"5ed2dd14-3ef7-4655-a41d-b5bd4c5266cc",
      "software_version":"5.1.2.3.4",
      "client_name":"Example Social Client",
...
      "token_endpoint_auth_method":"client_secret_post",
      "client_id":"2060107e82-fbe3-42bd-b199-15df7081a8ae",
      "client_secret":"Z7tk2XqLKo1CfE14374teR4V554e8JUS"
     }
  ]
}
```

Figure 6: Retrieving Registration Response

If no resource is found, the server will respond with a HTTP 200 message containing zero result resources.

When a client returns its own record, clients should check for updates to the following items:

- The client MUST read the assigned "client_id" value and retain for use in all authorization server interactions with the associated target endpoint.
- If returned, the client SHOULD update the "registration_token" as a credential that can be used to access and update the client's registration.
- The client MUST read the "token_endpoint_auth_method" to obtain the authentication method the server has chosen for the client.
- Based on the returned token endpoint authentication method returned, the client MUST read the registration for any changes to client credential such as "token_secret".

3.3. Modifying A Registration

Modification of a client resource can be done using the SCIM PUT or PATCH verbs as defined in Section 3.3 [I-D.ietf-scim-api].

When a server receives a PUT or PATCH request from an HTTP client whose access token is assigned to the OAuth 2.0 client that is the subject of the registration, the server MAY elect to rotate the client credential (e.g. client_secret) or other relevant attributes.

3.4. Deleting A Registration

De-registration of a client resource can be done using the SCIM DELETE verbs as defined in Section 3.4 [I-D.ietf-scim-api]. When a delete is completed, all associated client credentials and access tokens MUST be revoked or invalidated.

3.5. Expired Registration

An expired registration occurs when the client's credential has expired, or has been revoked by the registration service. On expiry, access tokens issued to the client MUST be revoked or invalidated.

If, on registration, the client was returned a registration access token ("registration_token"), the client MAY obtain new credentials by simply retrieving its client profile using the registration token and the SCIM GET verb at the location returned after the initial registration (see Figure 3).

Clients that do not have a valid registration access token MAY re-register as a new client instance. In such case, the client MAY include it's existing "client_id"in its client meta data as "client_id"for tracking purposes only. In such cases, the old

client_id SHOULD NOT be re-activated and the server SHOULD issue a new client_id value.

4. Software Assertion Token

A software assertion is an 'authorization' bearer JSON Web Token as defined in Section 2.1 [OAuth.JWT]. While the software assertion is a JWT authorization token, it SHOULD not be used with the authorization server token endpoint. It is used by the client registration endpoint for the purpose of registration. A software assertion is a statement of claims about the client software being registered and SHOULD NOT be used as an authentication of the software.

Software assertions may be generated by a software API publisher for the purpose of allowing a developer to incorporate a signed assertion that can be used to register client software at more than one Software API deployment.

4.1. Software Assertion Requirements

In order to create and validate a software assertion, the following requirements apply in addition to those stated in Section 3 [OAuth.JWT].

1. The JWT MAY contain any claim specified in Section 2.
2. The JWT MUST contain an "iss" (issuer) claim that contains a unique identifier for the entity that issued the JWT. This value SHOULD correspond to the software API publisher.
3. The JWT MUST contain a "sub" (subject) claim that contains a unique value corresponding to the "software_id". This number is MAY be assigned by the software API publisher.
4. The JWT MUST contain an "aud" (audience) claim containing a value that is ONE of the following:

   * A value that identifies one or more software API deployments, where the client software MAY be registered.
   * A value "urn:oauth:scim:reg:generic" which indicates the assertion MAY be used with any software API deployment environment.
5. The JWT MUST contain an "exp" (expiration) claim that limits the time window during which the JWT can be used to register clients. When registering clients, the registration server MUST verify that the expiration time has not passed, subject to allowable clock skew between systems, and reject expired JWTs. The authorization server SHOULD NOT use this value to revoke an existing client registration.

5. Server Schema Configurations

[[This section to be revised pending clarification of schema extensions methodology in draft-ietf-core-schema]]

5.1. Resource Type

The following is a normative JSON representation of a SCIM "ResourceType" representing a client resorce in a SCIM server returned by querying "GET [scimendpoint}/ResourceTypes". "..." indicates other ResourceTypes removed for clarity.

```
[
 ...
  {
    "name": "Client",
    "endpoint": "/Clients",
    "schema": "urn:scim:schemas:oauth:2.0"
  },
 ...
]
```

Figure 7: Client Resource Type

5.2. Schema Representation

The following is a normative exammple of the SCIM "Schema" for a client in JSON format returned when querying "GET {scimendpoint}/Schemas".

```
{  "id":"urn:scim:schemas:oauth:2.0:Client",
  "name":"Client",
  "description":"OAuth 2 Client",
  "schema":["urn:scim:core:1.0",
            "urn:scim:schemas:oauth:2.0"],
  "endpoint":"/Clients",
  "attributes":[
    {
      "name":"id",
      "type":"string",
      "multiValued":false,
      "description":"Unique identifier for the SCIM resource....",
      "readOnly":true,
      "required":true,
      "caseExact":false
    },
    {
      "name":"client_id",
```

```
        "type":"string",
        "multiValued":false,
        "description":
          "OAuth 2.0 client_id assigned to the registered client.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"software_assertion",
        "type":"string",
        "multiValued":false,
        "description":
          "A signed JWT assertion about the client.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"software_id",
        "type":"string",
        "multiValued":false,
        "description":"Unique identifier for client software.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"software_version",
        "type":"string",
        "multiValued":false,
        "description":"A version identifier for the software.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"client_name",
        "type":"string",
        "multiValued":false,
        "description":"A human readable name of the client.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"client_secret",
        "type":"string",
```

```
        "multiValued":false,
        "description":
          "A client secret assigned by the registration endpoint.",
        "readOnly":true,
        "required":false,
        "caseExact":true
      },
      {
        "name":"client_uri",
        "type":"string",
        "multiValued":false,
        "description":
          "URL of homepage for client(displayable to an end-user).",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"jwks_uri",
        "type":"string",
        "multiValued":false,
        "description":"A URL for the client's JSON Web Key Set.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"logo_uri",
        "type":"string",
        "multiValued":false,
        "description":"A URL that references a logo image for client.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"policy_uri",
        "type":"string",
        "multiValued":false,
        "description":
          "A URL of a human-readable policy document for the client.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"registration_token",
        "type":"string",
```

```
        "multiValued":false,
        "description":
          "A token issued to client for updating its registration.",
        "readOnly":true,
        "required":false,
        "caseExact":true
      },
      {
        "name":"scope",
        "type":"string",
        "multiValued":false,
        "description":"Registered OAuth 2 scopes the client uses.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"targetEndpoint",
        "type":"string",
        "multiValued":false,
        "description":
          "The OAuth2 resource endpoint the client intends to access.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"token_endpoint_auth_method",
        "type":"string",
        "multiValued":false,
        "description":"OAuth 2 Token Endpoint authorization method.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"tos_uri",
        "type":"string",
        "multiValued":false,
        "description":
          "A URL pointing to a human readable terms of service.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"contacts",
        "type":"string",
```

```
        "multiValued":true,
        "description":
          "One or more email addresses of contacts for client.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"redirect_uris",
        "type":"string",
        "multiValued":true,
        "description":"One or more OAuth 2 redirect URI values.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"grant_types",
        "type":"string",
        "multiValued":true,
        "description":
          "One or more OAuth 2 grant types the client may use.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
      {
        "name":"response_types",
        "type":"string",
        "multiValued":true,
        "description":"One or more OAuth 2 response types.",
        "readOnly":false,
        "required":false,
        "caseExact":true
      },
    ]
  }
```

Figure 8: Client Schema Representation

6. IANA Considerations

6.1. OAuth Token Endpoint Authentication Methods Registry

This specification establishes the OAuth Token Endpoint Authentication Methods registry.

Additional values for use as "token_endpoint_auth_method" metadata values are registered with a Specification Required ([RFC5226]) after a two-week review period on the oauth-ext-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests must be sent to the oauth-ext-review@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request to register token_endpoint_auth_method value: example").

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

IANA must only accept registry updates from the Designated Expert(s) and should direct all requests for registration to the review mailing list.

6.1.1. Registration Template

Token Endpoint Authorization Method name:
: The name requested (e.g., "example"). This name is case sensitive. Names that match other registered names in a case insensitive manner SHOULD NOT be accepted.

Change controller:
: For Standards Track RFCs, state "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification document(s):
: Reference to the document(s) that specify the token endpoint authorization method, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

6.1.2. Initial Registry Contents

The OAuth Token Endpoint Authentication Methods registry’s initial contents are:

- Token Endpoint Authorization Method name: "none"
- Change controller: IETF

- Specification document(s): [[this document]]

- Token Endpoint Authorization Method name: "client_secret_post"
- Change controller: IETF
- Specification document(s): [[this document]]

- Token Endpoint Authorization Method name: "client_secret_basic"
- Change controller: IETF
- Specification document(s): [[this document]]

7. Security Considerations

Since requests to the client registration endpoint result in the transmission of clear-text credentials (in the HTTP request and response), the server MUST require the use of a transport-layer security mechanism when sending requests to the registration endpoint. The server MUST support TLS 1.2 RFC 5246 [RFC5246] and/or TLS 1.0 [RFC2246] and MAY support additional transport-layer mechanisms meeting its security requirements. When using TLS, the client MUST perform a TLS/SSL server certificate check, per RFC 6125 [RFC6125].

Since the SCIM client configuration endpoint is an OAuth 2.0 protected resource, it SHOULD have some rate limiting on failures to prevent initial access tokens from being disclosed though repeated access attempts.

For clients that use redirect-based grant types such as Authorization Code and Implicit, authorization servers SHOULD require clients to register their "redirect_uris". Requiring clients to do so can help mitigate attacks where rogue actors inject and impersonate a validly registered client and intercept its authorization code or tokens through an invalid redirect URI.

The authorization server MUST treat all client metadata, including software assertions, as self-asserted. A rogue client might use the name and logo for the legitimate client, which it is trying to impersonate. An authorization server needs to take steps to mitigate this phishing risk, since the logo could confuse users into thinking they’re logging in to the legitimate client. For instance, an authorization server could warn if the domain/site of the logo doesn’t match the domain/site of redirect URIs. An authorization server can also present warning messages to end users about untrusted clients in all cases, especially if such clients have been dynamically registered and have not been trusted by any users at the authorization server before.

Authorization servers MAY assume that registered client software sharing the same software assertion, software_id, and other metadata SHOULD have similar operational behaviour metrics. Similarly, Authorization server administrators MAY use software_id and software_version to facilitate normal change control and approval management of client software including:

- Approval of specific clients software for use with specific protected resources.
- Lifecycle management and support of specific software versions as indicated by software_version.
- Revocation of groups of client credentials and associated access tokens when support issues or security risks identified with a particular client software as identified by software_id and software_version.

In a situation where the authorization server is supporting open client registration, it must be extremely careful with any URL provided by the client that will be displayed to the user (e.g. "logo_uri", "tos_uri", "client_uri", and "policy_uri"). For instance, a rogue client could specify a registration request with a reference to a drive-by download in the "policy_uri". The authorization server SHOULD check to see if the "logo_uri", "tos_uri", "client_uri", and "policy_uri" have the same host and scheme as the those defined in the array of "redirect_uris" and that all of these resolve to valid Web pages.

Access tokens issued to clients to facilitate update or retrieval of client registrations SHOULD be short lived.

Clients SHOULD rotate their client credentials before they expire by obtaining an access token from the authorization server using the registration scope. If a client has not successfully rotated its credential prior to expiry, the client MUST register as a new client.

If a client is deprovisioned from a server (due to expiry or de-registration), any outstanding Registration Access Token for that client MUST be invalidated at the same time. Otherwise, this can lead to an inconsistent state wherein a client could make requests to the client configuration endpoint where the authentication would succeed but the action would fail because the client is no longer valid.

Clients that are unable to retain a client credential for the life of the client instance MAY NOT register and should continue to be treated as Public clients as defined by OAuth 2.0.

8. Normative References

[I-D.ietf-scim-api]
Drake, T., Mortimore, C., Ansari, M., Grizzle, K., and E. Wahlstroem, "System for Cross-Domain Identity Management:Protocol", draft-ietf-scim-api-00 (work in progress), August 2012.

[I-D.ietf-scim-core-schema]
Mortimore, C., Harding, P., Madsen, P., and T. Drake, "System for Cross-Domain Identity Management: Core Schema", draft-ietf-scim-core-schema-00 (work in progress), August 2012.

[IANA.Language]
Internet Assigned Numbers Authority (IANA), "Language Subtag Registry", 2005.

[JWK] Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key (work in progress), May 2013.

[OAuth.JWT]
Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Bearer Token Profiles for OAuth 2.0", draft-ietf-oauth-jwt-bearer (work in progress), March 2013.

[OAuth.SAML2]
Campbell, B., Mortimore, C., and M. Jones, "SAML 2.0 Bearer Assertion Profiles for OAuth 2.0", draft-ietf-oauth-saml2-bearer (work in progress), March 2013.

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

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.

Appendix A. Acknowledgments

This draft was based upon in large part upon the work in draft-ietf-oauth-dyn-reg-12. The author would like to thank Justin Richter and the members of the OAuth Working Group.

In turn, the authors thank the OAuth Working Group, the User-Managed Access Working Group, and the OpenID Connect Working Group participants for their input to this document. In particular, the following individuals have been instrumental in their review and contribution to various versions of this document: Amanda Anganes, Tim Bray, Domenico Catalano, Donald Coffin, George Fletcher, Thomas Hardjono, Torsten Lodderstedt, Eve Maler, Josh Mandel, Nov Matake, Nat Sakimura, Christian Scholz, and Hannes Tschofenig.

Appendix B. Document History

[[to be removed by the RFC editor before publication as an RFC]]

-00

o First draft based on work from draft-ietf-oauth-dyn-reg-12.

Authors' Addresses

Phil Hunt (editor)
Oracle Corporation

Email: phil.hunt@yahoo.com

Morteza Ansari
Cisco

Email: moransar@cisco.com

Tony Nadalin
Microsoft

Email: tonynad@microsoft.com