OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 28, 2014

J. Richer, Ed.
The MITRE Corporation
J. Bradley
Ping Identity
M. Jones
Microsoft
M. Machulak
Newcastle University
August 27, 2013

OAuth 2.0 Dynamic Client Registration Management Protocol
draft-richer-oauth-dyn-reg-management-00

Abstract

This specification defines methods for a dynamically registered OAuth 2.0 client to manage its registration through an OAuth 2.0 protected web API as well as extended client metadata attributes.

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 February 28, 2014.

Copyright Notice

Table of Contents

1. Introduction

In some use-case scenarios, it is desirable or necessary to allow OAuth 2.0 clients to obtain authorization from an OAuth 2.0 authorization server without requiring the two parties to interact beforehand. Nevertheless, for the authorization server to accurately and securely represent to end-users which client is seeking authorization to access the end-user's resources, a method for automatic and unique registration of clients is needed. 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. Historically, this has happened out-of-band from the OAuth 2.0 protocol.

This specification extends the OAuth 2.0 Core Dynamic Client Registration [DynReg] specification (which provides a method for OAuth 2.0 clients to be registered dynamically with an authorization server) and defines a mechanism for the client to present the authorization server with a set of extended metadata, such as a

display name and icon to be presented to the user during the authorization step. This draft also provides a mechanism for the client to read and update this information after the initial registration action. This draft protects these actions through the use of an OAuth 2.0 bearer access token that is issued to the client during registration explicitly for this purpose.

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].

This specification defines the following additional terms:

Client Configuration Endpoint OAuth 2.0 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 information response.

Registration Access Token OAuth 2.0 bearer token issued by the authorization server through the client registration endpoint that is used to authenticate the caller when accessing the client's registration information at the client configuration endpoint. This access token is associated with a particular registered

client.

1.3. Protocol Flow

This extends the flow in the core dynamic registration [DynReg] specification as follows:

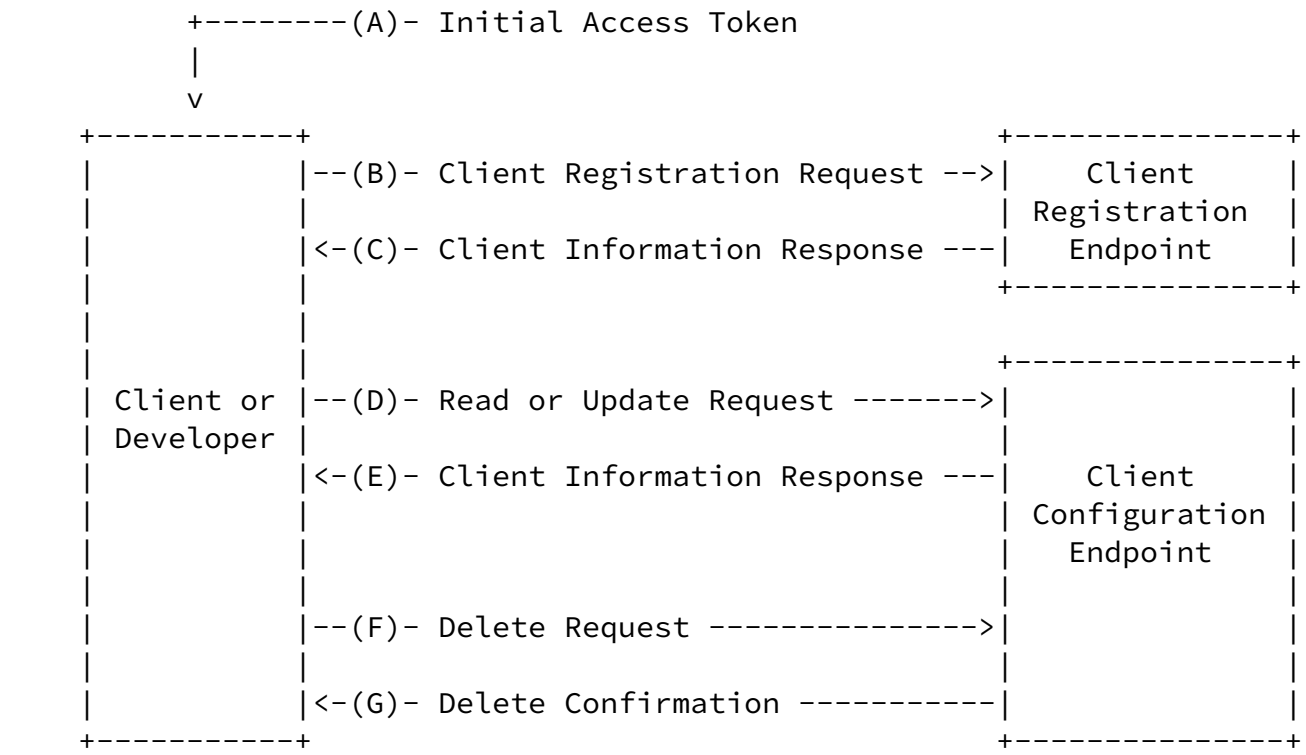

Figure 1: Abstract Protocol Flow

The abstract OAuth 2.0 Client dynamic registration flow illustrated in Figure 1 describes the interaction between the client or developer and the two endpoints defined in this specification and its parent.

This figure does not demonstrate error conditions. This flow includes the following steps:

(A)
Optionally, the client or developer is issued an initial access token for use with the client registration endpoint. The method by which the initial access token is issued to the client or developer is out of scope for this specification.

(B)
The client or developer calls the client registration endpoint with its desired registration metadata, optionally including the initial access token from (A) if one is required by the authorization server.

(C)
The authorization server registers the client and returns the client's registered metadata, a client identifier that is unique at the server, a set of client credentials such as a client secret

if applicable for this client, a URI pointing to the client configuration endpoint, and a registration access token to be used when calling the client configuration endpoint.

(D)
The client or developer optionally calls the client configuration endpoint with a read or update request using the registration access token issued in (C). An update request contains all of the client's registered metadata.

(E)
The authorization server responds with the client's current configuration, potentially including a new registration access token and a new set of client credentials such as a client secret if applicable for this client. If a new registration access token is issued, it replaces the token issued in (C) for all subsequent calls to the client configuration endpoint.

(F)
The client or developer optionally calls the client configuration endpoint with a delete request using the registration access token issued in (C).

(G)
The authorization server deprovisions the client and responds with a confirmation that the deletion has taken place.

1.4. Registration Tokens and Client Credentials

Throughout the course of the dynamic registration protocol, there are three different classes of credentials in play, each with different properties and targets.

- The initial access token is optionally used by the client or developer at the registration endpoint. This is an OAuth 2.0 token that is used to authorize the initial client registration request. The content, structure, generation, and validation of this token are out of scope for this specification. The authorization server can use this token to verify that the presenter is allowed to dynamically register new clients. This token may be shared between multiple instances of a client to allow them to each 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, usually an application developer. This token should be used only at the client registration endpoint.

- The registration access token is used by the client or developer at the client configuration endpoint and represents the holder's authorization to manage the registration of a client. This is an OAuth 2.0 bearer token that is issued from the client registration endpoint in response to a client registration request and is returned in a client information response. The registration access token is uniquely bound to the client identifier and is required to be presented with all calls to the client configuration endpoint. The registration access token should be protected and should not be shared between instances of a client (otherwise, one instance could change or delete registration values for all instances of the client). The registration access token can be rotated through the use of the client read and update methods on the client configuration endpoint. The registration access token should be used only at the client configuration

endpoint.

- The client credentials (such as "client_secret") are optional depending on the type of client and are used to retrieve OAuth tokens. Client credentials are most often bound to particular instances of a client and should not be shared between instances. Note that since not all types of clients have client credentials, they cannot be used to manage client registrations at the client configuration endpoint. The client credentials can be rotated through the use of the client read and update methods on the client configuration endpoint. The client credentials can not be used for authentication at the client registration endpoint or at the client configuration endpoint.

1.4.1. Credential Rotation

The Authorization Server MAY rotate the client's registration access token and/or client credentials (such as a "client_secret") throughout the lifetime of the client. The client can discovery that these values have changed by reading the client information response returned from either a read or update request to the client configuration endpoint. The client's current registration access token and client credentials (if applicable) MUST be included in this response.

The registration access token SHOULD be rotated only in response to a read or update request to the client configuration endpoint, at which point the new registration access token is returned to the client and the old registration access token SHOULD be discarded by both parties. If the registration access token to expire or be rotated outside of such requests, the client or developer may be locked out of managing the client's configuration.

2. Client Metadata

Many OAuth 2.0 clients wish to register different kinds of client metadata to facilitate authorization and usage of the protected API. This specification extends the list of client metadata defined in OAuth 2.0 Core Client Dynamic Registration [DynReg] with the following fields:

client_name
: 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 2.1).

client_uri
: URL of the homepage of the client. 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 2.1).

logo_uri
: URL that references a logo 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 2.1).

scope
: Space separated list of scope values (as described in OAuth 2.0 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.

contacts
: Array of 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.

tos_uri
: 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 2.1).

policy_uri
: 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 2.1).

jwks_uri
: 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 be used for higher level protocols that require signing or encryption.

software_id
: A identifier for the software that comprises a client. Unlike "client_id", which is issued by the authorization server and generally varies between instances, the "software_id" is asserted by the client software and is intended to be shared between all copies of the client software. The value for this field MAY be a UUID [RFC4122]. The identifier SHOULD NOT change when software version changes or when a new installation instance is detected. Authorization servers MUST treat this field as self-asserted by the client and MUST NOT make any trusted decisions on the value of this field alone.

software_version
: A version identifier for the software that comprises a client. The value of this field is a string that is intended to be compared using string equality matching. The value of the "software_version" SHOULD change on any update to the client software. Authorization servers MUST treat this field as self-asserted by the client and MUST NOT make any trusted decisions on the value of this field alone.

2.1. Human Readable Client Metadata

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"]".

3. Client Configuration Endpoint

The client configuration endpoint is an OAuth 2.0 protected resource that is provisioned by the server to facilitate viewing, updating, and deleting a client's registered information. The location of this endpoint is communicated to the client through the "registration_client_uri" member of the Client Information Response (Section 4.1). The client MUST use its registration access token in all calls to this endpoint as an OAuth 2.0 Bearer Token [RFC6750].

Operations on this endpoint are switched through the use of different HTTP methods [RFC2616]. If an authorization server does not support a particular method on the client configuration endpoint, it MUST respond with the appropriate error code.

3.1. Forming the Client Configuration Endpoint URL

The authorization server MUST provide the client with the fully qualified URL in the "registration_client_uri" element of the Client Information Response (Section 4.1). The authorization server MUST NOT expect the client to construct or discover this URL on its own. The client MUST use the URL as given by the server and MUST NOT construct this URL from component pieces.

Depending on deployment characteristics, the client configuration endpoint URL may take any number of forms. It is RECOMMENDED that this endpoint URL be formed through the use of a server-constructed URL string which combines the client registration endpoint's URL and the issued "client_id" for this client, with the latter as either a path parameter or a query parameter. For example, a client with the client identifier "s6BhdRkqt3" could be given a client configuration endpoint URL of "https://server.example.com/register/s6BhdRkqt3" (path parameter) or of "https://server.example.com/register?client_id=s6BhdRkqt3" (query parameter). In both of these cases, the client simply uses the URL as given by the authorization server.

These common patterns can help the server to more easily determine the client to which the request pertains, which MUST be matched

against the client to which the registration access token was issued. If desired, the server MAY simply return the client registration endpoint URL as the client configuration endpoint URL and change behavior based on the authentication context provided by the registration access token.

3.2. Client Read Request

To read the current configuration of the client on the authorization server, the client makes an HTTP GET request to the client configuration endpoint, authenticating with its registration access token.

Following is a non-normative example request (with line wraps for display purposes only):

```
GET /register/s6BhdRkqt3 HTTP/1.1
Accept: application/json
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483
```

Upon successful read of the information for a currently active client, the authorization server responds with an HTTP 200 OK with content type of "application/json" and a payload as described in Client Information Response (Section 4.1). Some values in the response, including the "client_secret" and "registration_access_token", MAY be different from those in the initial registration response. If the authorization server includes a new client secret and/or registration access token in its response, the client MUST immediately discard its previous client secret and/or registration access token. The value of the "client_id" MUST NOT change from the initial registration response.

If the registration access token used to make this request is not valid, the server MUST respond with an error as described in OAuth Bearer Token Usage [RFC6750].

If the client does not exist on this server, the server MUST respond

with HTTP 401 Unauthorized and the registration access token used to make this request SHOULD be immediately revoked.

If the client does not have permission to read its record, the server MUST return an HTTP 403 Forbidden.

3.3. Client Update Request

This operation updates a previously-registered client with new metadata at the authorization server. This request is authenticated by the registration access token issued to the client.

The client sends an HTTP PUT to the client configuration endpoint with a content type of "application/json". The HTTP entity payload is a JSON [RFC4627] document consisting of a JSON object and all parameters as top- level members of that JSON object.

This request MUST include all fields described in Client Metadata (Section 2) as returned to the client from a previous register, read, or update operation. The client MUST NOT include the "registration_access_token", "registration_client_uri", "client_secret_expires_at", or "client_id_issued_at" fields described in Client Information Response (Section 4.1).

Valid values of client metadata fields in this request MUST replace, not augment, the values previously associated with this client. Omitted fields MUST be treated as null or empty values by the server.

The client MUST include its "client_id" field in the request, and it MUST be the same as its currently-issued client identifier. If the client includes the "client_secret" field in the request, the value of this field MUST match the currently-issued client secret for that client. The client MUST NOT be allowed to overwrite its existing client secret with its own chosen value.

For all metadata fields, the authorization server MAY replace any invalid values with suitable default values, and it MUST return any such fields to the client in the response.

For example, a client could send the following request to the client registration endpoint to update the client registration in the above example with new information:

Following is a non-normative example request (with line wraps for display purposes only):

```
PUT /register/s6BhdRkqt3 HTTP/1.1
Accept: application/json
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483

{
 "client_id":"s6BhdRkqt3",
 "client_secret": "cf136dc3c1fc93f31185e5885805d",
 "redirect_uris":["https://client.example.org/callback",
    "https://client.example.org/alt"],
 "scope": "read write dolphin",
 "grant_types": ["authorization_code", "refresh_token"]
 "token_endpoint_auth_method": "client_secret_basic",
 "jwks_uri": "https://client.example.org/my_public_keys.jwks"
 "client_name":"My New Example",
```

```
 "client_name#fr":"Mon Nouvel Exemple",
 "logo_uri":"https://client.example.org/newlogo.png"
 "logo_uri#fr":"https://client.example.org/fr/newlogo.png"
}
```

Upon successful update, the authorization server responds with an HTTP 200 OK Message with content type "application/json" and a payload as described in Client Information Response (Section 4.1). Some values in the response, including the "client_secret" and r"egistration_access_token", MAY be different from those in the initial registration response. If the authorization server includes a new client secret and/or registration access token in its response, the client MUST immediately discard its previous client secret and/or registration access token. The value of the "client_id" MUST NOT change from the initial registration response.

If the registration access token used to make this request is not valid, the server MUST respond with an error as described in OAuth Bearer Token Usage [RFC6750].

If the client does not exist on this server, the server MUST respond with HTTP 401 Unauthorized, and the registration access token used to make this request SHOULD be immediately revoked.

If the client is not allowed to update its records, the server MUST respond with HTTP 403 Forbidden.

If the client attempts to set an invalid metadata field and the authorization server does not set a default value, the authorization

server responds with an error as described in Client Registration Error Response [DynReg].

3.4. Client Delete Request

To deprovision itself on the authorization server, the client makes an HTTP DELETE request to the client configuration endpoint. This request is authenticated by the registration access token issued to the client.

Following is a non-normative example request (with line wraps for

display purposes only):

```
DELETE /register/s6BhdRkqt3 HTTP/1.1
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483
```

A successful delete action will invalidate the "client_id", "client_secret", and "registration_access_token" for this client, thereby preventing the "client_id" from being used at either the authorization endpoint or token endpoint of the authorization server. The authorization server SHOULD immediately invalidate all existing authorization grants and currently-active tokens associated with this client.

If a client has been successfully deprovisioned, the authorization server responds with an HTTP 204 No Content message.

If the server does not support the delete method, the server MUST respond with an HTTP 405 Not Supported.

If the registration access token used to make this request is not valid, the server MUST respond with an error as described in OAuth Bearer Token Usage [RFC6750].

If the client does not exist on this server, the server MUST respond with HTTP 401 Unauthorized and the registration access token used to make this request SHOULD be immediately revoked.

If the client is not allowed to delete itself, the server MUST respond with HTTP 403 Forbidden.

Following is a non-normative example response:


```
HTTP/1.1 204 No Content
Cache-Control: no-store
Pragma: no-cache
```

4. Responses

In response to certain requests from the client to either the client registration endpoint or the client configuration endpoint as described in this specification, the authorization server sends the following response bodies.

4.1. Client Information Response

This specification extends the client information response defined in OAuth 2.0 Core Client Dynamic Registration. The response contains the client identifier as well as the client secret, if the client is a confidential client. The response also contains the fully qualified URL of the client configuration endpoint for this specific client that the client may use to obtain and update information about itself. The response also contains a registration access token that is to be used by the client to perform subsequent operations at the client configuration endpoint.

client_id
: REQUIRED. The unique client identifier, MUST NOT be currently valid for any other registered client.

client_secret
: OPTIONAL. The client secret. If issued, this MUST be unique for each "client_id". This value is used by confidential clients to authenticate to the token endpoint as described in OAuth 2.0 [RFC6749] Section 2.3.1.

client_id_issued_at
: OPTIONAL. Time at which the Client Identifier was issued. The time is represented as the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

client_secret_expires_at
: REQUIRED if "client_secret" is issued. Time at which the "client_secret" will expire or 0 if it will not expire. The time is represented as the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

registration_access_token

REQUIRED. Access token that is used at the client configuration endpoint to perform subsequent operations upon the client registration.

registration_client_uri
REQUIRED. The fully qualified URL of the client configuration endpoint for this client. The client MUST use this URL as given when communicating with the client configuration endpoint.

Additionally, the Authorization Server MUST return all registered metadata (Section 2) about this client, including any fields provisioned by the authorization server itself. The authorization server MAY reject or replace any of the client's requested metadata values submitted during the registration or update requests and substitute them with suitable values.

The response is an "application/json" document with all parameters as top-level members of a JSON object [RFC4627].

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
 "registration_access_token": "reg-23410913-abewfq.123483",
 "registration_client_uri":
    "https://server.example.com/register/s6BhdRkqt3",
 "client_id":"s6BhdRkqt3",
 "client_secret": "cf136dc3c1fc93f31185e5885805d",
 "client_id_issued_at":2893256800
 "client_secret_expires_at":2893276800
 "client_name":"My Example Client",
 "client_name#ja-Jpan-JP":
    "\u30AF\u30E9\u30A4\u30A2\u30F3\u30C8\u540D",
 "redirect_uris":["https://client.example.org/callback",
    "https://client.example.org/callback2"]
 "scope": "read write dolphin",
 "grant_types": ["authorization_code", "refresh_token"]
 "token_endpoint_auth_method": "client_secret_basic",
 "logo_uri": "https://client.example.org/logo.png",
 "jwks_uri": "https://client.example.org/my_public_keys.jwks"
}
```

5. IANA Considerations

This specification makes no requests of IANA.

6. Security Considerations

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

The authorization server MUST treat all client metadata as self-asserted. For instance, a rogue client might use the name and logo for the legitimate client which it is trying to impersonate. Additionally, a rogue client might try to use the software identifier or software version of a legitimate client to attempt to associate itself on the authorization server instances of the legitimate client. To counteract this, an authorization server needs to take steps to mitigate this phishing risk by looking at the entire registration request and client configuration. 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 could also refuse registration from a known software identifier that is requesting different redirect URIs or a different client homepage uri. An authorization server can also present warning messages to end users about dynamically registered clients in all cases, especially if such clients have been recently registered or have not been trusted by any users at the authorization server before.

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.

While the client secret can expire, the registration access token should not expire while a client is still actively registered. If this token were to expire, a developer or client could be left in a situation where they have no means of retrieving or updating the client's registration information. Were that the case, a new registration would be required, thereby generating a new client identifier. However, to limit the exposure surface of the

registration access token, the registration access token MAY be rotated when the developer or client does a read or update operation on the client's client configuration endpoint. As the registration access tokens are relatively long-term credentials, and since the registration access token is a Bearer token and acts as the sole authentication for use at the client configuration endpoint, it MUST be protected by the developer or client as described in OAuth 2.0 Bearer Token Usage [RFC6750].

If a client is deprovisioned from a server, 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. To prevent accidental disclosure from such an erroneous situation, the authorization server MUST treat all such requests as if the registration access token was invalid (by returning an HTTP 401 Unauthorized error, as described).

7. Normative References

[DynReg] Richer, J., "OAuth 2.0 Core Dynamic Client Registration", draft-richer-oauth-dyn-reg-core (work in progress), August 2013.

[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.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[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

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, Derek Atkins, Tim Bray, Domenico Catalano, Donald Coffin, Vladimir Dzhuvinov, George Fletcher, Thomas Hardjono, Phil Hunt, William Kim, 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 Partitioned dyn-reg specification into core and management specs

-14

o Added software_id and software_version metadata fields

o Added direct references to RFC6750 errors in read/update/delete methods

-13

o Fixed broken example text in registration request and in delete request

o Added security discussion of separating clients of different grant types

- Fixed error reference to point to RFC6750 instead of RFC6749
- Clarified that servers must respond to all requests to configuration endpoint, even if it's just an error code
- Lowercased all Terms to conform to style used in RFC6750

-12

- Improved definition of Initial Access Token
- Changed developer registration scenario to have the Initial Access Token gotten through a normal OAuth 2.0 flow
- Moved non-normative client lifecycle examples to appendix
- Marked differentiating between auth servers as out of scope
- Added protocol flow diagram
- Added credential rotation discussion
- Called out Client Registration Endpoint as an OAuth 2.0 Protected Resource
- Cleaned up several pieces of text

-11

- Added localized text to registration request and response examples.
- Removed "client_secret_jwt" and "private_key_jwt".
- Clarified "tos_uri" and "policy_uri" definitions.
- Added the OAuth Token Endpoint Authentication Methods registry for registering "token_endpoint_auth_method" metadata values.
- Removed uses of non-ASCII characters, per RFC formatting rules.

- Changed "expires_at" to "client_secret_expires_at" and "issued_at" to "client_id_issued_at" for greater clarity.

- Added explanatory text for different credentials (Initial Access Token, Registration Access Token, Client Credentials) and what they're used for.

- Added Client Lifecycle discussion and examples.

- Defined Initial Access Token in Terminology section.

-10

- Added language to point out that scope values are service-specific

- Clarified normative language around client metadata

- Added extensibility to token_endpoint_auth_method using absolute URIs

- Added security consideration about registering redirect URIs

- Changed erroneous 403 responses to 401's with notes about token handling

- Added example for initial registration credential

-09

- Added method of internationalization for Client Metadata values

- Fixed SAML reference

-08

- Collapsed jwk_uri, jwk_encryption_uri, x509_uri, and x509_encryption_uri into a single jwks_uri parameter

- Renamed grant_type to grant_types since it's a plural value

- Formalized name of "OAuth 2.0" throughout document
- Added JWT Bearer Assertion and SAML 2 Bearer Assertion to example grant types
- Added response_types parameter and explanatory text on its use with and relationship to grant_types

-07

- Changed registration_access_url to registration_client_uri
- Fixed missing text in 5.1
- Added Pragma: no-cache to examples
- Changed "no such client" error to 403
- Renamed Client Registration Access Endpoint to Client Configuration Endpoint
- Changed all the parameter names containing "_url" to instead use "_uri"
- Updated example text for forming Client Configuration Endpoint URL

-06

- Removed secret_rotation as a client-initiated action, including removing client secret rotation endpoint and parameters.
- Changed _links structure to single value registration_access_url.
- Collapsed create/update/read responses into client info response.
- Changed return code of create action to 201.
- Added section to describe suggested generation and composition of Client Registration Access URL.
- Added clarifying text to PUT and POST requests to specify JSON in the body.

- Added Editor's Note to DELETE operation about its inclusion.

- Added Editor's Note to registration_access_url about alternate syntax proposals.

-05

- changed redirect_uri and contact to lists instead of space delimited strings

- removed operation parameter

- added _links structure

- made client update management more RESTful

- split endpoint into three parts

- changed input to JSON from form-encoded

- added READ and DELETE operations

- removed Requirements section

- changed token_endpoint_auth_type back to token_endpoint_auth_method to match OIDC who changed to match us

-04

- removed default_acr, too undefined in the general OAuth2 case

- removed default_max_auth_age, since there's no mechanism for supplying a non-default max_auth_age in OAuth2

- clarified signing and encryption URLs

- changed token_endpoint_auth_method to token_endpoint_auth_type to match OIDC

-03

- added scope and grant_type claims

- fixed various typos and changed wording for better clarity

- endpoint now returns the full set of client information

- operations on client_update allow for three actions on metadata: leave existing value, clear existing value, replace existing value with new value

-02

- Reorganized contributors and references
- Moved OAuth references to RFC
- Reorganized model/protocol sections for clarity
- Changed terminology to "client register" instead of "client associate"
- Specified that client_id must match across all subsequent requests
- Fixed RFC2XML formatting, especially on lists

-01

- Merged UMA and OpenID Connect registrations into a single document
- Changed to form-paramter inputs to endpoint
- Removed pull-based registration

-00

- Imported original UMA draft specification

Authors' Addresses

Justin Richer (editor)
The MITRE Corporation

Email: jricher@mitre.org

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/

Maciej Machulak
Newcastle University

Email: m.p.machulak@ncl.ac.uk
URI: http://ncl.ac.uk/