OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 27, 2014

J. Richer, Ed.
The MITRE Corporation
J. Bradley
Ping Identity
M. Jones
Microsoft
M. Machulak
Newcastle University
August 26, 2013

OAuth 2.0 Core Dynamic Client Registration
draft-richer-oauth-dyn-reg-core-00

Abstract

This specification defines an endpoint and protocol for dynamic registration of OAuth 2.0 clients at an authorization server.

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 draft provides a mechanism for a client to register itself with the authorization server, which can be used to dynamically provision a client identifier, and optionally a client

secret. Additionally, the mechanisms in this draft may can be used by a client developer to register the client with the authorization server in a programmatic fashion.

As part of the registration process, this specification also defines a mechanism for the client to present the authorization server with a set of metadata, such as a set of valid redirect URIs.

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 Registration Endpoint OAuth 2.0 endpoint through which a client can be registered at an authorization server. The means by which the URL for this endpoint are obtained are out of scope for this specification.

Initial Access Token OAuth 2.0 access token optionally issued by an Authorization Server and used to authorize calls to the client registration endpoint. The type and format of this token are likely service-specific and are out of scope for this specification. The means by which the authorization server issues this token as well as the means by which the registration endpoint validates this token are out of scope for this specification.

1.3. Protocol Flow

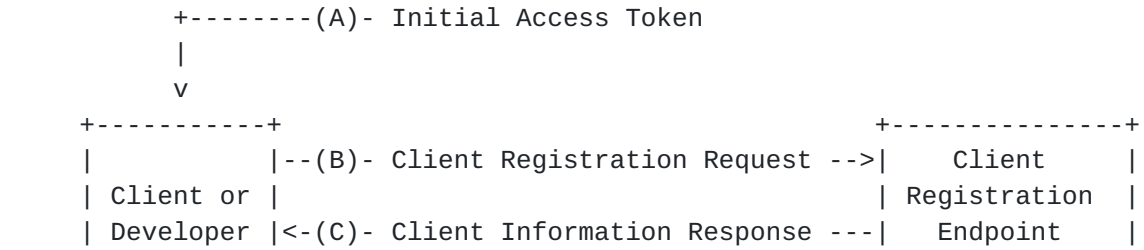

Figure 1: Abstract Protocol Flow

The abstract OAuth 2.0 Client dynamic registration flow illustrated in Figure 1 describes the interaction between the client or developer and the endpoint defined in this specification. 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, and possibly other values.

2. Client Metadata

Clients generally have an array of metadata associated with their unique client identifier at the authorization server, such as the list of valid redirect URIs.

The client metadata values serve two parallel purposes in the overall OAuth 2.0 dynamic client registration protocol:

- the client requesting its desired values for each parameter to the authorization server in a register (Section 3.1) request, and
- the authorization server informing the client of the current values of each parameter that the client has been registered to use through a client information response (Section 4.1).

An authorization server MAY override any value that a client requests during the registration process (including any omitted values) and replace the requested value with a default at the server's

discretion. The authorization server SHOULD provide documentation for any fields that it requires to be filled in by the client or to have particular values or formats. An authorization server MAY ignore the values provided by the client for any field in this list.

Extensions and profiles of this specification MAY expand this list, and authorization servers MUST accept all fields in this list. The authorization server MUST ignore any additional parameters sent by the Client that it does not understand.

redirect_uris
: Array of redirect URIs for use in redirect-based flows such as the authorization code and implicit grant types. It is RECOMMENDED that clients using these flows register this parameter, and an authorization server SHOULD require registration of valid redirect URIs for all clients that use these grant types to protect against token and credential theft attacks.

token_endpoint_auth_method
: The requested authentication method for the token endpoint. 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 5.1. Absolute URIs can also be used as values for this parameter without being registered. 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.

grant_types
: Array of 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 defined in OAuth JWT Bearer Token Profiles [OAuth.JWT].
* "urn:ietf:params:oauth:grant-type:saml2-bearer": The SAML 2 Bearer Grant 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. If the token endpoint is used in the grant type, 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
Array of the 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. If the authorization endpoint is used by the grant type, 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.

2.1. 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.

3. Client Registration Endpoint

The client registration endpoint is an OAuth 2.0 endpoint defined in this document that is designed to allow a client to be registered with the authorization server. The client registration endpoint MUST accept HTTP POST messages with request parameters encoded in the entity body using the "application/json" format. The client registration endpoint MUST be protected by a transport-layer security mechanism, and 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].

The client registration endpoint MAY be an OAuth 2.0 protected resource and accept an initial access token in the form of an OAuth 2.0 [RFC6749] access token to limit registration to only previously authorized parties. The method by which the initial access token is obtained by the registrant is generally out-of-band and is out of scope for this specification. The method by which the initial access token is verified and validated by the client registration endpoint is out of scope for this specification.

To support open registration and facilitate wider interoperability, the client registration endpoint SHOULD allow initial registration requests with no authorization (which is to say, with no OAuth 2.0 access token in the request). These requests MAY be rate-limited or otherwise limited to prevent a denial-of-service attack on the client registration endpoint.

The client registration endpoint MUST ignore all parameters it does not understand.

3.1. Client Registration Request

This operation registers a new client to the authorization server. The authorization server assigns this client a unique client identifier, optionally assigns a client secret, and associates the metadata given in the request with the issued client identifier. The request includes any parameters described in Client Metadata (Section 2) that the client wishes to specify for itself during the registration. The authorization server MAY provision default values for any items omitted in the client metadata.

To register, the client or developer sends an HTTP POST to the client registration 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.

For example, if the server supports open registration (with no initial access token), the client could send the following registration request to the client registration endpoint:

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

```
POST /register HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: server.example.com

{
 "redirect_uris":["https://client.example.org/callback",
    "https://client.example.org/callback2"],
 "token_endpoint_auth_method":"client_secret_basic",
 "scope":"read write dolphin",
 "extension_parameter":"foo"
}
```

Alternatively, if the server supports authorized registration, the developer or the client will be provisioned with an initial access token (the method by which the initial access token is obtained is out of scope for this specification). The developer or client sends the following authorized registration request to the client registration endpoint. Note that the initial access token sent in this example as an OAuth 2.0 Bearer Token [RFC6750], but any OAuth 2.0 token type could be used by an authorization server:

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

```
POST /register HTTP/1.1
Content-Type: application/json
Accept: application/json
Authorization: Bearer ey23f2.adfj230.af32-developer321
Host: server.example.com

{
 "redirect_uris":["https://client.example.org/callback",
    "https://client.example.org/callback2"],
 "token_endpoint_auth_method":"client_secret_basic",
 "scope":"read write dolphin",
 "extension_parameter":"foo"
}
```

3.2. Client Registration Response

Upon successful registration, the authorization server generates a new client identifier for the client. This client identifier MUST be unique at the server and MUST NOT be in use by any other client. The server responds with an HTTP 201 Created code and a body of type "application/json" with content described in Client Information Response (Section 4.1).

Upon an unsuccessful registration, the authorization server responds with an error as described in Client Registration Error (Section 4.2).

4. Responses

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

4.1. Client Information Response

The response contains the client identifier as well as the client secret, if the client is a confidential client. The response MAY contain additional fields as specified by extensions to this specification.

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.

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

{
 "client_id":"s6BhdRkqt3",
 "client_secret": "cf136dc3c1fc93f31185e5885805d",
 "client_id_issued_at":2893256800
 "client_secret_expires_at":2893276800
 "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",
 "extension_parameter": "foo"
}
```

4.2. Client Registration Error Response

When an OAuth 2.0 error condition occurs, such as the client presenting an invalid initial access token, the authorization server returns an error response appropriate to the OAuth 2.0 token type.

When a registration error condition occurs, the authorization server returns an HTTP 400 status code (unless otherwise specified) with content type "application/json" consisting of a JSON object [RFC4627] describing the error in the response body.

The JSON object contains two members:

error
: The error code, a single ASCII string.

error_description
: A human-readable text description of the error for debugging.

This specification defines the following error codes:

invalid_redirect_uri
: The value of one or more "redirect_uris" is invalid.

invalid_client_metadata
: The value of one of the client metadata (Section 2) fields is invalid and the server has rejected this request. Note that an Authorization server MAY choose to substitute a valid value for any requested parameter of a client's metadata.

Following is a non-normative example of an error response (with line wraps for display purposes only):

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
 "error":"invalid_redirect_uri",
 "error_description":"The redirect URI of http://sketchy.example.com
   is not allowed for this server."
}
```

5. IANA Considerations

5.1. 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.

5.1.1. 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.

5.1.2. 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]]

6. Security Considerations

Since requests to the client registration endpoint result in the transmission of clear-text credentials (in the HTTP request and response), the Authorization 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].

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.

Public clients MAY register with an authorization server using this protocol, if the authorization server's policy allows them. Public clients use a "none" value for the "token_endpoint_auth_method" metadata field and are generally used with the "implicit" grant type. Often these clients will be short-lived in-browser applications requesting access to a user's resources and access is tied to a user's active session at the authorization server. Since such clients often do not have long-term storage, it's possible that such clients would need to re-register every time the browser application is loaded. Additionally, such clients may not have ample opportunity to unregister themselves using the delete action before the browser closes. To avoid the resulting proliferation of dead client identifiers, an authorization server MAY decide to expire registrations for existing clients meeting certain criteria after a period of time has elapsed.

Since different OAuth 2.0 grant types have different security and usage parameters, an authorization server MAY require separate registrations for a piece of software to support multiple grant types. For instance, an authorization server might require that all clients using the "authorization_code" grant type make use of a client secret for the "token_endpoint_auth_method", but any clients using the "implicit" grant type do not use any authentication at the token endpoint. In such a situation, a server MAY disallow clients from registering for both the "authorization_code" and "implicit" grant types simultaneously. Similarly, the "authorization_code" grant type is used to represent access on behalf of an end user, but the "client_credentials" grant type represents access on behalf of the client itself. For security reasons, an authorization server could require that different scopes be used for these different use cases, and as a consequence it MAY disallow these two grant types from being registered together by the same client. In all of these cases, the authorization server would respond with an "invalid_client_metadata" error response (Section 4.2).

7. Normative References

[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. Use Cases

[[Editor's Note: These are some of the collected use cases that this protocol can address, they still need to be refactored into the two specifications.]]

In the OAuth 2.0 specification [RFC6749], a client is identified by its own unique Client identifier ("client_id") at each authorization server that it associates with. Dynamic registration as defined in this document is one way for a client to get a client identifier and associate a set of metadata with that identifier. Lack of such a client identifier is the expected trigger for a client registration operation.

In many cases, this client identifier is a unique, pairwise association between a particular running instance of a piece of client software and a particular running instance of an authorization server software. In particular:

- A single instance of client software (such as a Web server) talking to multiple authorization servers will need to register with each authorization server separately, creating a distinct client identifier with each authorization server. The client can not make any assumption that the authorization servers are correlating separate registrations of the client software together without further profiles and extensions to this specification document. The means by which a client discovers and differentiates between multiple authorization servers is out of scope for this specification.
- Multiple instances of client software (such as a native application installed on multiple devices simultaneously) talking to the same authorization server will need to each register with that authorization server separately, creating a distinct client identifier for each copy of the application. The authorization server cannot make any assumption of correlation between these clients without further specifications, profiles, and extensions to this specification. The client can not make any assumption that the authorization server will correlate separate registrations of the client software together without further profiles and extensions to this specification document.

A client identifier (and its associated credentials) could also be shared between multiple instances of a client. Mechanisms for sharing client identifiers between multiple instances of a piece of software (either client or authorization server) are outside the scope of this specification, as it is expected that every successful registration request (Section 3.1) results in the issuance of a new client identifier.

There are several patterns of OAuth client registration that dynamic registration protocol can enable. The following non-normative example lifecycle descriptions are not intended to be an exhaustive list. It is assumed that the authorization server supports the dynamic registration protocol and that all necessary discovery steps (which are out of scope for this specification) have already been performed.

B.1. Open Registration

Open registration, with no authorization required on the client registration endpoint, works as follows:

a. A client needs to get OAuth 2.0 tokens from an authorization server, but the client does not have a client identifier for that authorization server.
b. The client sends an HTTP POST request to the client registration endpoint at the authorization server and includes its metadata.
c. The authorization server issues a client identifier and returns it to the client along with a registration access token and a reference to the client's client configuration endpoint.
d. The client stores the returned response from the authorization server. At a minimum, it should remember the values of "client_id", "client_secret" (if present), "registration_access_token", and "registration_client_uri".
e. The client uses the its "client_id" and "client_secret" (if provided) to request OAuth 2.0 tokens using any valid OAuth 2.0 flow for which it is authorized.
f. If the client's "client_secret" expires or otherwise stops working, the client sends an HTTP GET request to the "registration_client_uri" with the "registration_access_token" as its authorization. This response will contain the client's refreshed "client_secret" along with any changed metadata values. Its "client_id" will remain the same.
g. If the client needs to update its configuration on the authorization server, it sends an HTTP PUT request to the "registration_client_uri" with the "registration_access_token" as its authorization. This response will contain the client's changed metadata values. Its "client_id" will remain the same.

h. If the client is uninstalled or otherwise deprovisioned, it can send an HTTP DELETE request to the "registration_client_uri" with the "registration_access_token" as its authorization. This will effectively deprovision the client from the authorization server.

B.2. Stateless Open Registration using JWT

Open registration, with no authorization required on the client registration endpoint. The registration endpoint/Authorization server maintain no state for the client. All information is storted in the client_id that is returned to the client and passed back to the Authorization server and Token Endpoint on subsiquent requests. If the client is using the implicit flow then the JWT MUST include the redirect URI and be signed by the AS for its later consumption. If the client is registering it's public key for use in the self signed assertion flow, the JWT MUST include the client's public key in the signed JWT. If the client is using a symetric client secret, the AS MUST include the secret as a claim in the JWT and encrypt or sign and encrypt the token to itself as appropriate. This method is transperent to the client and requires no aditional paramaters.

The flow works as follows:

a. A client needs to get OAuth 2.0 tokens from an authorization server, but the client does not have a client identifier for that authorization server.
b. The client sends an HTTP POST request to the client registration endpoint at the authorization server and includes its metadata.
c. The authorization server creates a JWE containing the required metadata such as redirect_uri and client secret for http basic authentication. (For clients using the assertion flow for authentication the registration endpoint can create a JWS containing the clients public key)
d. The authorization server issues the JWT as the client identifier and returns it to the client along with a JWT registration access token and a reference to the client's client configuration endpoint. (The client_id cannot be changed currently so updates are not possable the registration access token would only allow for reads)
e. The client stores the returned response from the authorization server. At a minimum, it should remember the values of "client_id", "client_secret" (if present), "registration_access_token", and "registration_client_uri".
f. The client uses the its "client_id" and "client_secret" (if provided) to request OAuth 2.0 tokens using any valid OAuth 2.0 flow for which it is authorized.
g. If the client's "client_secret" expires or otherwise stops working, the client must re-register.

B.3. Protected Registration

An authorization server may require an initial access token for requests to its registration endpoint. While the method by which a client receives this initial Access token and the method by which the authorization server validates this initial access token are out of scope for this specification, a common approach is for the developer to use a manual pre-registration portal at the authorization server that issues an initial access token to the developer. This allows the developer to package the initial access token with different instances of the client application. While each copy of the application would get its own client identifier (and registration access token), all instances of the application would be tied back to the developer by their shared use of this initial access token.

a. A developer is creating a client to use an authorization server and knows that instances of the client will dynamically register at runtime, but that the authorization server requires authorization at the registration endpoint.
b. The developer visits a manual pre-registration page at the authorization server and is issued an initial access token in the form of an OAuth 2.0 Bearer Token [RFC6750].
c. The developer packages that token with all instances of the client application.
d. The client needs to get OAuth 2.0 tokens from an authorization server, but the client does not have a client identifier for that authorization server.
e. The client sends an HTTP POST request to the client registration endpoint at the authorization server with its metadata, and the initial access token as its authorization.
f. The authorization server issues a client identifier and returns it to the client along with a registration access token and a reference to the client's client configuration endpoint.
g. The client stores the returned response from the authorization server. At a minimum, it should know the values of "client_id", "client_secret" (if present), "registration_access_token", and "registration_client_uri".
h. The client uses the its "client_id" and "client_secret" (if provided) to request OAuth 2.0 tokens using any supported OAuth 2.0 flow for which this client is authorized.
i. If the client's "client_secret" expires or otherwise stops working, the client sends an HTTP GET request to the "registration_client_uri" with the "registration_access_token" as its authorization. This response will contain the client's refreshed "client_secret" along with any metadata values registered to that client, some of which may have changed. Its "client_id" will remain the same.

j. If the client needs to update its configuration on the authorization server, it sends an HTTP PUT request to the "registration_client_uri" with the "registration_access_token" as its authorization. The response will contain the client's changed metadata values. Its "client_id" will remain the same.
k. If the client is uninstalled or otherwise deprovisioned, it can send an HTTP DELETE request to the "registration_client_uri" with the "registration_access_token" as its authorization. This will effectively deprovision the client from the Authorization Server.

B.4. Developer Automation

The dynamic registration protocol can also be used in place of a manual registration portal, for instance as part of an automated build and deployment process. In this scenario, the authorization server may require an initial access token for requests to its registration endpoint, as described in Protected Registration (Appendix B.3). However, here the developer manages the client's registration instead of the client itself. Therefore, the initial registration token and registration access token all remain with the developer. The developer packages the client identifier with the client as part of the client's build process.

a. A developer is creating a client to use an authorization server and knows that instances of the client will not dynamically register at runtime.
b. If required for registrations at the authorization server, the developer performs an OAuth 2.0 authorization of his build environment against the authorization server using any valid OAuth 2.0 flow. The authorization server and is issues an initial access token to the developer's build environment in the form of an OAuth 2.0 Bearer Token [RFC6750].
c. The developer configures his build environment to send an HTTP POST request to the client registration endpoint at the authorization server with the client's metadata, using the initial access token obtained the previous step as an OAuth 2.0 Bearer Token [RFC6750].
d. The authorization server issues a client identifier and returns it to the developer along with a registration access token and a reference to the client's client configuration endpoint.
e. The developer packages the client identifier with the client and stores the "registration_access_token", and "registration_client_uri" in the deployment system.
f. The client uses the its "client_id" and "client_secret" (if provided) to request OAuth 2.0 tokens using any supported OAuth 2.0 flow.
g. If the client's "client_secret" expires or otherwise stops working, the developer's deployment system sends an HTTP GET

request to the "registration_client_uri" with the "registration_access_token" as its authorization. This response will contain the client's refreshed "client_secret" along with any changed metadata values. Its "client_id" will remain the same. These new values will then be packaged and shipped to or retrieved by instances of the client, if necessary.

h. If the developer needs to update its configuration on the authorization server, the deployment system sends an HTTP PUT request to the "registration_client_uri" with the "registration_access_token" as its authorization. This response will contain the client's changed metadata values. Its "client_id" will remain the same. These new values will then be packaged and shipped to or retrieved by instances of the client, if necessary.

i. If the client is deprovisioned, the developer's deployment system can send an HTTP DELETE request to the "registration_client_uri" with the "registration_access_token" as its authorization. This will effectively deprovision the client from the authorization server and prevent any instances of the client from functioning.

Appendix C. 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

[[Previous changelog from draft-ietf-oauth-dyn-reg]]

-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
o Fixed error reference to point to RFC6750 instead of RFC6749
o Clarified that servers must respond to all requests to configuration endpoint, even if it's just an error code
o 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/