OAuth Working Group   J. Richer, Ed.
Internet-Draft   The MITRE Corporation
Intended status: Standards Track   J. Bradley
Expires: November 06, 2013   Ping Identity
                                                    M.B. Jones
                                                     Microsoft
                                                   M. Machulak
                                           Newcastle University
                                                  May 05, 2013

OAuth 2.0 Dynamic Client Registration Protocol
draft-ietf-oauth-dyn-reg-10

Abstract

   This specification defines an endpoint and protocol for dynamic
   registration of OAuth 2.0 Clients at an Authorization Server and
   methods for the dynamically registered client 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 November 06, 2013.

Copyright Notice

to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

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, in order 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.

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 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.

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:

o Client Registration Endpoint: The OAuth 2.0 Endpoint through which
   a Client can request new registration. The means of the Client
   obtaining the URL for this endpoint are out of scope for this
   specification.

o Client Configuration Endpoint: The OAuth 2.0 Endpoint through
   which a specific Client can manage its registration information,
   provided by the Authorization Server to the Client. This URL for
   this endpoint is communicated to the client by the Authorization
   Server in the Client Information Response.

o Registration Access Token: An OAuth 2.0 Bearer Token issued by the
   Authorization Server through the Client Registration Endpoint
   which is used by the Client to authenticate itself during read,
   update, and delete operations. This token is associated with a
   particular Client.

2. **Client Metadata**

   Clients generally have an array of metadata associated with their
   unique Client Identifier at the Authorization Server. These can
   range from human-facing display strings, such as a client name, to
   items that impact the security of the protocol, 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:

   o   the Client requesting its desired values for each parameter to the
       Authorization Server in a register (Section 3.1) or update
       (Section 4.3) request, and

   o   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 5.1).

   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.

   Extensions and profiles of this specification MAY expand this list,
   but Authorization Servers MUST accept or ignore all parameters on
   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 in order to
      protect against token and credential theft attacks.

   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 2.2).

   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 MAY be internationalized as described in Human Readable
Client Metadata (Section 2.2).

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 MAY be internationalized as described in Human
   Readable Client Metadata (Section 2.2).

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 for the
   Client. The Authorization Server SHOULD display this URL to the
   End-User if it is given. The value of this field MAY be
   internationalized as described in Human Readable Client Metadata
   (Section 2.2).

token_endpoint_auth_method
   The requested authentication type for the Token Endpoint. Valid
   values are:

   * "none": this 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

   * "client_secret_jwt": the client uses the JWT Assertion
      [OAuth.JWT] profile with a symmetric signature using the
      "client_secret" issued by the server

   * "private_key_jwt": the client uses the JWT Assertion
      [OAuth.JWT] profile with its own private key, usually with its
      public key location indicated by the "jwks_uri" field

Other authentication methods may be defined by extension. 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. Other authentication methods MAY be
supported by using an absolute URI as the value of this parameter.

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.

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 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

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, 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.

policy_uri
    A URL location that the Client provides to the End-User to read
    about the how the profile data will be used. The Authorization
    Server SHOULD display this URL to the End-User if it is given.
    The value of this field MAY be internationalized as described in
    Human Readable Client Metadata (Section 2.2).

jwks_uri
    URL for the Client's JSON Web Key Set [JWK] document that is used
    for signing requests, such as requests to the Token Endpoint using
    the "private_key_jwt" assertion client credential. The keys MAY
    also be used for higher level protocols that require signing or
    encryption.

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)           |
+----------------------------------------------+------------------+
```


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

2.2. 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":
   "クライアント名"" 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 Registration Endpoint

The Client Registration Endpoint is an OAuth 2.0 Endpoint defined in
this document that is designed to allow a Client to register itself
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 accept an initial authorization
credential in the form of an OAuth 2.0 [RFC6749] access token in
order to limit registration to only previously authorized parties.
The method by which this access token is obtained by the registrant
is generally out-of-band and is out of scope of this specification.

In order to support open registration and facilitate wider
interoperability, the Client Registration Endpoint SHOULD allow
initial registration requests with no authentication. These requests
MAY be rate-limited or otherwise limited to prevent a denial-of-
service attack on the Client Registration Endpoint.

In order to facilitate registered clients updating their information,
the Client Registration Endpoint issues a Request Access Token for
clients to securely identify themselves in future connections to the
Client Configuration Endpoint (Section 4). As such, the Client
Configuration Endpoint MUST accept requests with OAuth 2.0 Bearer
Tokens [RFC6750] for these operations, whether or not the initial
registration call requires authentication of some form.

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.

The Client 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, a client could send the following open 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"]
 "client_name":"My Example Client",
 "token_endpoint_auth_method":"client_secret_basic",
 "scope":"read write dolphin",
 "logo_uri":"https://client.example.org/logo.png",
 "jwks_uri":"https://client.example.org/my_public_keys.jwks"
}
```

Alternatively, if the client has been provisioned with an initial
authorization credential, it 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
Authorization: Bearer ey23f2.adfj230.af32-developer321
Host: server.example.com

{
 "redirect_uris":["https://client.example.org/callback",
    "https://client.example.org/callback2"]
 "client_name":"My Example Client",
 "token_endpoint_auth_method":"client_secret_basic",
 "scope":"read write dolphin",
 "logo_uri":"https://client.example.org/logo.png",
 "jwks_uri":"https://client.example.org/my_public_keys.jwks"
}
```

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 5.1).

Upon an unsuccessful registration, the Authorization Server responds
with an error as described in Client Registration Error
(Section 5.2).

4. Client Configuration Endpoint

The Client Configuration Endpoint is an OAuth 2.0 protected endpoint
that is provisioned by the server for a specific client to be able to
view and update its registered information. The Client MUST include
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].

4.1. **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 5.1). 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 ID "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 follows the URL as given.

   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.

4.2. **Client Read Request**

   In order 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 5.1](#)).

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.

[4.3](#). 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", "expires_at",
or "issued_at" fields described in Client Information Response
([Section 5.1](#)).

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",
 "logo_uri":"https://client.example.org/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 5.1).
The Authorization Server MAY include a new Client Secret and/or
Registration Access Token in its response. If so, the Client MUST
immediately discard its previous Client Secret and/or Registration
Access Token.

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 (Section 5.2).

4.4. Client Delete Request

In order 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
Accept: application/json
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 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
```

5. 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.

5.1. Client Information Response

The response contains the Client Identifier as well as the Client
Secret, if the Client is a confidential Client. The response also
contains the fully qualified URL to 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
   Section 2.3.1.

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.

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.

registration_access_token
   REQUIRED. The Access token to be used by the client to perform
   actions on the Client Configuration Endpoint.

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",
 "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",
 "logo_uri": "https://client.example.org/logo.png",
 "jwks_uri": "https://client.example.org/my_public_keys.jwks"
}
```

5.2. Client Registration Error Response

When an OAuth 2.0 error condition occurs, such as the client
presenting an invalid Registration Access Token, the Authorization
Server returns an Error Response as defined in Section 5.2 of the
OAuth 2.0 specification.

When a registration error condition occurs, the Authorization Server
returns an HTTP 400 status code 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.

invalid_client_id
   Value of "client_id" is invalid.

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."
}

6. IANA Considerations

This document makes no requests of IANA.

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].

As this endpoint is an OAuth 2.0 Protected Resource, requests to the
Registration Endpoint SHOULD have some rate limiting on failures to
prevent the Registration Access Token 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
pre-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 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.

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" and "policy_uri"). 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" and "policy_uri" have the same host as the hosts defined
in the array of "redirect_uris".

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 Client could be left in a situation
where it has no means of updating itself and must register itself
anew. As the Registration Access Tokens are 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 Client as described in OAuth 2.0 Bearer
[RFC6750].

If a Client is deprovisioned from a server, any outstanding
Registration Access Tokens 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.

8. Normative References

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

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

[OAuth.JWT]
           Jones, M.B., 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.B. 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.

[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, Tim Bray,
Domenico Catalano, George Fletcher, Torsten Lodderstedt, Eve Maler,
Nov Matake, Thomas Hardjono, Nat Sakimura, and Christian Scholz.

Appendix B. Document History

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

-10

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

o Clarified normative language around client metadata

o Added extensibility to token_endpoint_auth_method using absolute
   URIs

o Added security consideration about registering redirect URIs

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

o Added example for initial registration credential

-09

o Added method of internationalization for Client Metadata values

o Fixed SAML reference

-08

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

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

o Formalized name of "OAuth 2.0" throughout document

o Added JWT Bearer Assertion and SAML 2 Bearer Assertion to example
   grant types

o Added response_types parameter and explanatory text on its use
   with and relationship to grant_types

-07

o Changed registration_access_url to registration_client_uri

o Fixed missing text in 5.1

o Added Pragma: no-cache to examples

o Changed "no such client" error to 403

o Renamed Client Registration Access Endpoint to Client
   Configuration Endpoint

o Changed all the parameter names containing "_url" to instead use
   "_uri"

o Updated example text for forming Client Configuration Endpoint URL

-06

o Removed secret_rotation as a client-initiated action, including
   removing client secret rotation endpoint and parameters.

o Changed _links structure to single value registration_access_url.

o Collapsed create/update/read responses into client info response.

o Changed return code of create action to 201.

o Added section to describe suggested generation and composition of
   Client Registration Access URL.

o Added clarifying text to PUT and POST requests to specify JSON in
   the body.

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

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

-05

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

o removed operation parameter

o added _links structure

o made client update management more RESTful

o split endpoint into three parts

o changed input to JSON from form-encoded

o added READ and DELETE operations

o removed Requirements section

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

-04

o removed default_acr, too undefined in the general OAuth2 case

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

o clarified signing and encryption URLs

o changed token_endpoint_auth_method to token_endpoint_auth_type to
   match OIDC

-03

o added scope and grant_type claims

o fixed various typos and changed wording for better clarity

o endpoint now returns the full set of client information

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

-02

o Reorganized contributors and references

   o Moved OAuth references to RFC

   o Reorganized model/protocol sections for clarity

   o Changed terminology to "client register" instead of "client
      associate"

   o Specified that client_id must match across all subsequent requests

   o Fixed RFC2XML formatting, especially on lists

   -01

   o Merged UMA and OpenID Connect registrations into a single document

   o Changed to form-paramter inputs to endpoint

   o Removed pull-based registration

   -00

   o 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

   Maciej Machulak
   Newcastle University

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