Internet Engineering Task Force (IETF)
Request for Comments: 7591
Category: Standards Track
ISSN: 2070-1721

J. Richer, Ed.

M. Jones
Microsoft

J. Bradley
Ping Identity

M. Machulak
Newcastle University

P. Hunt
Oracle Corporation

July 2015

OAuth 2.0 Dynamic Client Registration Protocol

Abstract

This specification defines mechanisms for dynamically registering OAuth 2.0 clients with authorization servers. Registration requests send a set of desired client metadata values to the authorization server. The resulting registration responses return a client identifier to use at the authorization server and the client metadata values registered for the client. The client can then use this registration information to communicate with the authorization server using the OAuth 2.0 protocol. This specification also defines a set of common client metadata fields and values for clients to use during registration.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7591.

Table of Contents

1. Introduction

In order for an OAuth 2.0 [RFC6749] client to utilize an OAuth 2.0 authorization server, the client needs specific information to interact with the server, including an OAuth 2.0 client identifier to use at that server. This specification describes how an OAuth 2.0 client can be dynamically registered with an authorization server to obtain this information.

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 redirection URIs. This metadata can either be communicated in a self-asserted fashion or as a set of metadata called a software statement, which is digitally signed or protected with a Message Authentication Code (MAC); in the case of a software statement, the issuer is vouching for the validity of the data about the client.

Traditionally, registration of a client with an authorization server is performed manually. The mechanisms defined in this specification can be used either for a client to dynamically register itself with authorization servers or for a client developer to programmatically register the client with authorization servers. Multiple applications using OAuth 2.0 have previously developed mechanisms for accomplishing such registrations. This specification generalizes the registration mechanisms defined by "OpenID Connect Dynamic Client Registration 1.0" [OpenID.Registration] and used by "User Managed Access (UMA) Profile of OAuth 2.0" [UMA-Core] in a way that is compatible with both, while being applicable to a wider set of OAuth 2.0 use cases.

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", "authorization code", "authorization endpoint", "authorization grant", "authorization server", "client", "client identifier", "client secret", "grant type", "protected resource", "redirection URI",

"refresh token", "resource owner", "resource server", "response type", and "token endpoint" defined by OAuth 2.0 [RFC6749] and uses the term "Claim" defined by JSON Web Token (JWT) [RFC7519].

This specification defines the following terms:

Client Software
: Software implementing an OAuth 2.0 client.

Client Instance
: A deployed instance of a piece of client software.

Client Developer
: The person or organization that builds a client software package and prepares it for distribution. At the time the client is built, the developer is often not aware of who the deploying service provider organizations will be. Client developers will need to use dynamic registration when they are unable to predict aspects of the software, such as the deployment URLs, at compile time. For instance, this can occur when the software API publisher and the deploying organization are not the same.

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 is obtained are out of scope for this specification.

Initial Access Token
: OAuth 2.0 access token optionally issued by an authorization server to a developer or client 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. Use of an initial access token is required when the authorization server limits the parties that can register a client.

Deployment Organization
: An administrative security domain under which a software API (service) is deployed and protected by an OAuth 2.0 framework. In some OAuth scenarios, the deployment organization and the software API publisher are the same. In these cases, the deploying organization will often have a close relationship with client software developers. In many other cases, the definer of the service may be an independent third-party publisher or a standards organization. When working to a published specification for an

API, the client software developer is unable to have a prior relationship with the potentially many deployment organizations deploying the software API (service).

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

Software API Publisher
The organization that defines a particular web-accessible API that may be deployed in one or more deployment environments. A publisher may be any standards body, commercial, public, private, or open source organization that is responsible for publishing and distributing software and API specifications that may be protected via OAuth 2.0. In some cases, a software API publisher and a client developer may be the same organization. At the time of publication of a web-accessible API, the software publisher often does not have a prior relationship with the deploying organizations.

Software Statement
A digitally signed or MACed JSON Web Token (JWT) [RFC7519] that asserts metadata values about the client software. In some cases, a software statement will be issued directly by the client developer. In other cases, a software statement will be issued by a third-party organization for use by the client developer. In both cases, the trust relationship the authorization server has with the issuer of the software statement is intended to be used as an input to the evaluation of whether the registration request is accepted. A software statement can be presented to an authorization server as part of a client registration request.

1.3. Protocol Flow

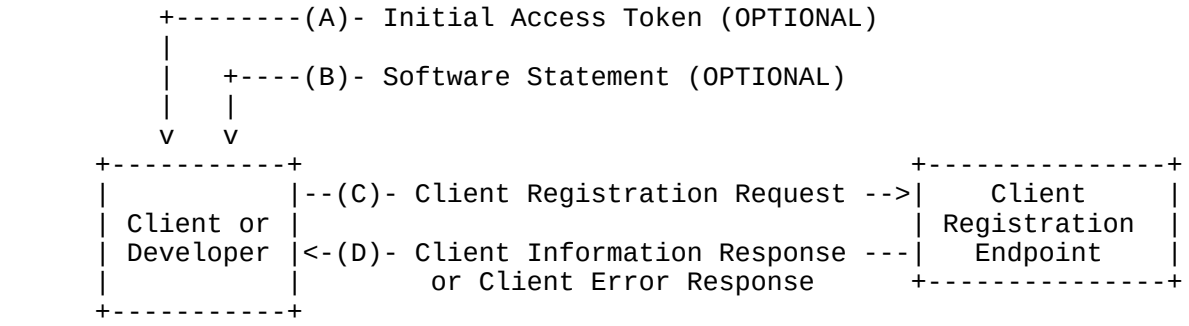

Figure 1: Abstract Dynamic Client Registration Flow

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 giving access to 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) Optionally, the client or developer is issued a software statement for use with the client registration endpoint. The method by which the software statement is issued to the client or developer is out of scope for this specification.

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

(D) The authorization server registers the client and returns:

* the client's registered metadata,

* a client identifier that is unique at the server, and

* a set of client credentials such as a client secret, if applicable for this client.

Examples of different configurations and usages are included in Appendix A.

2. Client Metadata

Registered clients have a set of metadata values associated with their client identifier at an authorization server, such as the list of valid redirection URIs or a display name.

These client metadata values are used in two ways:

- as input values to registration requests, and
- as output values in registration responses.

The following client metadata fields are defined by this specification. The implementation and use of all client metadata fields is OPTIONAL, unless stated otherwise. All data member types (strings, arrays, numbers) are defined in terms of their JSON [RFC7159] representations.

redirect_uris
: Array of redirection URI strings for use in redirect-based flows such as the authorization code and implicit flows. As required by Section 2 of OAuth 2.0 [RFC6749], clients using flows with redirection MUST register their redirection URI values. Authorization servers that support dynamic registration for redirect-based flows MUST implement support for this metadata value.

token_endpoint_auth_method
: String indicator of 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, Section 2.1, and does not have a client secret.
  * "client_secret_post": The client uses the HTTP POST parameters as defined in OAuth 2.0, Section 2.3.1.
  * "client_secret_basic": The client uses HTTP Basic as defined in OAuth 2.0, Section 2.3.1.

  Additional values can be defined via the IANA "OAuth Token Endpoint Authentication Methods" registry established in Section 4.2. 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 the HTTP Basic authentication scheme as specified in Section 2.3.1 of OAuth 2.0.

grant_types
: Array of OAuth 2.0 grant type strings that the client can use at the token endpoint. These grant types are defined as follows:

  * "authorization_code": The authorization code grant type defined in OAuth 2.0, Section 4.1.

  * "implicit": The implicit grant type defined in OAuth 2.0, Section 4.2.

  * "password": The resource owner password credentials grant type defined in OAuth 2.0, Section 4.3.

  * "client_credentials": The client credentials grant type defined in OAuth 2.0, Section 4.4.

  * "refresh_token": The refresh token grant type defined in OAuth 2.0, Section 6.

  * "urn:ietf:params:oauth:grant-type:jwt-bearer": The JWT Bearer Token Grant Type defined in OAuth JWT Bearer Token Profiles [RFC7523].

  * "urn:ietf:params:oauth:grant-type:saml2-bearer": The SAML 2.0 Bearer Assertion Grant defined in OAuth SAML 2 Bearer Token Profiles [RFC7522].

  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 grant type definition. Authorization servers MAY allow for other values as defined in the grant type extension process described in OAuth 2.0, Section 4.5. If omitted, the default behavior is that the client will use only the "authorization_code" Grant Type.

response_types
: Array of the OAuth 2.0 response type strings that the client can use at the authorization endpoint. These response types are defined as follows:

  * "code": The authorization code response type defined in OAuth 2.0, Section 4.1.

  * "token": The implicit response type defined in OAuth 2.0, Section 4.2.

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 grant type definition. Authorization servers MAY allow for other values as defined in the grant type extension process is described in OAuth 2.0, Section 4.5. If omitted, the default is that the client will use only the "code" response type.

client_name
: Human-readable string name of the client to be presented to the end-user during authorization. If omitted, the authorization server MAY display the raw "client_id" value to the end-user instead. It is RECOMMENDED that clients always send this field. The value of this field MAY be internationalized, as described in Section 2.2.

client_uri
: URL string of a web page providing information about 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 Section 2.2.

logo_uri
: URL string 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 Section 2.2.

scope
: String containing a space-separated list of scope values (as described in Section 3.3 of OAuth 2.0 [RFC6749]) that the client can use when requesting access tokens. The semantics of values in this list are service specific. If omitted, an authorization server MAY register a client with a default set of scopes.

contacts
: Array of strings representing ways to contact people responsible for this client, typically email addresses. The authorization server MAY make these contact addresses available to end-users for support requests for the client. See Section 6 for information on Privacy Considerations.

tos_uri
: URL string that points to a human-readable terms of service document for the client that describes a contractual relationship between the end-user and the client that the end-user accepts when authorizing the client. The authorization server SHOULD display this URL to the end-user if it is provided. The value of this field MUST point to a valid web page. The value of this field MAY be internationalized, as described in Section 2.2.

policy_uri
: URL string that points to a human-readable privacy policy document that describes how the deployment organization collects, uses, retains, and discloses personal data. The authorization server SHOULD display this URL to the end-user if it is provided. The value of this field MUST point to a valid web page. The value of this field MAY be internationalized, as described in Section 2.2.

jwks_uri
: URL string referencing the client's JSON Web Key (JWK) Set [RFC7517] document, which contains the client's public keys. The value of this field MUST point to a valid JWK Set document. These keys can be used by higher-level protocols that use signing or encryption. For instance, these keys might be used by some applications for validating signed requests made to the token endpoint when using JWTs for client authentication [RFC7523]. Use of this parameter is preferred over the "jwks" parameter, as it allows for easier key rotation. The "jwks_uri" and "jwks" parameters MUST NOT both be present in the same request or response.

jwks
: Client's JSON Web Key Set [RFC7517] document value, which contains the client's public keys. The value of this field MUST be a JSON object containing a valid JWK Set. These keys can be used by higher-level protocols that use signing or encryption. This parameter is intended to be used by clients that cannot use the "jwks_uri" parameter, such as native clients that cannot host public URLs. The "jwks_uri" and "jwks" parameters MUST NOT both be present in the same request or response.

software_id
: A unique identifier string (e.g., a Universally Unique Identifier (UUID)) assigned by the client developer or software publisher used by registration endpoints to identify the client software to be dynamically registered. Unlike "client_id", which is issued by the authorization server and SHOULD vary between instances, the "software_id" SHOULD remain the same for all instances of the client software. The "software_id" SHOULD remain the same across

multiple updates or versions of the same piece of software. The value of this field is not intended to be human readable and is usually opaque to the client and authorization server.

software_version
A version identifier string for the client software identified by "software_id". The value of the "software_version" SHOULD change on any update to the client software identified by the same "software_id". The value of this field is intended to be compared using string equality matching and no other comparison semantics are defined by this specification. The value of this field is outside the scope of this specification, but it is not intended to be human readable and is usually opaque to the client and authorization server. The definition of what constitutes an update to client software that would trigger a change to this value is specific to the software itself and is outside the scope of this specification.

Extensions and profiles of this specification can expand this list with metadata names and descriptions registered in accordance with the IANA Considerations in Section 4 of this document. The authorization server MUST ignore any client metadata sent by the client that it does not understand (for instance, by silently removing unknown metadata from the client's registration record during processing). The authorization server MAY reject any requested client metadata values by replacing requested values with suitable defaults as described in Section 3.2.1 or by returning an error response as described in Section 3.2.2.

Client metadata values can be either communicated directly in the body of a registration request, as described in Section 3.1, or included as claims in a software statement, as described in Section 2.3; a mixture of both is also possible. If the same client metadata name is present in both locations and the software statement is trusted by the authorization server, the value of a claim in the software statement MUST take precedence.

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, for example, by returning an "invalid_client_metadata" error response to an inconsistent registration request.

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

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

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

2.2. 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 facilitate use in different locations.

To specify the languages and scripts, BCP 47 [RFC5646] language tags are added to client metadata member names, delimited by a "#" character. Since JSON [RFC7159] 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 BCP 47 language tag values are case-insensitive, implementations SHOULD interpret the language tag values supplied in a case insensitive manner. Per the recommendations in BCP 47, 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 as a workaround the member "client_name#en-us" can be accessed using the JavaScript syntax "j["client_name#en-us"]".

2.3. Software Statement

A software statement is a JSON Web Token (JWT) [RFC7519] that asserts metadata values about the client software as a bundle. A set of claims that can be used in a software statement are defined in Section 2. When presented to the authorization server as part of a client registration request, the software statement MUST be digitally signed or MACed using JSON Web Signature (JWS) [RFC7515] and MUST contain an "iss" (issuer) claim denoting the party attesting to the claims in the software statement. It is RECOMMENDED that software statements be digitally signed using the "RS256" signature algorithm, although particular applications MAY specify the use of different algorithms. It is RECOMMENDED that software statements contain the "software_id" claim to allow authorization servers to correlate different instances of software using the same software statement.

For example, a software statement could contain the following claims:

```
{
 "software_id": "4NRB1-0XZABZI9E6-5SM3R",
 "client_name": "Example Statement-based Client",
 "client_uri": "https://client.example.net/"
}
```

The following non-normative example JWT includes these claims and has been asymmetrically signed using "RS256" (with line breaks for display purposes only):

```
eyJhbGciOiJSUzI1NiJ9.
eyJzb2Z0d2FyZV9pZCI6IjROUkIxLTBYWkFCWkk5RTYtNVNNM1IiLCJjbGll
bnRfbmFtZSI6IkV4YW1wbGUgU3RhdGVtZW50LWJhc2VkIENsaWVudCIsImNs
aWVudF91cmkiOiJodHRwczovL2NsaWVudC5leGFtcGxlLm5ldC8ifQ.
GHfL4QNIrQwL18BSRdE595T9jbzqa06R9BT8w409x9oIcKaZo_mt15riEXHa
zdISUvDIZhtiyNrSHQ8K4TvqWxH6uJgcmoodZdPwmWRIEYbQDLqPNxREtYn0
5X3AR7ia4FRjQ2ojZjk5fJqJdQ-JcfxyhK-P8BAWBd6I2LLA77IG32xtbhxY
fHX7VhuU5ProJO8uvu3Ayv4XRhLZJY4yKfmyjiiKiPNe-Ia4SMy_d_QSWxsk
U5XIQl5Sa2YRPMbDRXttm2TfnZM1xx70DoYi8g6czz-CPGRi4SW_S2RKHIJf
IjoI3zTJ0Y2oe0_EJAiXbL6OyF9S5tKxDXV8JIndSA
```

The software statement is typically distributed with all instances of a client application. The means by which a client or developer obtains a software statement are outside the scope of this specification. Some common methods could include a client developer generating a client-specific JWT by registering with a software API publisher to obtain a software statement for a class of clients.

The criteria by which authorization servers determine whether to trust and utilize the information in a software statement are outside the scope of this specification.

In some cases, authorization servers MAY choose to accept a software statement value directly as a client identifier in an authorization request, without a prior dynamic client registration having been performed. The circumstances under which an authorization server would do so, and the specific software statement characteristics required in this case, are outside the scope of this specification.

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, as described in Section 5.

The client registration endpoint MAY be an OAuth 2.0 [RFC6749] protected resource and it MAY accept an initial access token in the form of an OAuth 2.0 access token to limit registration to only previously authorized parties. The method by which the initial access token is obtained by the client or developer 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 registration requests with no authorization (which is to say, with no initial 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.

3.1. Client Registration Request

This operation registers a client with the authorization server. The authorization server assigns this client a unique client identifier, optionally assigns a client secret, and associates the metadata provided in the request with the issued client identifier. The request includes any client metadata parameters being specified for the client 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 [RFC7159] document consisting of a JSON object and all requested client metadata values 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.

The following is a non-normative example request not using an initial access token:

```
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",
 "client_name#ja-Jpan-JP":
    "\u30AF\u30E9\u30A4\u30A2\u30F3\u30C8\u540D",
 "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",
 "example_extension_parameter": "example_value"
}
```

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.

The following is a non-normative example request using an initial access token and registering a JWK Set by value (with line breaks within values 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",
   "client_name#ja-Jpan-JP":
      "\u30AF\u30E9\u30A4\u30A2\u30F3\u30C8\u540D",
   "token_endpoint_auth_method": "client_secret_basic",
   "policy_uri": "https://client.example.org/policy.html",
   "jwks": {"keys": [{
      "e": "AQAB",
      "n": "nj3YJwsLUFl9BmpAbkOswCNVx17Eh9wMO-_AReZwBqfaWFcfG
HrZXsIV2VMCNVNU8Tpb4obUaSXcRcQ-VMsfQPJm9IzgtRdAY8NN8Xb7PEcYyk
lBjvTtuPbpzIaqyiUepzUXNDFuAOOkrIol3WmflPUUgMKULBN0EUd1fpOD70p
RM0rlp_gg_WNUKoW1V-3keYUJoXH9NztEDm_D2MQXj9eGOJJ8yPgGL8PAZMLe
2R7jb9TxOCPDED7tY_TU4nFPlxptw59A42mldEmViXsKQt60s1SLboazxFKve
qXC_jpLUt22OC6GUG63p-REw-ZOr3r845z50wMuzifQrMI9bQ",
      "kty": "RSA"
   }]},
   "example_extension_parameter": "example_value"
  }
```

3.1.1. Client Registration Request Using a Software Statement

In addition to JSON elements, client metadata values MAY also be provided in a software statement, as described in Section 2.3. The authorization server MAY ignore the software statement if it does not support this feature. If the server supports software statements, client metadata values conveyed in the software statement MUST take precedence over those conveyed using plain JSON elements.

Software statements are included in the requesting JSON object using this OPTIONAL member:

software_statement
: A software statement containing client metadata values about the client software as claims. This is a string value containing the entire signed JWT.

In the following example, some registration parameters are conveyed as claims in a software statement from the example in Section 2.3, while some values specific to the client instance are conveyed as regular parameters (with line breaks within values 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"
    ],
    "software_statement": "eyJhbGciOiJSUzI1NiJ9.
eyJzb2Z0d2FyZV9pZCI6IjROUkIxLTBYWkFCWkk5RTYtNVNNM1IiLCJjbGll
bnRfbmFtZSI6IkV4YW1wbGUgU3RhdGVtZW50LWJhc2VkIENsaWVudCIsImNs
aWVudF91cmkiOiJodHRwczovL2NsaWVudC5leGFtcGxlLm5ldC8ifQ.
GHfL4QNIrQwL18BSRdE595T9jbzqa06R9BT8w409x9oIcKaZo_mt15riEXHa
zdISUvDIZhtiyNrSHQ8K4TvqWxH6uJgcmoodZdPwmWRIEYbQDLqPNxREtYn0
5X3AR7ia4FRjQ2ojZjk5fJqJdQ-JcfxyhK-P8BAWBd6I2LLA77IG32xtbhxY
fHX7VhuU5ProJO8uvu3Ayv4XRhLZJY4yKfmyjiiKiPNe-Ia4SMy_d_QSWxsk
U5XIQl5Sa2YRPMbDRXttm2TfnZM1xx70DoYi8g6czz-CPGRi4SW_S2RKHIJf
IjoI3zTJ0Y2oe0_EJAiXbL6OyF9S5tKxDXV8JIndSA",
    "scope": "read write",
    "example_extension_parameter": "example_value"
  }
```

3.2. Responses

Upon a successful registration request, the authorization server returns a client identifier for the client. The server responds with an HTTP 201 Created status code and a body of type "application/json" with content as described in Section 3.2.1.

Upon an unsuccessful registration request, the authorization server responds with an error, as described in Section 3.2.2.

3.2.1. 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. OAuth 2.0 client identifier string. It SHOULD NOT be currently valid for any other registered client, though an authorization server MAY issue the same client identifier to multiple instances of a registered client at its discretion.

client_secret
: OPTIONAL. OAuth 2.0 client secret string. If issued, this MUST be unique for each "client_id" and SHOULD be unique for multiple instances of a client using the same "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 1970-01-01T00:00:00Z as measured in UTC until the date/time of issuance.

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 1970-01-01T00:00:00Z as measured in UTC until the date/time of expiration.

Additionally, the authorization server MUST return all registered metadata 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 and substitute them with suitable values. The client or developer can check the values in the response to determine if the registration is sufficient for use (e.g., the registered "token_endpoint_auth_method" is supported by the client software) and determine a course of action appropriate for the client software. The response to such a situation is out of scope for this specification but could include filing a report with the application developer or authorization server provider, attempted re-registration with different metadata values, or various other methods. For instance, if the server also supports a registration management mechanism such as that defined in [RFC7592], the client or developer could attempt to update the registration with different metadata values. This process could also be aided by a service discovery protocol, such as [OpenID.Discovery], which can list a server's capabilities, allowing a client to make a more informed registration request. The use of any such management or discovery system is optional and outside the scope of this specification.

The successful registration response uses an HTTP 201 Created status code with a body of type "application/json" consisting of a single JSON object [RFC7159] with all parameters as top-level members of the object.

If a software statement was used as part of the registration, its value MUST be returned unmodified in the response along with other metadata using the "software_statement" member name. Client metadata elements used from the software statement MUST also be returned directly as top-level client metadata values in the registration response (possibly with different values, since the values requested and the values used may differ).

The following is a non-normative example response of a successful registration:

```
HTTP/1.1 201 Created
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"],
 "grant_types": ["authorization_code", "refresh_token"],
 "client_name": "My Example Client",
 "client_name#ja-Jpan-JP":
    "\u30AF\u30E9\u30A4\u30A2\u30F3\u30C8\u540D",
 "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",
 "example_extension_parameter": "example_value"
}
```

3.2.2. 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 [RFC7159] describing the error in the response body.

Two members are defined for inclusion in the JSON object:

error
: REQUIRED. Single ASCII error code string.

error_description
: OPTIONAL. Human-readable ASCII text description of the error used for debugging.

Other members MAY also be included and, if they are not understood, they MUST be ignored.

This specification defines the following error codes:

invalid_redirect_uri
: The value of one or more redirection URIs is invalid.

invalid_client_metadata
: The value of one of the client metadata 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_software_statement
: The software statement presented is invalid.

unapproved_software_statement
: The software statement presented is not approved for use by this authorization server.

The following is a non-normative example of an error response resulting from a redirection URI that has been blacklisted by the authorization server (with line breaks within values 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 redirection URI
   http://sketchy.example.com is not allowed by this server."
}
```

The following is a non-normative example of an error response resulting from an inconsistent combination of "response_types" and "grant_types" values (with line breaks within values for display purposes only):

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

{
 "error": "invalid_client_metadata",
 "error_description": "The grant type ’authorization_code’ must be
   registered along with the response type ’code’ but found only
  ’implicit’ instead."
}
```

4. IANA Considerations

4.1. OAuth Dynamic Client Registration Metadata Registry

This specification establishes the "OAuth Dynamic Client Registration Metadata" registry.

OAuth registration client metadata names and descriptions 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 names prior to publication, the Designated Experts may approve registration once they are satisfied that such a specification will be published, per [RFC7120].

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register OAuth Dynamic Client Registration Metadata name: example").

Within the review period, the Designated Experts 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 Experts and should direct all requests for registration to the review mailing list.

4.1.1. Registration Template

Client Metadata 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.

Client Metadata Description:
: Brief description of the metadata value (e.g., "Example description").

Change Controller:
: For Standards Track RFCs, list "IESG". 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 or documents that specify the client metadata definition, preferably including a URI that can be used to retrieve a copy of the documents. An indication of the relevant sections may also be included but is not required.

4.1.2. Initial Registry Contents

The initial contents of the "OAuth Dynamic Client Registration Metadata" registry are:

- Client Metadata Name: "redirect_uris"
- Client Metadata Description: Array of redirection URIs for use in redirect-based flows
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "token_endpoint_auth_method"
- Client Metadata Description: Requested authentication method for the token endpoint
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "grant_types"
- Client Metadata Description: Array of OAuth 2.0 grant types that the client may use
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "response_types"
- Client Metadata Description: Array of the OAuth 2.0 response types that the client may use
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "client_name"
- Client Metadata Description: Human-readable name of the client to be presented to the user
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "client_uri"
- Client Metadata Description: URL of a web page providing information about the client
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "logo_uri"
- Client Metadata Description: URL that references a logo for the client
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "scope"
- Client Metadata Description: Space-separated list of OAuth 2.0 scope values
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "contacts"
- Client Metadata Description: Array of strings representing ways to contact people responsible for this client, typically email addresses
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "tos_uri"
- Client Metadata Description: URL that points to a human-readable terms of service document for the client
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "policy_uri"
- Client Metadata Description: URL that points to a human-readable policy document for the client
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "jwks_uri"
- Client Metadata Description: URL referencing the client's JSON Web Key Set [RFC7517] document representing the client's public keys
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "jwks"
- Client Metadata Description: Client's JSON Web Key Set [RFC7517] document representing the client's public keys
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "software_id"
- Client Metadata Description: Identifier for the software that comprises a client
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "software_version"
- Client Metadata Description: Version identifier for the software that comprises a client
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "client_id"
- Client Metadata Description: Client identifier
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "client_secret"
- Client Metadata Description: Client secret
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "client_id_issued_at"
- Client Metadata Description: Time at which the client identifier was issued
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Client Metadata Name: "client_secret_expires_at"
- Client Metadata Description: Time at which the client secret will expire
- Change Controller: IESG
- Specification Document(s): RFC 7591

4.2. 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" 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 Experts may approve registration once they are satisfied that such a specification will be published, per [RFC7120].

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 Experts 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 Experts and should direct all requests for registration to the review mailing list.

4.2.1. Registration Template

Token Endpoint Authentication 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, list "IESG". 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 or documents that specify the token endpoint authentication method, preferably including a URI that can be used to retrieve a copy of the document or documents. An indication of the relevant sections may also be included but is not required.

4.2.2. Initial Registry Contents

The initial contents of the "OAuth Token Endpoint Authentication Methods" registry are:

- Token Endpoint Authentication Method Name: "none"
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Token Endpoint Authentication Method Name: "client_secret_post"
- Change Controller: IESG
- Specification Document(s): RFC 7591

- Token Endpoint Authentication Method Name: "client_secret_basic"
- Change Controller: IESG
- Specification Document(s): RFC 7591

5. 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 [RFC5246] and MAY support additional transport-layer security mechanisms meeting its security requirements. When using TLS, the client MUST perform a TLS/SSL server certificate check, per RFC 6125 [RFC6125]. Implementation security considerations can be found in Recommendations for Secure Use of TLS and DTLS [BCP195].

For clients that use redirect-based grant types such as "authorization_code" and "implicit", authorization servers MUST require clients to register their redirection URI values. This can help mitigate attacks where rogue actors inject and impersonate a validly registered client and intercept its authorization code or tokens through an invalid redirection URI or open redirector. Additionally, in order to prevent hijacking of the return values of the redirection, registered redirection URI values MUST be one of:

- A remote web site protected by TLS (e.g., https://client.example.com/oauth_redirect)
- A web site hosted on the local machine using an HTTP URI (e.g., http://localhost:8080/oauth_redirect)
- A non-HTTP application-specific URL that is available only to the client application (e.g., exampleapp://oauth_redirect)

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 is possible that such clients would need to re-register every time the browser application is loaded. 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. Alternatively, such clients could be registered on the server where the in-browser application's code is served from, and the client's configuration could be pushed to the browser alongside the code.

Since different OAuth 2.0 grant types have different security and usage properties, 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 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.

Unless used as a claim in a software statement, the authorization server MUST treat all client metadata as self-asserted. For instance, a rogue client might use the name and logo of a legitimate client that 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 with instances of the legitimate client. To counteract this, an authorization server MUST take appropriate steps to mitigate this risk by looking at the entire registration request and client configuration. For instance, an authorization server could issue a warning if the domain/site of the logo doesn't match the domain/site of redirection URIs. An authorization server could also refuse registration requests from a known software identifier that is requesting different redirection URIs or a different client 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", enticing the user to click on it during the authorization. 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 URIs resolve to valid web pages. Since these URI values that are intended to be displayed to the user at the authorization page, the authorization server SHOULD protect the user from malicious content hosted at the URLs where possible. For instance, before presenting the URLs to the user at the authorization page, the authorization server could download the content hosted at the URLs, check the content against a malware scanner and blacklist filter, determine whether or not there is mixed secure and non-secure content at the URL, and other possible server-side mitigations. Note that the content in these URLs can change at any time and the authorization server cannot provide complete confidence in the safety of the URLs, but these practices could help. To further mitigate this kind of threat, the authorization server can also warn the user that the URL links have been provided by a third party, should be treated with

caution, and are not hosted by the authorization server itself. For instance, instead of providing the links directly in an HTML anchor, the authorization server can direct the user to an interstitial warning page before allowing the user to continue to the target URL.

Clients MAY use both the direct JSON object and the JWT-encoded software statement to present client metadata to the authorization server as part of the registration request. A software statement is cryptographically protected and represents claims made by the issuer of the statement, while the JSON object represents the self-asserted claims made by the client or developer directly. If the software statement is valid and signed by an acceptable authority (such as the software API publisher), the values of client metadata within the software statement MUST take precedence over those metadata values presented in the plain JSON object, which could have been intercepted and modified.

Like all metadata values, the software statement is an item that is self-asserted by the client, even though its contents have been digitally signed or MACed by the issuer of the software statement. As such, presentation of the software statement is not sufficient in most cases to fully identify a piece of client software. An initial access token, in contrast, does not necessarily contain information about a particular piece of client software but instead represents authorization to use the registration endpoint. An authorization server MUST consider the full registration request, including the software statement, initial access token, and JSON client metadata values, when deciding whether to honor a given registration request.

If an authorization server receives a registration request for a client that is not intended to have multiple instances registered simultaneously and the authorization server can infer a duplication of registration (e.g., it uses the same "software_id" and "software_version" values as another existing client), the server SHOULD treat the new registration as being suspect and reject the registration. It is possible that the new client is trying to impersonate the existing client in order to trick users into authorizing it, or that the original registration is no longer valid. The details of managing this situation are specific to the authorization server deployment and outside the scope of this specification.

Since a client identifier is a public value that can be used to impersonate a client at the authorization endpoint, an authorization server that decides to issue the same client identifier to multiple instances of a registered client needs to be very particular about the circumstances under which this occurs. For instance, the authorization server can limit a given client identifier to clients

using the same redirect-based flow and the same redirection URIs. An authorization server SHOULD NOT issue the same client secret to multiple instances of a registered client, even if they are issued the same client identifier, or else the client secret could be leaked, allowing malicious impostors to impersonate a confidential client.

6. Privacy Considerations

As the protocol described in this specification deals almost exclusively with information about software and not people, there are very few privacy concerns for its use. The notable exception is the "contacts" field as defined in Section 2, which contains contact information for the developers or other parties responsible for the client software. These values are intended to be displayed to end-users and will be available to the administrators of the authorization server. As such, the developer may wish to provide an email address or other contact information expressly dedicated to the purpose of supporting the client instead of using their personal or professional addresses. Alternatively, the developer may wish to provide a collective email address for the client to allow for continuing contact and support of the client software after the developer moves on and someone else takes over that responsibility.

In general, the metadata for a client, such as the client name and software identifier, are common across all instances of a piece of client software and therefore pose no privacy issues for end-users. Client identifiers, on the other hand, are often unique to a specific instance of a client. For clients such as web sites that are used by many users, there may not be significant privacy concerns regarding the client identifier, but for clients such as native applications that are installed on a single end-user’s device, the client identifier could be uniquely tracked during OAuth 2.0 transactions and its use tied to that single end-user. However, as the client software still needs to be authorized by a resource owner through an OAuth 2.0 authorization grant, this type of tracking can occur whether or not the client identifier is unique by correlating the authenticated resource owner with the requesting client identifier.

Note that clients are forbidden by this specification from creating their own client identifier. If the client were able to do so, an individual client instance could be tracked across multiple colluding authorization servers, leading to privacy and security issues. Additionally, client identifiers are generally issued uniquely per registration request, even for the same instance of software. In this way, an application could marginally improve privacy by registering multiple times and appearing to be completely separate

applications. However, this technique does incur significant usability cost in the form of requiring multiple authorizations per resource owner and is therefore unlikely to be used in practice.

7. References

7.1. Normative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015, <http://www.rfc-editor.org/info/bcp195>.

[IANA.Language] IANA, "Language Subtag Registry", <http://www.iana.org/assignments/language-subtag-registry>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <http://www.rfc-editor.org/info/rfc5646>.

[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, DOI 10.17487/RFC6125, March 2011, <http://www.rfc-editor.org/info/rfc6125>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <http://www.rfc-editor.org/info/rfc7120>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://www.rfc-editor.org/info/rfc7519>.

[RFC7522] Campbell, B., Mortimore, C., and M. Jones, "Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7522, DOI 10.17487/RFC7522, May 2015, <http://www.rfc-editor.org/info/rfc7522>.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <http://www.rfc-editor.org/info/rfc7523>.

7.2. Informative References

[OpenID.Discovery]
Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0", November 2014, <http://openid.net/specs/openid-connect-discovery-1_0.html>.

[OpenID.Registration]
Sakimura, N., Bradley, J., and M. Jones, "OpenID Connect Dynamic Client Registration 1.0", November 2014, <http://openid.net/specs/openid-connect-registration-1_0.html>.

[RFC7592] Richer, J., Jones, M., Bradley, J., and M. Machulak, "OAuth 2.0 Dynamic Client Registration Management Protocol", RFC 7592, DOI 10.17487/RFC7592, July 2015, <http://www.rfc-editor.org/info/rfc7592>.

[UMA-Core]
Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", Work in Progress, draft-hardjono-oauth-umacore-13, April 2015.

Appendix A. Use Cases

This appendix describes different ways that this specification can be utilized, including describing some of the choices that may need to be made. Some of the choices are independent and can be used in combination, whereas some of the choices are interrelated.

A.1. Open versus Protected Dynamic Client Registration

A.1.1. Open Dynamic Client Registration

Authorization servers that support open registration allow registrations to be made with no initial access token. This allows all client software to register with the authorization server.

A.1.2. Protected Dynamic Client Registration

Authorization servers that support protected registration require that an initial access token be used when making registration requests. While the method by which a client or developer 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 preregistration portal at the authorization server that issues an initial access token to the developer.

A.2. Registration without or with Software Statements

A.2.1. Registration without a Software Statement

When a software statement is not used in the registration request, the authorization server must be willing to use client metadata values without them being digitally signed or MACed (and thereby attested to) by any authority. (Note that this choice is independent of the Open versus Protected choice, and that an initial access token is another possible form of attestation.)

A.2.2. Registration with a Software Statement

A software statement can be used in a registration request to provide attestation by an authority for a set of client metadata values. This can be useful when the authorization server wants to restrict registration to client software attested to by a set of authorities or when it wants to know that multiple registration requests refer to the same piece of client software.

A.3. Registration by the Client or Developer

A.3.1. Registration by the Client

In some use cases, client software will dynamically register itself with an authorization server to obtain a client identifier and other information needed to interact with the authorization server. In this case, no client identifier for the authorization server is packaged with the client software.

A.3.2. Registration by the Developer

In some cases, the developer (or development software being used by the developer) will preregister the client software with the authorization server or a set of authorization servers. In this case, the client identifier value(s) for the authorization server(s) can be packaged with the client software.

A.4. Client ID per Client Instance or per Client Software

A.4.1. Client ID per Client Software Instance

In some cases, each deployed instance of a piece of client software will dynamically register and obtain distinct client identifier values. This can be advantageous, for instance, if the code flow is being used, as it also enables each client instance to have its own client secret. This can be useful for native clients, which cannot maintain the secrecy of a client secret value packaged with the software, but which may be able to maintain the secrecy of a per-instance client secret.

A.4.2. Client ID Shared among All Instances of Client Software

In some cases, each deployed instance of a piece of client software will share a common client identifier value. For instance, this is often the case for in-browser clients using the implicit flow, when no client secret is involved. Particular authorization servers might choose, for instance, to maintain a mapping between software statement values and client identifier values, and return the same client identifier value for all registration requests for a particular piece of software. The circumstances under which an authorization server would do so, and the specific software statement characteristics required in this case, are beyond the scope of this specification.

A.5. Stateful or Stateless Registration

A.5.1. Stateful Client Registration

In some cases, authorization servers will maintain state about registered clients, typically indexing this state using the client identifier value. This state would typically include the client metadata values associated with the client registration, and possibly other state specific to the authorization server's implementation. When stateful registration is used, operations to support retrieving and/or updating this state may be supported. One possible set of operations upon stateful registrations is described in [RFC7592].

A.5.2. Stateless Client Registration

In some cases, authorization servers will be implemented in a manner the enables them to not maintain any local state about registered clients. One means of doing this is to encode all the registration state in the returned client identifier value, and possibly encrypting the state to the authorization server to maintain the confidentiality and integrity of the state.

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 draft versions of this document: Amanda Anganes, Derek Atkins, Tim Bray, Domenico Catalano, Donald Coffin, Vladimir Dzhuvinov, George Fletcher, Thomas Hardjono, William Kim, Torsten Lodderstedt, Eve Maler, Josh Mandel, Nov Matake, Tony Nadalin, Nat Sakimura, Christian Scholz, and Hannes Tschofenig.

Authors' Addresses

Justin Richer (editor)

Email: ietf@justin.richer.org

Michael B. Jones
Microsoft

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

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com

Maciej Machulak
Newcastle University

Email: maciej.machulak@gmail.com

Phil Hunt
Oracle Corporation

Email: phil.hunt@yahoo.com