Internet Engineering Task Force (IETF)
Request for Comments: 7592
Category: Experimental
ISSN: 2070-1721

J. Richer, Ed.

M. Jones
Microsoft

J. Bradley
Ping Identity

M. Machulak
Newcastle University

July 2015

OAuth 2.0 Dynamic Client Registration Management Protocol

Abstract

This specification defines methods for management of OAuth 2.0 dynamic client registrations for use cases in which the properties of a registered client may need to be changed during the lifetime of the client. Not all authorization servers supporting dynamic client registration will support these management methods.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. 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). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see 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/rfc7592.

Table of Contents

1. Introduction

In order for an OAuth 2.0 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 with that server. "OAuth 2.0 Dynamic Client Registration Protocol" [RFC7591] describes how an OAuth 2.0 client can be dynamically registered with an authorization server to obtain this information and how metadata about the client can be registered with the server.

This specification extends the core registration specification by defining a set of methods for management of dynamic OAuth 2.0 client registrations beyond those defined in the core registration specification. In some situations, the registered metadata of a client can change over time, either by modification at the authorization server or by a change in the client software itself. This specification provides methods for the current registration state of a client to be queried at the authorization server, methods for the registration of a client to be updated at the authorization server, and methods for the client to be unregistered from the authorization server.

This Experimental RFC is intended to encourage development and deployment of interoperable solutions with the intent that feedback from this experience will inform a future standard.

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 the terms defined by "OAuth 2.0 Client Dynamic Registration Protocol" [RFC7591].

This specification defines the following terms:

Client Configuration Endpoint
: OAuth 2.0 endpoint through which registration information for a registered client can be managed. This URL for this endpoint is returned by the authorization server in the client information response.

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

1.3. Protocol Flow

This extends the flow in "OAuth 2.0 Dynamic Client Registration Protocol" [RFC7591] as follows:

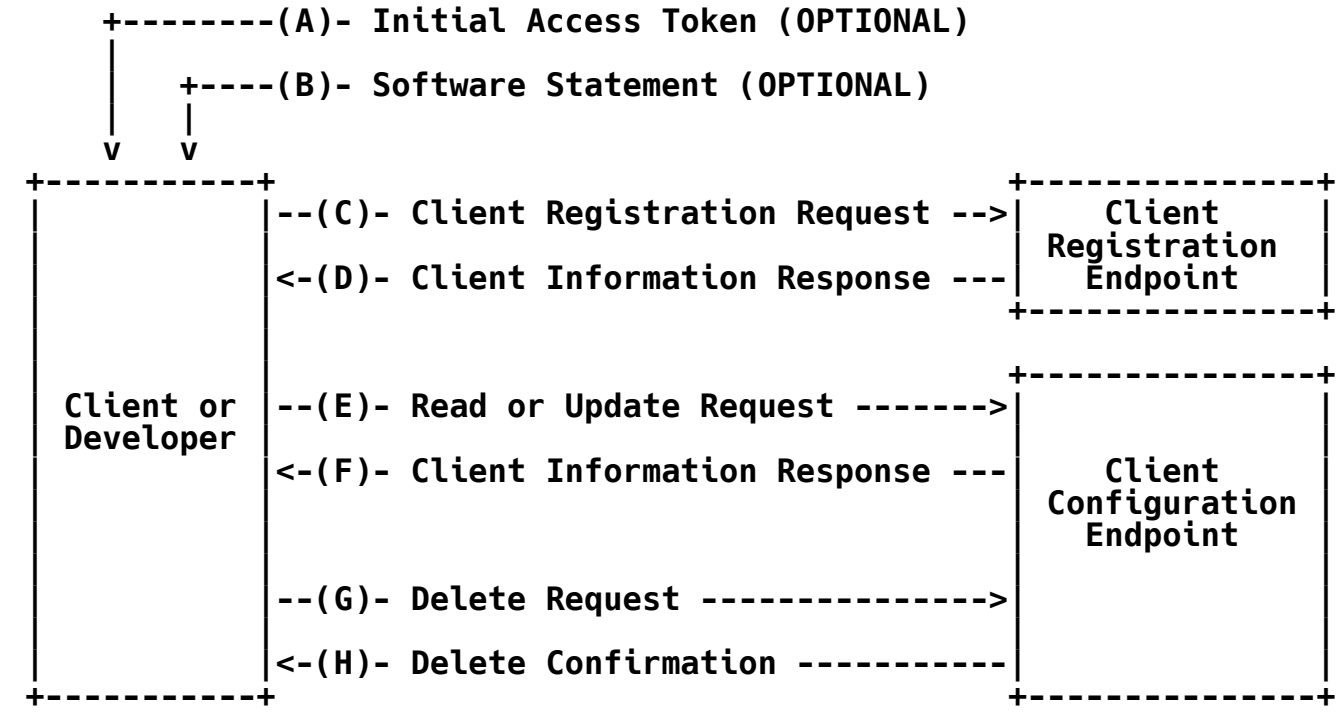

Figure 1: Abstract Extended Dynamic Client Registration Flow

The abstract OAuth 2.0 client dynamic registration flow illustrated in Figure 1 describes the interaction between the client or developer and the endpoints defined in this specification and its parent. This figure does not demonstrate error conditions. This flow includes the following steps:

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

(B) 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 its 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 to the server,

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

* a URI pointing to the client configuration endpoint, and

* a registration access token to be used when calling the client configuration endpoint.

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

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

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

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

2. Client Configuration Endpoint

The client configuration endpoint is an OAuth 2.0 protected resource that is provisioned by the server to facilitate viewing, updating, and deleting a client's registered information. The location of this

endpoint is communicated to the client through the "registration_client_uri" member of the client information response, as specified in Section 3. The client MUST use its registration access token in all calls to this endpoint as an OAuth 2.0 Bearer Token [RFC6750].

The client configuration endpoint MUST be protected by a transport-layer security mechanism, as described in Section 5.

Operations on this endpoint are switched through the use of different HTTP methods [RFC7231]. If an authorization server does not support a particular method on the client configuration endpoint, it MUST respond with the appropriate error code.

2.1. Client Read Request

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

The following is a non-normative example request:

```
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 Section 3. Some values in the response, including the "client_secret" and "registration_access_token", MAY be different from those in the initial registration response. If the authorization server includes a new client secret and/or registration access token in its response, the client MUST immediately discard its previous client secret and/or registration access token. The value of the "client_id" MUST NOT change from the initial registration response.

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

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

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

2.2. Client Update Request

To update a previously registered client's registration with an authorization server, the client makes an HTTP PUT request to the client configuration endpoint with a content type of "application/json". The HTTP entity payload is a JSON [RFC7159] document consisting of a JSON object and all parameters as top-level members of that JSON object. This request is authenticated by the registration access token issued to the client.

This request MUST include all client metadata fields as returned to the client from a previous registration, read, or update operation. The updated client metadata fields request MUST NOT include the "registration_access_token", "registration_client_uri", "client_secret_expires_at", or "client_id_issued_at" fields described in Section 3.

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, indicating the client's request to delete them from the client's registration. The authorization server MAY ignore any null or empty value in the request just as any other value.

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.

The following is a non-normative example request:

```
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"],
 "grant_types": ["authorization_code", "refresh_token"],
 "token_endpoint_auth_method": "client_secret_basic",
 "jwks_uri": "https://client.example.org/my_public_keys.jwks",
 "client_name": "My New Example",
 "client_name#fr": "Mon Nouvel Exemple",
 "logo_uri": "https://client.example.org/newlogo.png",
 "logo_uri#fr": "https://client.example.org/fr/newlogo.png"
}
```

This example uses client metadata values defined in [RFC7591].

Upon successful update, the authorization server responds with an HTTP 200 OK message with content type "application/json" and a payload as described in Section 3. Some values in the response, including the "client_secret" and "registration_access_token", MAY be different from those in the initial registration response. If the authorization server includes a new client secret and/or registration access token in its response, the client MUST immediately discard its previous client secret and/or registration access token. The value of the "client_id" MUST NOT change from the initial registration response.

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

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

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

If the client attempts to set an invalid metadata field and the authorization server does not set a default value, the authorization server responds with an error as described in [RFC7591].

2.3. Client Delete Request

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

The following is a non-normative example request:

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

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

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

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

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

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

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

The following is a non-normative example response:

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

3. Client Information Response

This specification extends the client information response defined in "OAuth 2.0 Client Dynamic Registration" [RFC7591], which states that the response contains the client identifier (as well as the client secret if the client is a confidential client). When used with this specification, the client information response also contains the fully qualified URL of the client configuration endpoint (Section 2) for this specific client that the client or developer may use to manage the client's registration configuration, as well as a registration access token that is to be used by the client or developer to perform subsequent operations at the client configuration endpoint.

registration_client_uri
: REQUIRED. String containing the fully qualified URL of the client configuration endpoint for this client.

registration_access_token
: REQUIRED. String containing the access token to be used at the client configuration endpoint to perform subsequent operations upon the client registration.

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 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 [RFC7159].

The following is a non-normative example response:

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

{
 "registration_access_token": "reg-23410913-abewfq.123483",
 "registration_client_uri":
    "https://server.example.com/register/s6BhdRkqt3",
 "client_id": "s6BhdRkqt3",
 "client_secret": "cf136dc3c1fc93f31185e5885805d",
 "client_id_issued_at": 2893256800,
 "client_secret_expires_at": 2893276800,
 "client_name": "My Example Client",
 "client_name#ja-Jpan-JP":
    "\u30AF\u30E9\u30A4\u30A2\u30F3\u30C8\u540D",
 "redirect_uris": [
   "https://client.example.org/callback",
   "https://client.example.org/callback2"],
 "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"
}
```

4. IANA Considerations

This specification registers the following client metadata names and descriptions in the "OAuth Dynamic Client Registration Metadata" registry established by [RFC7591]:

- Client Metadata Name: "registration_access_token"
- Client Metadata Description: OAuth 2.0 Bearer Token used to access the client configuration endpoint
- Change Controller: IESG
- Specification Document(s): RFC 7592
- Client Metadata Name: "registration_client_uri"
- Client Metadata Description: Fully qualified URI of the client registration endpoint
- Change Controller: IESG

- Specification Document(s): RFC 7592

5. Security Considerations

While the client secret can expire, the registration access token SHOULD NOT expire while a client is still actively registered. If this token were to expire, a developer or client could be left in a situation where they have no means of retrieving, updating, or deleting the client's registration information. Were that the case, a new registration would be required, thereby generating a new client identifier. However, to limit the exposure surface of the registration access token, the registration access token MAY be rotated when the developer or client does a read or update operation on the client's client configuration endpoint. As the registration access tokens are relatively long-term credentials, and since the registration access token is a Bearer Token and acts as the sole authentication for use at the client configuration endpoint, it MUST be protected by the developer or client as described in the OAuth 2.0 Bearer Token Usage specification [RFC6750].

Since requests to the client configuration 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 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].

Since possession of the registration access token authorizes the holder to potentially read, modify, or delete a client's registration (including its credentials such as a client_secret), the registration access token MUST contain sufficient entropy to prevent a random guessing attack of this token, such as described in Section 5.2 of [RFC6750] and Section 5.1.4.2.2 of [RFC6819].

If a client is deprovisioned from a server, any outstanding registration access token for that client MUST be invalidated at the same time. Otherwise, this can lead to an inconsistent state wherein a client could make requests to the client configuration endpoint where the authentication would succeed but the action would fail because the client is no longer valid. The authorization server MUST treat all such requests as if the registration access token was invalid by returning an HTTP 401 Unauthorized error, as described.

6. Privacy Considerations

This specification poses no additional privacy considerations beyond those described in the core "OAuth 2.0 Dynamic Client Registration Protocol" [RFC7591].

7. 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>.

[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>.

[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>.

[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>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <http://www.rfc-editor.org/info/rfc6819>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <http://www.rfc-editor.org/info/rfc7591>.

Appendix A. Registration Tokens and Client Credentials

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

- The initial access token is optionally used by the client or developer at the registration endpoint. This is an OAuth 2.0 token that is used to authorize the initial client registration request. The content, structure, generation, and validation of this token are out of scope for this specification. The authorization server can use this token to verify that the presenter is allowed to dynamically register new clients. This token may be shared among multiple instances of a client to allow them to each register separately, thereby letting the authorization server use this token to tie multiple instances of registered clients (each with their own distinct client identifier) back to the party to whom the initial access token was issued, usually an application developer. This token is usually intended to be used only at the client registration endpoint.

- The registration access token is used by the client or developer at the client configuration endpoint and represents the holder's authorization to manage the registration of a client. This is an OAuth 2.0 Bearer Token that is issued from the client registration endpoint in response to a client registration request and is returned in a client information response. The registration access token is uniquely bound to the client identifier and is required to be presented with all calls to the client configuration endpoint. The registration access token should be protected as described in [RFC6750] and should not be shared between instances of a client. If a registration access token is shared between client instances, one instance could change or delete registration values for all other instances of the client. The registration access token can be rotated through the use of the client read or update method on the client configuration endpoint. The registration access token is intended to be used only at the client configuration endpoint.

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

A.1. Credential Rotation

The authorization server may be configured to issue new registration access tokens and/or client credentials (such as a "client_secret") throughout the lifetime of the client. This may help minimize the impact of exposed credentials. The authorization server conveys new registration access tokens and client credentials (if applicable) to the client in the client information response of either a read or update request to the client configuration endpoint. The client's current registration access token and client credentials (if applicable) MUST be included in the client information response.

The registration access token SHOULD be rotated only in response to a read or update request to the client configuration endpoint. At this point, the new registration access token is returned to the client, the old registration access token MUST be discarded by the client, and it SHOULD be discarded by the server, if possible. If, instead, the registration access token were to expire or be invalidated outside of such requests, the client or developer might be locked out of managing the client's configuration.

Note that the authorization server decides the frequency of the credential rotation and not the client. Methods by which the client can request credential rotation are outside the scope of this document.

Appendix B. 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, as specified in Section 3. 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 that combines the client registration endpoint's URL and the issued "client_id" for this client, with the latter as either a path parameter or a query parameter. For example, a client with the client identifier "s6BhdRkqt3" could be given a client configuration endpoint URL of "https://server.example.com/register/s6BhdRkqt3" (path parameter) or of "https://server.example.com/register?client_id=s6BhdRkqt3" (query parameter). In both of these cases, the client simply uses the URL as given by the authorization server.

These common patterns can help the server to more easily determine the client to which the request pertains, which MUST be matched against the client to which the registration access token was issued. If desired, the server MAY simply return the client registration endpoint URL as the client configuration endpoint URL and change behavior based on the authentication context provided by the registration access token.

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, Phil Hunt, 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