Authorization Framework for Network APIs

Approved Version 1.0 – 09 Dec 2014

Open Mobile Alliance

OMA-ER-Autho4API-V1_0-20141209-A

Contents

Figures

Tables

1. Scope

The Authorization Framework for Network APIs enables a Resource Owner owning network resources exposed by Network APIs and RESTful APIs in particular, to authorize third-party Applications (desktop, mobile and web Applications)to access these resources via that API on the Resource Owner's behalf.

2. References

2.1 Normative References

[AES] "The specification of Advanced Encryption Standard", NIST FIPS Publications, 197, URL:http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[AESMode] "Recommendation of Block Cipher Modes of Operation", NIST, NIST Special Publication 800-38A, URL:http://www.nist.gov/

[AUTHO4API_RD_10] "Authorization Framework for Network APIs Requirements", Version 1.0, Open Mobile Alliance™

[HTML_4.01] "HTML 4.01 Specification", W3C Recommendation 24 December 1999, URL:http://www.w3.org/TR/html401/

[OMNA_Autho4API] Open Mobile Naming Authority "Autho4API Scope Values Registry", Open Mobile Alliance™, URL:http://www.openmobilealliance.org/tech/omna

NOTE: The hyperlink above will point directly to the OMNA registry page once available.

[OSE] "OMA Service Environment", Open Mobile Alliance™, URL:http://www.openmobilealliance.org/

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed, N. Borenstein, November 1996, URL:http://www.ietf.org/rfc/rfc2045.txt

[RFC2046] "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", N. Freed, N. Borenstein, November 1996, URL:http://www.ietf.org/rfc/rfc2046.txt

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt

[RFC2616] "Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. Mogu, H. Frystyk, L, Masinter, P. Leach, T. Berners-Lee, June 1999, URL:http://www.ietf.org/rfc/rfc2616.txt

[RFC2617] "HTTP Authentication: Basic and Digest Access Authentication", June 1999, URL:http://www.ietf.org/rfc/rfc2617.txt

[RFC3986] "Uniform Resource Identifier (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L.Masinter, January 2005, URL:http://www.ietf.org/rfc/rfc3986.txt

[RFC4234] "Augmented BNF for Syntax Specifications: ABNF". D. Crocker, Ed., P. Overell. October 2005, URL:http://www.ietf.org/rfc/rfc4234.txt

[RFC4346] "Transport Layer Security (TLS) Version 1.1", T. Dierks, E. Rescorla, IETF RFC 4346, April 2006, URL:http://www.ietf.org/rfc/rfc4346.txt

[RFC5246] "The Transport Layer Security (TLS) Protocol Version 1.2", T. Dierks, E. Rescorla, August 2008, URL:http://www.ietf.org/rfc/rfc5246.txt

[RFC6749] "The OAuth 2.0 Authorization Framework", D. Hardt, October 2012, URL:http://www.ietf.org/rfc/rfc6749.txt

[RFC6750] "The OAuth 2.0 Authorization Framework: Bearer Token Usage", M. Jones, D. Hardt, October 2012, URL:http://www.ietf.org/rfc/rfc6750.txt

[RFC7009] "OAuth 2.0 Token Revocation", T. Lodderstedt, S. Dronia, M. Scurtescu, August 2013, URL:http://www.ietf.org/rfc/rfc7009.txt

[SCRRULES] "SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL:http://www.openmobilealliance.org/

[SEC_CF-V1_1] "Security Common Functions ", Version 1.1, Open Mobile Alliance™, URL:http://www.openmobilealliance.org/

2.2 Informative References

[AUTHO4API _SUP_Resource_Server]	"XML schema for Autho4API for the URL Prefixes for Granted Resources", Open Mobile Alliance™, OMA-SUP-XSD_rest_autho4api_url_prefixes_for_granted_resources-V1.0, URL:http://www.openmobilealliance.org/
[OMADICT]	"Dictionary for OMA Specifications", Version x.y, Open Mobile Alliance™, OMA-ORG-Dictionary-Vx_y, URL:http://www.openmobilealliance.org/
[OMAPUSH]	"OMA Push", Version 2.3, Open Mobile Alliance™, URL:http://www.openmobilealliance.org/
[REST_NetAPI_Commo n]	"Common definitions for OMA RESTful Network APIs", Open Mobile Alliance™, OMA-TS-REST_NetAPI_Common-V1_0, URL:http://www.openmobilealliance.org/
[WAC2.1]	"WAC Specifications 2.1", Wholesale Applications Community, URL:http://www.wacapps.net/
[XMLSchema1]	W3C Recommendation, XML Schema Part 1: Structures Second Edition, URL:http://www.w3.org/TR/xmlschema-1/
[XMLSchema2]	C Recommendation, XML Schema Part 2: Data types Second Edition, URL:http://www.w3.org/TR/xmlschema-2/

3. Terminology and Conventions

3.1 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].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

For the purpose of this specification, all definitions from the OMA Dictionary apply [OMADICT].

Access Token	Credentials used to access protected resources, further defined in section 1.4 of [RFC6749].
Authorization Grant	Credential representing the Resource Owner's authorization (to access its protected resources) used by the Autho4API Client to obtain an Access Token.
Refresh Token	Credentials used to obtain new Access Tokens, further defined in section 1.5 of [RFC6749].
Resource Owner	An entity capable of granting access to a protected resource (e.g. end-user).
Scope	The total resource access privileges of a given authorization process, either requested by Autho4API Client or issued to Autho4API Client in the form of an Access Token. It is expressed as a list of Scope Values.
Scope Request Access Token	An Access Token authorizing the AuthoAPI Client to request the scope value required to access a resource.
Scope Value	A well-defined set of authorized operations on Network API resources.

3.3 Abbreviations

AES	Advanced Encryption Standard
API	Application Program Interface
Autho4API	Authorization Framework for Network APIs
CBC	Cipher Block Chaining
GSMA	Global System for Mobile communications Association
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
MSISDN	Mobile Subscriber ISDN Number
OMA	Open Mobile Alliance
OMNA	Open Mobile Naming Authority
OS	Operating System
SMS	Short Message Service
TLS	Transport Layer Security
URI	Unified Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
WAC	Wholesale Applications Community
XML	eXtensible Markup Language

4. Introduction

OMA RESTful Network APIs may be complemented with a common delegated authorization framework based on IETF OAuth 2.0, for access of third party Applications via those APIs.

The delegated authorization framework will enable a Resource Owner owning network resources exposed by an OMA RESTful API, to authorize third-party Applications (desktop, mobile and web Applications) to access these resources via that API on the Resource Owner's behalf.

4.1 Version 1.0

Autho4API is based on IETF OAuth 2.0 specifications ([RFC6749], [RFC6750]and [RFC7009]) and in addition defines the following:

- OMNA registry for Scope Values
- Secondary channel, i.e. alternative to HTTP redirection for the delivery of response to Authorization Request
- Deployment scenarios for environments where multiple Service Providers expose the same service
- Resolution of resource server location from an issued Access Token
- One-time Access Tokens
- Considerations on:
  - Scope Value definitions
  - self-contained Access Token formats
  - service discovery
  - native Applications
  - HTTP redirect capture mechanisms
- Scope requests

4.2 Abstract protocol flow

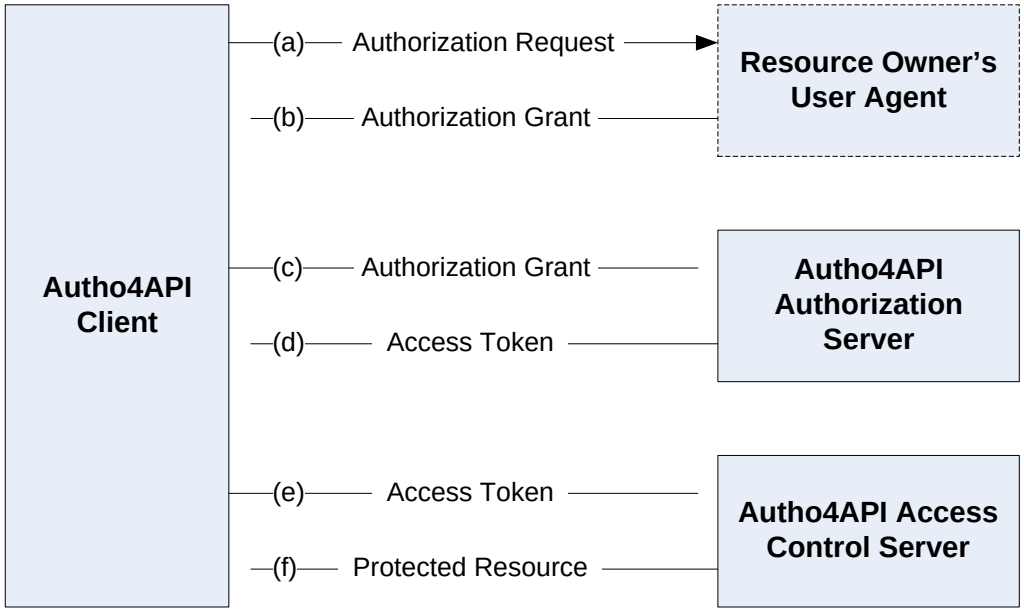

Figure 1: Abstract protocol flow

a) The Autho4API Client requests authorization from the Resource Owner.

b) The Autho4API Client receives an Authorization Grant which is a credential representing the Resource Owner's authorization.

c) The Autho4API Client requests an Access Token by authenticating with the Autho4API Authorization Server and presenting the Authorization Grant.

d) The Autho4API Authorization Server authenticates the Autho4API Client and validates the Authorization Grant, and if valid issues an Access Token.

e) The Autho4API Client requests the protected resource from the Autho4API Access Control Server and authenticates by presenting the Access Token.

f) The Autho4API Access Control Server validates the Access Token, and if valid, serves the request.

5. Requirements (Normative)

Requirements applicable to this combined release are defined in [AUTHO4API_RD_10].

6. Architectural Model

6.1 Dependencies

The Autho4API v1.0 enabler depends on other OMA Enablers and external entities, including the following:

- IETF OAuth 2.0 Authorization Framework [RFC6749] which provides mandatory support of OAuth 2.0 Access Token obtention
- IETF OAuth 2.0 Authorization Framework: Bearer Token Usage [RFC6750] which provides mandatory support of OAuth 2.0 Bearer Access Token usage
- IETF Token Revocation [RFC7009] which provides optional support of OAuth 2.0 Access Token and Refresh Token revocation
- OMA Push [OMAPUSH] which provides optional support for the delivery of response to Authorization Request over the secondary channel "OMA Connectionless Push over SMS"

6.2 Architectural Diagram

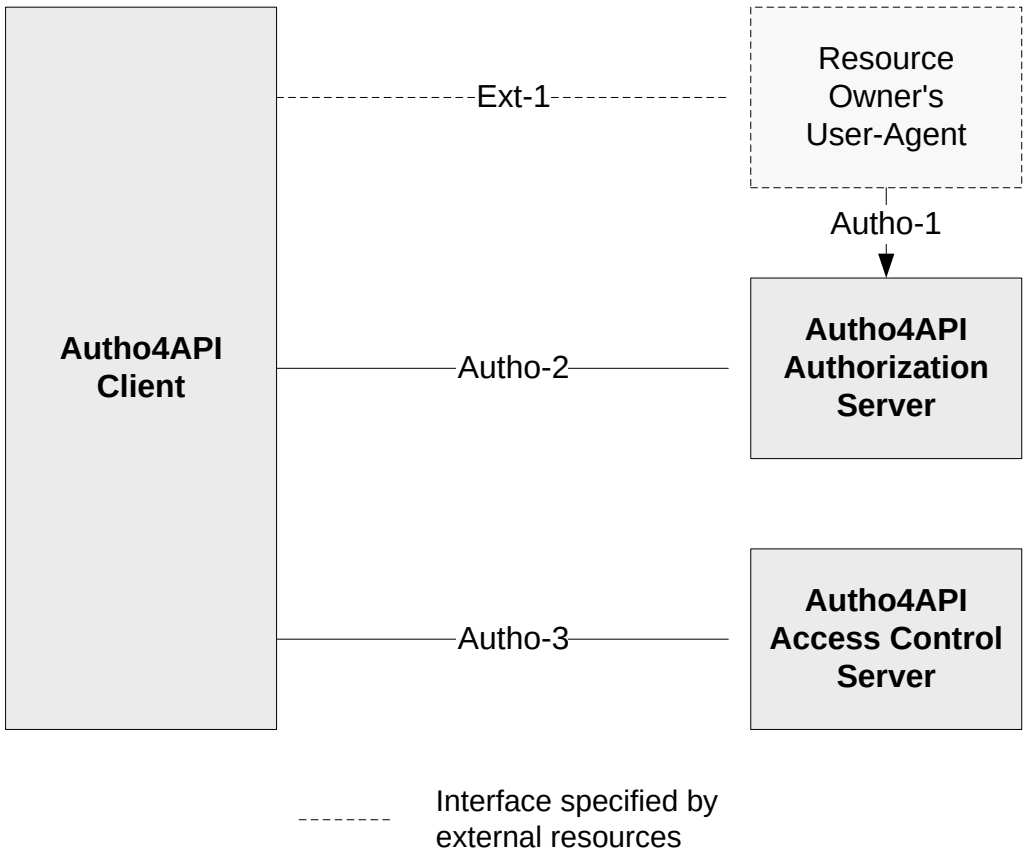

Figure 2: Architectural Diagram

6.3 Functional Components and Interfaces/reference points definition

6.3.1 Functional components

6.3.1.1 Internal functional components

6.3.1.1.1 Autho4API Authorization Server

The Autho4API Authorization Server is responsible for issuing, verifying and revoking the Access Tokens enabling an Autho4API Client to access to some Resource Owner's network resources on behalf of this Resource Owner.

The Autho4API Authorization Server is responsible for issuing, verifying and revoking the Scope Request Access Tokens enabling an Autho4API Client to request the scope required for accessing a specific Resource Owner's network resource.

The Autho4API Authorization Server must be able to determine the validity of an Access Token.

The Autho4API Authorization Server must be aware of the Resource Owner's authentication which must take place prior to the authorization process.

When required, the Autho4API Authorization Server must authenticate the Autho4API Client prior to Access Token issuing or revoking.

Note: the Autho4API Authorization Server can authenticate the Resource Owner prior to the authorization process, using mechanisms out of the scope of this enabler.

When required (e.g. in specific shared multi-Service Provider environments), this component must be able to route Authorization Requests and Access Token Requests to the actual Service Provider-specific Autho4API Authorization Server. When required, in order to later perform such routing to the actual Service Provider-specific Autho4API Authorization Server, this component must be able to cache Authorization Codes and Access Tokens, and its association with a Service Provider-specific Autho4API Authorization Server. Additionally, it must be able to indicate a redirection URI to the actual Service Provider-specific Autho4API Authorization Server, in order to redirect the Resource Owner's User Agent back to the component and be able to cache the Authorization Code or the Access Token.

The Autho4API Authorization Server must be able to establish and use secured channel with Autho4API Client to protect confidentiality of some key information exchanged between them (e.g. Access Token, client secret) and resist replay attacks.

6.3.1.1.2 Autho4API Access Control Server

The Autho4API Access Control Server is responsible for controlling access to the exposed Resource Owner's protected resources, based on the validity of the Access Token included in protected resource request. The Autho4API Access Control Server checks this validity, e.g. by collaborating in the back-end with the Autho4API Authorization Server. The mechanisms used for this collaboration are out of the scope of this enabler.

The Autho4API Access Control Server is responsible for providing the scope required for accessing a specific Resource Owner's network resource.

When required (e.g. in specific shared multi-Service Provider environments), this component must be able to route resource requests to the actual Service Provider-specific Access Control Server.

The Autho4API Access Control Server must be able to establish and use secured channel with Autho4API Client to protect confidentiality of some key information exchanged between them (e.g. Access Token) and resist replay attacks.

6.3.1.1.3 Autho4API Client

The Autho4API Client is responsible for:

- obtaining the Access Tokens enabling access to some Resource Owner's protected network resources, on Resource Owner's behalf;

- obtaining the Scope Request Access Tokens allowing to request the scope required for accessing a specific Resource Owner's network resource;
- accessing the Resource Owner's protected resources using the obtained Access Tokens, when required;
- requesting the scope required for accessing a specific Resource Owner's network resource;
- establishing and using secured channel with Autho4API Authorization Server and Autho4API Access Control Server to protect confidentiality of some key information exchanged between them (e.g. Access Token, client secret) and resist replay attacks.

6.3.1.2 External functional components (Informative)

6.3.1.2.1 Resource Owner's User Agent

For the flows where explicit Resource Owner's authorization is required, the Resource Owner's User Agent acts as a proxy for the Autho4API Client to direct the Resource Owner to the Autho4API Authorization Server where can take place Resource Owner authentication and Autho4API Client authorization. The Resource Owner's User Agent component is typically involved in these steps of authentication and authorization.

The Resource Owner's User Agent is able to transmit the Authorization Grant or Access Token to Autho4API Client, when issued by Autho4API Authorization Server through Autho-1.

6.3.2 Interfaces

6.3.2.1 Autho-1

This interface is exposed by the Autho4API Authorization Server and can be used to obtain from the Resource Owner using a User Agent, the authorization to access on his behalf some of his protected network resources. The obtained authorization is returned in a form of an Authorization Grant or an Access Token.

Autho-1 can also be used to authenticate the Resource Owner prior to the authorization process. The mechanisms used for this authentication are out of the scope of this enabler.

Autho-1 can also be used to present to the Resource Owner information about the requestor, the requested resources and the requested operations on these resources.

Note: this interface maps to the OAuth 2.0 authorization endpoint defined in [RFC6749].

6.3.2.2 Autho-2

This interface is exposed by the Autho4API Authorization Server and can be used to exchange Authorization Grants for Access Tokens (including Scope Request Access Tokens) and optionally Refresh Tokens, or to exchange Refresh Tokens for Access Tokens and optionally Refresh Tokens. Note: In this role, the interface then maps to the OAuth 2.0 token endpoint defined in [RFC6749].

Autho-2 can also be used to revoke Access Tokens (including Scope Request Access Tokens) and Refresh Tokens. In this role, the interface then maps to the OAuth 2.0 token revocation endpoint defined in [RFC7009].

Autho-2 can also be used to obtain the URL prefixes of the resources for which access is granted by a specific Access Token.

Autho-2 must support to resist replay attacks and protect confidentiality of the information exchanged between Autho4API Client and Autho4API Authorization Server (e.g. Access Token, client secret).

6.3.2.3 Autho-3

This interface is exposed by the Autho4API Access Control Server and can be used to access Resource Owner's protected resources on behalf of the Resource Owner, using Access Tokens as a proof of authorization.

Autho-3 can also be used to request the scope needed to access a protected resource, using Scope Request Access Tokens as a proof of authorization.

Autho-3 must support to resist replay attacks and protect confidentiality of the information exchanged between Autho4API Client and Autho4API Access Control Server (e.g. Access Token).

7. Technical Specification

7.1 Client Registration

7.1.1 Client Types

An Autho4API Client SHALL be either of confidential type (e.g., web Application executing on a web server) or of public type (e.g., native Application), which types are described in section 2.1 of [RFC6749].

An Autho4API Client SHALL be registered in Autho4API Authorization Server to obtain a unique client identifier issued according to the provided client registration information (e.g., Client Name, Redirection Endpoints). Moreover, an Autho4API Client of confidential type SHALL obtain client credentials (e.g. client secret) from Autho4API Authorization Server to be used for client authentication.

Security consideration for native Application is out of scope of Autho4API 1.0.

7.1.2 Client Authentication

Autho4API Clients of confidential type and in possession of a client password MAY use the HTTP Basic authentication scheme to authenticate with the Autho4API Authorization Server, as defined in section 2.3.1 of [RFC6749].

The Autho4API Authorization Server MAY support any other suitable HTTP authentication scheme matching its security requirements. These authentication mechanisms are beyond the scope of Autho4API v1.0.

7.1.3 Unregistered Clients

This enabler does not exclude the use of unregistered clients as defined in the section 2.4 of [RFC6749]. However, the use of such clients is beyond the scope of Autho4API v1.0.

7.2 Protocol Endpoints

7.2.1 Authorization Endpoint

The authorization endpoint described in the section 3.1 of [RFC6749] embedded in the Autho4API Authorization Server is used to interact with the Resource Owner and obtain an Authorization Grant through the interface Autho-1.

The Autho4API Authorization Server can authenticate the Resource Owner prior to the authorization process, using mechanisms out of the scope of this enabler.

The Autho4API Authorization Server SHALL support TLS 1.1 [RFC4346] and SHOULD support TLS 1.2 [RFC5246].

In order to secure a secondary channel used to transport Authorization Grant from authorization endpoint to the Autho4API Client (as defined in section 7.4), additional transport layer mechanisms and security may be taken into account.

7.2.1.1 Redirection Endpoint

As described in section 3.1.2 of [RFC6749], Autho4API Authorization Server redirects the Resource Owner's user-agent back to the Autho4API Client’s Redirection Endpoint, i.e. the absolute URI which was registered in the Autho4API Authorization Server during the client registration process.

Autho4API Clients able to receive HTTP redirection responses (i.e. for which the Redirection Endpoint is a public HTTP URL):

- SHOULD require the use of TLS 1.1 [RFC4346] or TLS 1.2 [RFC5246] on their Redirection Endpoint.
- SHALL support the HTTP status codes defined in the 3xx range (which includes in particular - but not only - the “302 Found” code).

In order to support native Application client, Autho4API Authorization Server may support non-absolute URIs and also support other alternative channels for Autho4API Client to receive Authorization Code as described in section 7.5.6 of this specification.

7.2.1.2 Endpoint extensions

In the Authorization Request sent over the Autho-1 interface, the Autho4API enabler can support authorization endpoint extension parameters, which can be in particular:

- The specific encoding of 'redirect_uri' parameter to request response delivery over a secondary channel as defined in section 7.5.7.

7.2.2 Token Endpoint

The Token Endpoint described in the section 3.2 of [RFC6749] embedded in the Autho4API Authorization Server is used to interact with the Autho4API Client and obtain an Access Token (eventually being a Scope Request Access Token) through the interface Autho-2.

When required, the Autho4API Authorization Server SHALL authenticate the Autho4API Client and SHALL validate the Authorization Grant type prior to Access Token issuing.

In order to secure the transport of client credentials, Authorization Grant and Access Token, the Autho4API Authorization Server SHALL support TLS 1.1 [RFC4346] and SHOULD support TLS 1.2 [RFC5246].

7.2.2.1 Grant Type

Regarding the Authorization Grants described in section 1.3 of [RFC6749]:

- the Autho4API Client SHALL support at least one of Authorization Code, Implicit, Resource Owner Password Credentials and Client Credentials grant types;
- the Autho4API Authorization Server SHALL support at least Authorization Code and Implicit grant types and MAY support Resource Owner Password Credentials and Client Credentials grant types.

7.2.3 Token Revocation Endpoint

Autho4API Clients and Autho4API Authorization Servers MAY support the token revocation functionality defined in [RFC7009].

When supported, the Token Revocation Endpoint described in [RFC7009] and exposed by the Autho4API Authorization Server on Autho-2 MAY be used by the Autho4API Client to revoke an Access Token (eventually being a Scope Request Access Token) or a Refresh Token. In addition:

- If the Autho4API Client is of confidential type, the Autho4API Authorization Server SHALL authenticate the Autho4API Client and SHALL verify whether the client is authorized to revoke the particular token prior to the token revocation.
- If the Autho4API Client is of public type, the Autho4API Authorization Server SHALL identify the Autho4API Client and SHALL verify whether the client is authorized to revoke the particular token prior to the token revocation.
- In order to provide transport layer security, the Autho4API Authorization Server SHALL support TLS 1.1 [RFC4346] and SHOULD support TLS 1.2 [RFC5246].

7.3 Service Parameters Discovery (Informative)

OAuth 2.0 [RFC6749] does not define how an Autho4API Client can obtain the service parameters required to run the protocol:

- Server endpoints URLs;
- Server capabilities, i.e. supported protocol options;

- Resource locations;
- Scope Values and their mapping with resource access.

When these parameters are fixed and described in the service documentation of the network API, the Autho4API Client can statically embed these parameters prior to its deployment. The Autho4API Client can also dynamically refine fixed Scope Values, using a mechanism defined in Section 7.7.

Otherwise, the Autho4API Client needs to dynamically obtain the unknown parameters using some mechanism, not only to proper run the protocol, but also to determine in advance - for the sake of user experience - if it is able to run the protocol before engaging the Resource Owner in the authorization process.

7.3.1 Service Parameters

This section describes the parameters which the Autho4API Client must obtain somehow to properly run the protocol flow till the protected resource request.

7.3.1.1 Bootstrapping

An Autho4API Client not knowing at the time of deployment all the parameters required to use the Authorization Framework needs at least to be provisioned with some information allowing to obtain these parameters later on, such as:

- The location of a service discovery endpoint.

7.3.1.2 Scope Values (Normative)

7.3.1.2.1 Definitions

An individual Network API specification using this enabler as its authorization framework may define Scope Values.

Appendix D provides considerations on the definition of Scope Values for a given Network API.

7.3.1.2.2 Naming and registration

The Open Mobile Naming Authority maintains a registry [OMNA_Autho4API] of Autho4API Scope Values applicable to the Network APIs which use the Autho4API enabler as their authorization framework.

Scope Value names in this registry SHALL be case-sensitive string values conforming:

- firstly to the "scope-token" element specified in section 3.3 of [RFC6749], which defines in particular the allowed character set for scope values
- secondly to the following grammar to assure uniqueness:

```
{ScopeValue}             ::= { {OMAScopeValue} | {ExtOrgScopeValue} |
                               {UnregisteredScopeValue} }
{OMAScopeValue}          ::= oma_{ApiType}_{ApiIdentification}.{Token}
{OMASubScopeValue}       ::= {OMAScopeValue}_{Subscope}
{ExtOrgScopeValue}       ::= {ExtOrgPrefix}_{Label}
{UnregisteredScopeValue} ::= x_{Label}
```

Where:

Name	Description
ApiType	Type of the Network API. In string format excluding "_" character

ApiIdentification	Identification of the Network API, in the scope of `{ApiType}`. In string format excluding "_"and "." characters.
Token	Identification of a set of operations on a set of resources of the API identified by `{ApiType}_{ApiIdentification}`. In string format excluding "_" character.
Subscope	OMA Scope Value suffix, containing the "subscope" string when used to request a Scope Request Access Token, or else a subscope identifier originated by the Autho4API Access Control Server. In string format excluding "_"and "." characters.
ExtOrgPrefix	Prefix defined by an external organization, RECOMMENDED to identify the organization itself. In string format excluding "_" character.
Label	Identification of a set of operations on a set of resources of some Network API(s) in the scope of the Scope Value prefix. In string format.

Autho4API Scope Values defined by OMA specifications SHALL follow the `{OMAScopeValue}` or `{OMASubScopeValue}` grammar. Those following the `{OMAScopeValue}` grammar SHALL be registered with the OMNA Autho4API Scope Value Registry.

External organizations defining Autho4API Scope Values are RECOMMENDED to register a Scope Value prefix with the OMNA Autho4API Scope Value Registry and suitably publish their Scope Value definitions in a publicly available specification.

Scope Value prefix "x_" MAY be used to freely define Autho4API Scope Values which are not intended to be registered with OMNA. As such, these Scope Values are not guaranteed to be collision-free when used on Autho4API endpoints.

7.3.1.3 Autho4API Authorization Server parameters

To properly interact with the Autho4API Authorization Server on Autho-1 and Autho-2 interfaces, the Autho4API Client needs to know the following server-specific parameters:

For a protocol flow which includes an Authorization Request:

- Location of authorization endpoint;
- Supported/required authorization endpoint extension parameters;
- Supported secondary channel(s) for the delivery of response to Authorization Request, with supported response encryption methods if applicable;
- Supported client types;
- Supported response types;
- Supported transport-layer mechanisms;
- Required client environment capabilities, e.g. in case they take part in the user authentication process;
- Support for dynamic scope values obtained from a scope request.

For a protocol flow which includes an Access Token Request:

- Location of token endpoint;
- Supported/required token endpoint extension parameters;

- Supported/preferred client authentication methods;
- Supported Authorization Grant types;
- Supported/preferred transport-layer mechanisms;
- Types of issued Access Tokens (bearer tokens –generic Access Tokens or Scope Request Access Tokens or both)

When the token revocation feature is supported:

- Location of token revocation endpoint
- Supported/preferred transport-layer mechanisms;

Note that there might be multiple reasons for a given endpoint (e.g. authorization endpoint URL) to not be known statically at the time of Autho4API Client deployment. For instance:

- The Service Provider wants the flexibility to change the endpoint URL over time;
- The Service Provider exposes several authorization endpoints (e.g. each dedicated to a user authentication method, to a well-known set of APIs…);
- The aggregation of Service Providers exposes one endpoint per Service Provider, and client deployment is Service-Provider agnostic in the context of this aggregation.

7.3.1.4 Autho4API Access Control Server parameters

To properly interact with the Autho4API Access Control Server on Autho-3 interface, the Autho4API Client needs to know the following server-specific parameters:

- Types of issued Access Tokens
  - In case of bearer tokens: supported/preferred methods of Access Token inclusion
- Support for scope requests

7.3.2 Discovery Mechanisms

This version of the Autho4API enabler does not define the mechanisms by which the Autho4API Client can obtain the parameters required to interact with the Autho4API Authorization Server and Access Control Server.

7.4 Discovering scope and subscope

7.4.1 Overview (Informative)

The service documentation of a network API typically specifies a well-defined set of Scope Values mapping strictly to identified network resources and operations on these resources. This definition method allows to statically embed these values in the Autho4API Client prior to its deployment. In this case, well-known Scope Values definitions depend on resource access parameters known in advance.

It may occur though that the appropriate scope value for which Resource Owner's authorization should be requested does not only depend on parameters known in advance but also on parameters that are either client-specific or not known in advance. This situation can be addressed by the following subscoping mechanism:

- A well-known Scope Value is defined to represent the access parameters known in advance;
- The appropriate "final" scope value is the well-known Scope Value, narrowed down by some scope-narrowing parameters provided by the Autho4API Client. This final scope value is called subscope.
- The scope-narrowing parameters arepresent (in some form or other) in the resource request payload.

- There are multiple scenarios that allow an Autho4API Client to obtain the required subscope needed in order to accessa given resource:
    - Before client deployment, at the time of client registration: the application registration request includes each combination of well-known Scope Value and scope-narrowing parameters. The Service Provider returns in the registration response one subscope value for each of these combinations. These subscope values (along with well-known Scope Values) can therefore be statically embedded in the Autho4API Client prior to its deployment.
    - After client deployment, at the best opportunity (e.g. at the earliest, subsequently to client's installation, at the latest, prior to Authorization Request).

7.4.2 Obtaining a subscope after deployment

This section specifies a mechanism enabling a deployed Autho4API client to obtain from the Autho4API Access Control Server the scope required to authorize a protected resource request, when this scope can be defined as a well-known Scope Value narrowed down by some scope-narrowing parameters, and if these parameters happen to be carried in some form or other in the protected resource request.

Autho4API Client, Autho4API Authorization Server and Autho4API Access Control Server MAY support this subscoping mechanism. When supported, this section SHALL apply.

This subscoping mechanism is summarized as follows:

- The Autho4API Client first obtains a Scope Request Access Token, which represents the authorization to request scopes;
- The Autho4API Client sends to the Autho4API Access Control Server a scope request, which is a normal protected resource request that includes a Scope Request Access Token;
- The Autho4API Access Control Server computes the required scope based on the well-known Scope Value defined for this resource request and on some scope-narrowing parameters found in the resource request payload;
- The Autho4API Access Control Server returns an "Insufficient scope" error as well as the required scope to be used for requesting Resource Owner's authorization.

7.4.2.1 Obtaining a Scope Request Access Token

The Autho4API Client SHALL first obtain the authorization to request scopes, in the form of a Scope Request Access Token. The method for obtaining this Access Token depends on the type of client and of the supported authorization flows:

- Clients of confidential type SHALL use the Client Credentials flow if supported, or else SHALL use the Authorization Code flow if supported;
- Clients of public type SHALL use the Authorization Code flow if supported, or else SHALL use the Implicit Grant flow if supported.
- The method for obtaining this Access Token depends on the type of client and type of supported authorization flows:

The scope parameter of the Authorization Request (or Access Token Request in case of Client Credentials flow) SHALL contain the list of well-known Scope Values to be subscoped, each followed by the "_subscope" suffix. Each resulting scope values SHALL conform to the `{OMASubScopeValue}` grammar specified in section 7.3.1.2.2.

Following the Resource Owner's authorization (if applicable) and the subsequent Access Token Request, the Autho4API Authorization Server SHALL issue in the Access Token Response a Scope Request Access Token, along with a Refresh Token if supported and if applicable.

Note that an Autho4API Authorization Server not supporting the subscoping functionality will return an Access Token Response with the error "invalid scope".

7.4.2.2 Obtaining a subscope

Once the Autho4API Client has obtained a valid Scope Request Access Token for some well-known Scope Values, it can send scope requests to the Autho4API Access Control Server as follows:

- the scope request SHALL be equal to the final resource request (in terms of Request URI, URI query parameters, headers, payload), but where the authorization credentials SHALL be the Scope Request Access Token;
- the corresponding final resource request SHALL be in the scope of one of the well-known Scope Values for which the Scope Request Access Token has been issued.

The Autho4API Access Control Server then:

- SHALL verify that the Access Token is valid in general (and not expired, not revoked);
- SHALL identify that the Access Token is a Scope Request Access Token (using a method not defined in this specification);
- SHALL verify that the resource request is in the scope of one of the well-known scope value(s) for which the Scope Request Access Token was issued and SHALL select the narrowest of these scope values (called selected well-known scope value);
- SHALL extract from the request payload the scope-narrowing parameters (as defined in the concerned Network API documentation);
- SHALL generate a subscope identifier binding altogether: the resource request characteristics and the scope-narrowing parameters extracted from the payload;
- SHALL construct the subscope value as per `{OMASubScopeValue}` grammar (specified in section 7.3.1.2.2), i.e. the selected well-known scope value suffixed by the subscope identifier;
- SHALL then return an HTTP error response containing the error field set to “insufficient scope” and the scope field set to the subscope value.

7.4.2.3 Obtaining Authorization

Once the Autho4API Client has obtained a subscope from Autho4API Access Control Server, it can request Resource Owner’s authorization using this subscope value in the scope parameter of the Authorization Request.

The Autho4API Authorization Server serving the request and supporting the subscoping functionality SHALL be able to determine that the requested scope is a subscope, and by interacting with the Autho4API Access Control Server which has issued the subscope, SHALL be able to inform the Resource Owner about the reduced scope for which his authorization is requested.

7.4.2.4 Accessing the protected resource

Once the Autho4API Client has obtained the Resource Owner’s authorization for the given subscope, and once it has exchanged the authorization grant for an Access Token (called final access token), the Autho4API Client SHALL request the resource using the same request as the earlier scope request, where the authorization credentials are changed from the Scope Request Access Token to the final acces token.

7.5 Obtaining Authorization

7.5.1 Authorization Code flow

7.5.1.1 Implementation based on OAuth 2.0

This section describes the process by which the Autho4API Client obtains authorization from the Resource Owner to access to the Resources, using the Authorization Code grant type, described in section 4.1 of [RFC6749].

The indications described in section 4.1 of [RFC6749] SHALL be followed by the different actors involved in the flow: Autho4API Client, Resource Owner’s User Agent and Autho4API Authorization Server.

7.5.1.1.1 Detailed protocol flow (Informative)

The flow shown in figure 3 of section 4.1 of [RFC6749] maps to the following flow, using OMA Autho4API Entities and interfaces:

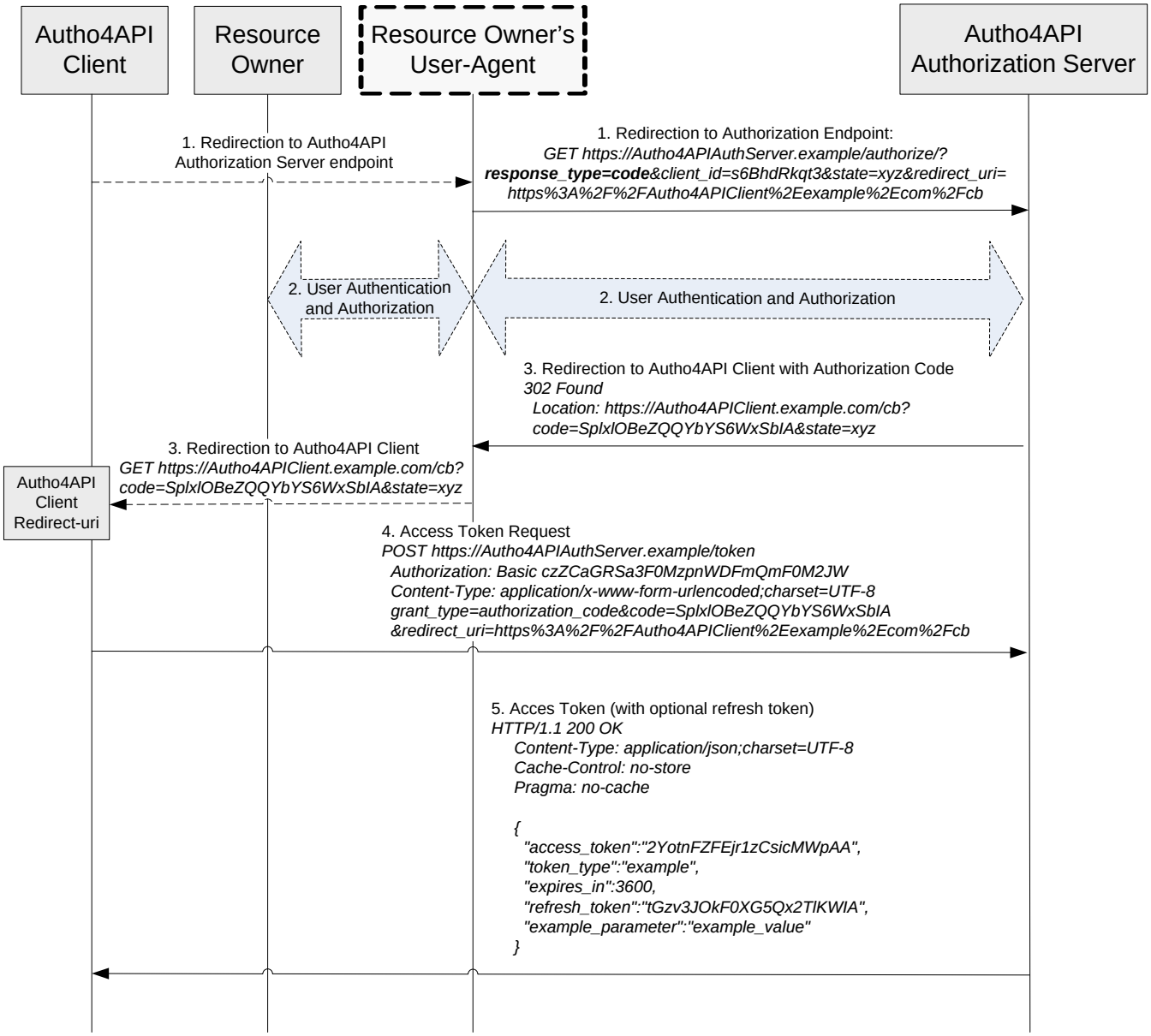

Figure 3: Obtaining Authorization using the Authorization Code grant type: Detailed Protocol Flow

1. Autho4API Client redirects the Resource Owner's User Agent to the Autho4API Authorization Server Endpoint. This step maps with step (A) in section 4.1 of [RFC6749]. The step is detailed in section 4.1.1 of [RFC6749]. How the Autho4API Client redirects the Resource Owner's User Agent is out of scope of this specification, as it is done through Ext-1 interface.
2. The Resource Owner is authenticated and grants to the Autho4API Client the access to the Resources. This step maps with step (B) in section 4.1 of [RFC6749]. How this step 2 is performed is out of scope of Autho4API.
3. The Autho4API Authorization Server answers to the request in step 1 redirecting the Resource Owner's User Agent to the redirection URI provided in step 1; the Resource Owner's User Agent sends the corresponding HTTP GET

request to the URI indicated in the Location header of received HTTP 302 response. This step maps with step (C) in section 4.1 of [RFC6749]. The step is detailed in section 4.1.2 of [RFC6749]. How the Resource Owner's User Agent executes the redirection to Autho4API Client is out of scope of this specification, as it is done through Ext-1 interface.

4. The Autho4API Client sends an Access Token Request to the Autho4API Authorization Server. This step maps with step (D) in section 4.1 of [RFC6749]. The step is detailed in section 4.1.3 of [RFC6749].
5. The Autho4API Authorization Server answers to the Autho4API Client, providing the Access Token and optionally the Refresh Token. This step maps with step (E) in section 4.1 of [RFC6749]. The step is detailed in section 4.1.4 of [RFC6749].

7.5.1.2 Support in Native Applications (Informative)

Client-side installed Applications (like native code Applications), although in general not able to receive incoming HTTP requests (including the HTTP redirection carrying the response to Authorization Request) can nevertheless support the Authorization Code flow using strategies specified in section 7.5.6.

7.5.2 Implicit Grant flow

7.5.2.1 Implementation based on OAuth 2.0

This section describes the process by which the Autho4API Client obtains authorization from the Resource Owner to access to the Resources, using the Implicit Grant type, described in section 4.2 of [RFC6749].

The indications described in section 4.2 of [RFC6749] SHALL be followed by the different actors involved in the flow: Autho4API Client, Resource Owner's User Agent, and Autho4API Authorization Server.

7.5.2.1.1 Detailed protocol flow (Informative)

The flow shown in figure 4 in section 4.2 of [RFC6749] maps to the following flow, using OMA Autho4API Entities and interfaces:

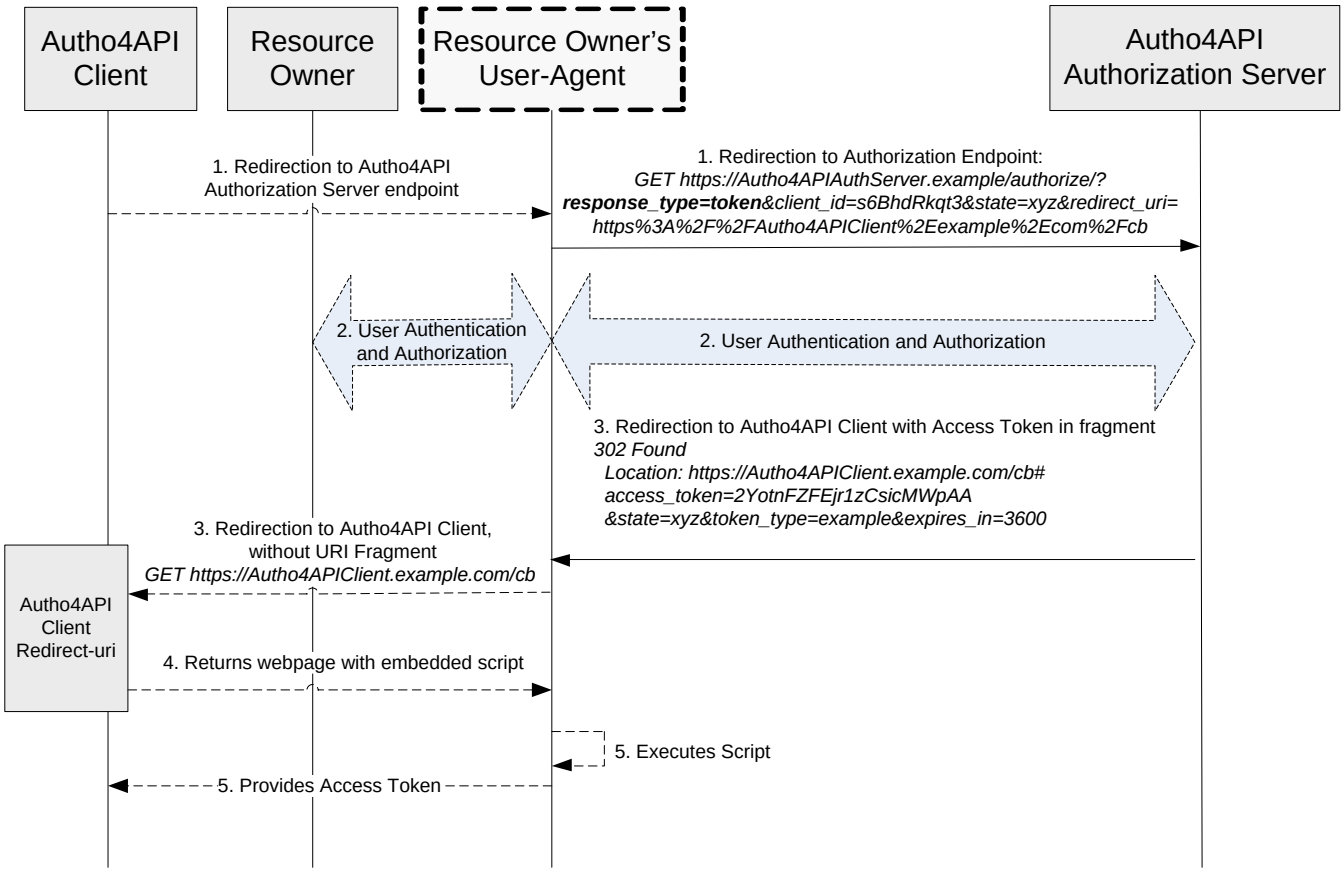

Figure 4: Obtaining Authorization using the Implicit grant type: Detailed Protocol Flow

1. Autho4API Client redirects the Resource Owner's User Agent to the Autho4API Authorization Server Endpoint. This step maps with step (A) in section 4.2 of [RFC6749], The step is detailed in section 4.2.1 of [RFC6749]. How the Autho4API Client redirects the Resource Owner's User Agent is out of scope of this specification, as it is done through Ext-1 interface.
2. The Resource Owner is authenticated and grants to the Autho4API Client the access to the Resources. This step maps with step (B) in section 4.2 of [RFC6749]. How this step 2 is performed is out of scope of Autho4API.
3. The Autho4API Authorization Server answers to the request in step 1 redirecting the Resource Owner's User Agent to the redirection URI provided in step 1; The Access Token is provided in the URI fragment component. The Resource Owner's User Agent sends the corresponding HTTP GET request to the URI indicated in the Location header of received HTTP 302 response, without including the URI Fragment. This step maps with steps (C) and (D) in section 4.2 of [RFC6749]. The step is detailed in section 4.2.2 of [RFC6749]. How the Resource Owner's User Agent executes the redirection to Autho4API Client is out of scope of this specification, as it is done through Ext-1 interface.
4. The Autho4API Client returns a web page (typically an HTML document with an embedded script) capable of accessing the full redirection URI including the fragment retained by the Resource Owner's User-Agent, and extracting the Access Token (and other parameters) contained in the fragment. This step maps with step (E) in section 4.2 of [RFC6749]. Details of this step are out of scope of this specification, as the step is done through Ext-1 interface.
5. The Resource Owner's User Agent executes the script provided by the web-hosted Autho4API Client resource locally, which extracts the Access Token and passes it to the Autho4API Client. This step maps with steps (F) and (G) in section 4.2 of [RFC6749]. Details of this step are out of scope of this specification, as the step is done through Ext-1 interface.

7.5.2.2 Support in Native Applications (Informative)

Client-side installed Applications (like native code Applications), although in general not able to receive incoming HTTP requests (including the HTTP redirection carrying the response to Authorization Request) can nevertheless support the Implicit Grant flow using strategies specified in section 7.5.6.

7.5.3 Resource Owner Password Credentials flow

The Autho4API Client and Authorization Server MAY support the Resource Owner Password Credentials flow, as defined in section 4.3 of [RFC6749]. The mechanism(s) by which the Autho4API Client can obtain the Resource Owner's credentials are out of the scope of this specification.

7.5.4 Client Credentials flow

The Autho4API Client and Authorization Server MAY support the Client Credentials flow, as defined in section 4.4 of [RFC6749]. The mechanism(s) by which the Autho4API Client can obtain its client credentials (or other supported means of authentication) are out of the scope of this specification.

7.5.5 Other flows

The Autho4API enabler MAY support other IETF-defined OAuth 2.0 flows, via the use of extension grant types as defined in section 4.5 of [RFC6749].

7.5.6 Support in Native Applications

[RFC6749] defines some flows (Authorization Code flow, Implicit Grant flow) for which the protocol response subsequent to Resource Owner's authentication and authorization steps is delivered over an HTTP redirection to a client's public HTTP URL.

However client-side installed Applications (like native code Applications) are in general not able to receive incoming HTTP requests (including those resulting from HTTP redirections) and thus not able to receive this protocol response to Authorization Request.

This section specifies some strategies enabling Native Applications to nonetheless support such HTTP-redirect based flows. These strategies (which some are outlined in section 9 of [RFC6749]) fall into two categories:

- Response delivered over an HTTP redirection combined with a device-side mechanism, further detailed in section 7.5.6.1.
- Response delivered over a secondary channel (i.e.: an alternative channel to HTTP redirection), further detailed in section 7.5.7.

7.5.6.1 Response delivery over HTTP redirection (Informative)

This section details methods by which the response to Authorization Request is delivered by the Autho4API Authorization Server to the device over a regular HTTP redirection and then – via some additional mechanism – routed in the device to the Application. The following options are possible:

- **Embedding a User Agent in the Autho4API Client**. In this approach, the Autho4API Client is able to capture HTTP redirections by embedding the User Agent itself. As explained in [RFC6749] though:
  - The use of embedded User Agents may imply security challenges due to the fact that the Resource Owner authenticates in an unidentified window without the visual protections provided by most external User Agents, thus reducing Resource Owner's protection against certain types of attacks (e.g. clickjacking, phising).
- **Running a local Web server.** In this approach, the Autho4API Client invokes an external User Agent later on redirected to a local Web server running as an external generic component or embedded in the Autho4API Client, and the Autho4API Client is then able to interact with this local Web server to obtain the response. This mechanism raises several issues:

  - In case of local Web server external to the Application, a malware Application could also interact with the local Web server to intercept the processed response.
  - In case multiple Applications using this mechanism are installed in the device, and since their redirection URIs share the same domain (i.e. localhost), avoiding redirection URI value conflicts (through assignment of specific port or path or query parameters) requires special care and coordination.

- **WAC Webview Device API**: using this Device API defined in [WAC2.1], the Application executing in a web runtime environment can launch – through the control of the web runtime – a limited browser to the authorization endpoint, and later on receive the response to Authorization Request captured by the web runtime from the HTTP-redirected limited browser. This capture mechanism:
  - can be considered reasonably secure because first the channels for request and response are the same (User Agent browsing context controlled by the web runtime), and second the response can only be returned to the instance of the Application which has initiated the flow.
  - is limited though to the native Applications executing in a web runtime environment (e.g. widgets).

- **Redirection URI with OS-registered URI components**: in this approach, the Redirection URI pre-registered with the Service Provider contains components (scheme, and eventually other components like authority and path) registered with the operating system, typically at Application installation time. Whenever the native browser is opening a URL matching these components, it handles the URL to the Application. This mechanism commonly available on operating systems has a few downsides:
  - The mechanism is OS-specific.
  - It raises a routing problem when several instances of the same Application (sharing the same Redirection URI) reside in the device.
  - It raises the security issue that a malware Application can register the same URI components, and thus intercept the redirection payload.

7.5.7 Secondary channel for response delivery

This section details methods by which the response to Authorization Request is delivered by the Autho4API Authorization Server to the Autho4API Client over a secondary channel (i.e. not the channel of HTTP redirection).

These methods MAY be used by Native Autho4API Clients, and MAY be used also by other types of Clients (public or confidential according to [RFC6749].

7.5.7.1 Delivery methods

- **Response display with Resource Owner interaction**. The response is displayed in the Resource Owner's User Agent (i.e.: in the browser) and the Resource Owner is instructed to copy the response back to the Autho4API Client. This implies that:
  - The Resource Owner is willing to perform this interaction. The user experience is indeed poor especially when the response is long and/or contain special characters (e.g. Access Token).

- **Response delivery to User Agent with automatic client retrieval**. The response is delivered to the Resource Owner's User Agent in such a way that the Autho4API Client can retrieve it by means out of scope of this specification. One approach is the response being displayed in the window title of Resource Owner's User Agent and the Autho4API Client is able to read this title.

- **Response delivery over textual SMS with Resource Owner interaction.** The response is delivered as the payload of a textual SMS, and the Resource Owner is able to read this response and provide it to the Autho4API Client. This implies that:
  - The Resource Owner is willing to perform this interaction. The user experience is indeed poor especially when the response is long and/or contains special characters (e.g. Access Token).

- The Autho4API Authorization Server or the entity that actually sends the SMS is able to know the Resource Owner's MSISDN (using methods out of scope of this specification).

- **Response delivery using OMA Connectionless Push over SMS [OMAPUSH]**. The response is delivered to the OMA Push Client in the device, and then the OMA Push Client routes the response to the Autho4API Client through Push application addressing. This implies that:
  - the Resource Owner's device where the response is pushed has to be the device where the Application and Autho4API Client are running.
  - in case of downloaded Applications and Autho4API Clients (i.e. not pre-installed along with the OMA Push Client), and since these applications do not have in general an Application-ID represented by a well-known application-assigned code, the OMA Push Client will usually need:
    - to accept dynamic registrations of Application-IDs.
    - to support Application-IDs in absolute URI format, which can happen to contain a query component if the "inst" parameter is used to distinguish between multiples instances of the same Application installed in the device.
  - the Autho4API Authorization Server or the entity that actually sends the Connectionless Push over SMS is able to know the Resource Owner's MSISDN (using methods out of scope of this specification).

7.5.7.2 Pre-registered Redirection Endpoint

Autho4API Clients intending to request the use of one or more secondary channels for the delivery of response to Authorization Request SHALL register one Redirection Endpoint per secondary channel intended for use, where the Redirection URI has the following form:

http://{authorizationServer}/autho4apiSecondaryChannel/{channel}

The following table specifies the value of the URI variables:

Name	Description
authorizationServer	The authority part [RFC3986] of Authorization Server's authorization endpoint URL. This part SHALL be optional to provide at registration time for the environments where the Autho4API Client dynamically discovers the authorization endpoint (using some mechanism not defined in this specification).
channel	The specific type of secondary channel to be used. It SHALL take one of the following values: • 'sms_text', to request response delivery over a textual SMS • 'push_over_sms', to request response delivery using OMA Connectionless Push over SMS • 'browser_title', to request response delivery to User Agent with automatic client retrieval by setting browser window title to the response value • 'browser_display', to request response display in browser page for further interaction with Resource Owner

When one of these secondary channels is OMA Connectionless Push over SMS, Autho4API Clients SHALL register the 'app-id-base' part of Push Application-ID, specified in section 7.5.7.3.

Autho4API Clients SHALL NOT register any other URI query parameters defined for the "redirect_uri" parameter, as they are either instance-dependent (e.g. "inst") or Authorization Request-dependent (e.g. "encryption").

Autho4API Authorization Servers SHALL detect an Autho4API Client's request for the potential use of a secondary channel if the Redirection URI submitted for registration has the following prefix:

http://{authorizationServer}/autho4apiSecondaryChannel/

In that case an Autho4API Authorization Server SHALL reject the Redirection Endpoint registration if any of the following is true:

- the Autho4API Authorization Server does not support the secondary channel functionality
- the channel following the URI prefix is missing or unknown or not supported

7.5.7.3 Authorization Request "redirect_uri" parameter

An Autho4API Client willing to receive the response to Authorization Request over a secondary channel SHALL construct the "redirect_uri" parameter of Authorization Request as follows:

1. Select the Redirection URI pre-registered by the Autho4API Client for the use of this secondary channel
2. If not known at the time of registration, set the authority component of this URI to the authority component of Authorization Server's authorization endpoint
3. If required by this specific channel, append to this URI a query component made of the query parameters defined in table below:

Name	Type/value	Optional	Description
app-id-base	Absolute URI as defined in [RFC3986] , or String representation of application-assigned code	Yes	This parameter SHALL be included in case the channel is set to 'push_over_sms'. It SHALL NOT be included otherwise. When a string representation of an application-assigned code: • It corresponds to the hexadecimal representation of the Application-ID assigned code [OMAPUSH] (used to route the Connectionless Push over SMS to the Autho4API Client) that is known at the client registration time, prefixed by "0x". Example: 0x909A When in Absolute URI format: • It corresponds to the part of the Application-ID [OMAPUSH] (used to route the Connectionless Push over SMS to the Autho4API Client) that is known at the client registration time. • If the 'inst' parameter is not conveyed in the Authorization Request, the Autho4API Authorization Server sets the OMA Push Application-ID to the 'app-id-base' URI. • If the 'inst' parameter is conveyed in the Authorization Request, the Autho4API Authorization Server sets the OMA Push Application-ID to the 'app-id-base' URI appended by the 'inst' query parameter. Example: app-id-base=http://example_app_id.com inst=qwertyasdf Resulting Application-ID: http://example_app_id.com?inst=qwertyasdf
inst	String	Yes	This parameter MAY be included in case the channel is set to 'push_over_sms' and if the included 'app-id-base' query parameter is in Absolute URI format. It SHALL NOT be included otherwise. This parameter corresponds to the part of the Application-ID [OMAPUSH] that is specific to an instance of an installed Application and therefore cannot be known at client registration time. This parameter is intended for the client platforms where an installed Application can dynamically register a Push Application-ID to the OMA Push Client.

Name	Type/value	Optional	Description
encryption	String	Yes	This parameter MAY be included to request encryption of the response delivered over secondary channel. The value of this parameter indicates the algorithm to use by Autho4API Authorization Server to encrypt the response with the symmetric key. The encryption algorithm SHALL be Advanced Encryption Standard (AES) described in [AES], with a key size of 128, 192 or 256 bits and with the confidentiality mode of operation Cipher Block Chaining (CBC) described in [AESMode]. The ABNF definition of 'encryption' parameter is: encryption = "AES_" ("128"|"192"|"256") "_CBC"
encryption_key	String	Yes	This parameter SHALL be present if the 'encryption' parameter is included. The value of this parameter indicates the symmetric key to use for response encryption The encryption key is randomly generated by the Autho4API Client and distinct for each Authorization Request. The ABNF definition of 'encryption_key' parameter is: encryption_key = 32 (HEX) | 48 (HEX) | 64(HEX)
encryption_IV	String	Yes	This parameter SHALL be present if the 'encryption' parameter is included. The value of this parameter indicates the initialization vector needed to encrypt the response according to the CBC confidentiality mode. The initialization vector is generated by the Autho4API Client individually for each Authorization Request. The ABNF definition of 'encryption_IV' parameter is: encryption_IV = 32 (HEX)

Autho4API Authorization Servers supporting the secondary channel functionality SHALL support the "encryption", "encryption_key" and "encryption_IV" parameters.

Autho4API Authorization Servers supporting the secondary channel "OMA Connectionless Push over SMS" SHALL support the "app-id-base" and "inst" parameters.

Sample values of 'redirect_uri' parameter:

http://exampleServiceProviderAuthServer.com/autho4apiSecondaryChannel/sms_text

http://exampleServiceProviderAuthServer.com/autho4apiSecondaryChannel/push_over_sms?app-id-base=http%3A%2F%2Fexample_app_id.com&encryption=AES_128_CBC&encryption_key=63cab7040953d051cd60e0e7ba70e18c&encryption_IV=6353e08c0960e104cd70b751bacad0e7

http://exampleServiceProviderAuthServer.com/autho4apiSecondaryChannel/push_over_sms?app-id-base=http%3A%2F%2Fexample_app_id.com&inst=qwertyasdf

http://exampleServiceProviderAuthServer.com/autho4apiSecondaryChannel/push_over_sms?app-id-base=0x909A

http://exampleServiceProviderAuthServer.com/autho4apiSecondaryChannel/browser_title

7.5.7.4 Request for response delivery over secondary channel

An Autho4API Client willing to receive the response to Authorization Request over a secondary channel SHALL make sure beforehand that:

1. It has registered a redirection endpoint with the Autho4API Authorization Server specifying the potential use of this channel
2. It has registered its Application-ID (including 'inst' part if applicable and if in Absolute URI format) with the OMA Push Client, in case of response delivery using OMA Connectionless Push over SMS

The Autho4API Client SHALL then construct each Authorization Request as follows:

3. The "redirect_uri" parameter SHALL be included, with a value structured as defined in section 7.5.7.3. In addition:
    - The Autho4API Client MAY require response encryption over secondary channel, using the encryption-related URI-query parameters specified in previous section. It is an implementation decision whether to mandate the Autho4API Client to request response encryption, depending on the Autho4API Client nature and the used secondary channel.
4. Regarding the "state" parameter defined in [RFC6749]:
    - It SHOULD NOT be included in the cases of "Response display with Resource Owner interaction" and "Response delivery over textual SMS with Resource Owner interaction"
    - It SHOULD be included in the cases of "Response delivery to User Agent with automatic client retrieval" and "Response delivery using OMA Connectionless Push over SMS"
5. Other server-specific authorization endpoint extensions (not defined in this specification) MAY be included
6. Other information required for the proper response delivery over indicated channel MAY be included

A response (whether in plain text or encrypted) requested to be delivered over SMS (OMA Connectionless Push over SMS or a textual SMS), SHOULD ideally fit within one single SMS for transport reliability reasons, and SHOULD NOT exceed the size of a 4-segment concatenated SMS, as this is the minimal size which an OMA Push Client is required to reassemble. To guarantee this, the Autho4API Authorization Server can instruct the Autho4API Client to use a state parameter not longer than a given length, by means out of scope of this specification.

7.5.7.5 Request processing and error handling

This section details Autho4API Authorization Server's error handling when the Autho4API Client requests the use of secondary channel in Authorization Request via the "redirect_uri" specific structure defined in section 7.5.7.3.

Autho4API Authorization Servers SHALL detect an Autho4API Client's request for the use of a secondary channel if the "redirect_uri" parameter has the following prefix:

http://{authorizationServer}/autho4apiSecondaryChannel/

where {authorizationServer} matches the authority component of the authorization endpoint URL to which the Authorization Request is sent.

In that case the Autho4API Authorization Server SHALL raise an "invalid Redirection URI" error if:

- It does not supporting the secondary channel functionality
- It supports the secondary channel functionality but any of the following is true:
    - the "redirect_uri" is not a valid URI

- the "redirect_uri" does not match the general syntax of "redirect_uri" specified in section 7.5.7.3
- the Autho4API Client is registered, but "redirect_uri" value (not including its query component) does not match any of the Redirection Endpoints registered for this client
- the Autho4API Client is registered, and "redirect_uri" value (not including its query component) matches a Redirection Endpoint registered for this client, and the signaled secondary channel is "OMA Connectionless Push over SMS", but the Application-ID provided in the request (not including the "inst" parameter) does not match the registered one for this client.
- the Autho4API Client is unregistered, but the Autho4API Authorization Server does not support the indicated secondary channel

An Autho4API Authorization Server raising an "invalid Redirection URI" error SHALL follow the error handling defined in section 3.1.2.4 of [RFC6749].

If neither the error cases above nor other Authorization Request error cases specified in [RFC6749] are raised, the Autho4API Authorization Server SHALL verify the query component of "redirect_uri". If present:

- The Autho4API Server SHALL raise an "invalid_request" error if the query component contains an unknown key or value (in respect of section 7.5.7.3) or does not support some key or value (e.g. an AES key length).

If the Authorization Request fails for any reasons, the Autho4API Authorization Server SHOULD in all cases inform the Resource Owner of this situation, i.e. via an explanatory Web page.

7.5.7.6 Response encoding

This section specifies how the Autho4API Authorization Server SHALL construct the response to be sent to Autho4API Client, which varies depending on the secondary channel to use and on the protocol flow.

For the cases of "Response display with Resource Owner interaction" and "Response delivery over textual SMS with Resource Owner interaction":

If the Authorization Request fails for any reasons, the Autho4API Authorization Server SHALL NOT send a response over the secondary channel.

Otherwise, the response to be delivered over secondary channel SHALL be constructed as follows:

1. Encode the value of applicable flow-specific parameter using the "application/x-www-form-urlencoded" encoding type defined in [HTML_4.01]. This parameter is:
   - For the Authorization Code flow: "code"

     Example: vc1234
   - For the Implicit Grant flow: "access_token"

     Example: AAACa8r8lZA4ABAAp0vw1sSG
2. If the encryption parameters defined in section 7.5.7.3 are provided in the Authorization Request, encrypt the previous string using these parameters and encode the resulting data as Base64, otherwise leave the response in plain text
3. Include the resulting string in the text of the browsed page or textual SMS together with optional textual information instructing the Resource Owner about what to do with the response.

For the cases of "Response delivery to User Agent with automatic client retrieval" and "Response delivery using OMA Connectionless Push over SMS", the response SHALL be constructed as follows:

1. Establish the list of applicable parameters (key/value pairs) to be included in the response, which includes the applicable flow-specific parameters with the considerations included in this section, and which may include other server-specific parameters (not defined in this specification). Applicable flow-specific parameters are:

- For the Authorization Code flow: the OAuth 2.0 parameters defined in section 4.1.2 (if successful response) and in section 4.1.2.1 (if error response) of [RFC6749]

  Example: code=vc1234&state=xyz

  Example: code=1vc456

  Example: error=access_denied

- For the Implicit Grant flow: the OAuth 2.0 parameters defined in section 4.2.2 (if successful response) and in section 4.2.2.1 (if error response) of [RFC6749]
  Example: access_token=AAACa8r8lZA4ABAAp0vw1sSG&token_type=Bearer&expires_in=3600&state=xyz

2. Encode the resulting form data set using the "application/x-www-form-urlencoded" encoding type defined in [HTML_4.01].
3. If the encryption parameters defined in section 7.5.7.3 are provided in the Authorization Request, encrypt the previous string using these parameters

For the case of “Response delivery to User Agent with automatic client retrieval”:

4. If the response is encrypted, encode the resulting data as Base64, otherwise leave the response in plain text

For the case of “Response delivery using OMA Connectionless Push over SMS”:

5. Construct the Push Message as per [OMAPUSH], with the content body containing the response
6. If the response is encrypted, set Content-Type header to “application/octet-stream” [RFC2045][RFC2046], otherwise set Content-Type header to “text/plain”
7. Set X-Wap-Application-Id header to the Autho4API Client’s “app-id-base” value, and - when in absolute URI format - appended by “inst” query parameter if provided in the Authorization Request

7.5.7.7 Detailed protocol flows (Informative)

When the response to Authorization Request is delivered over a secondary channel, the general Authorization Code flow shown in Figure 3 is modified as follows:

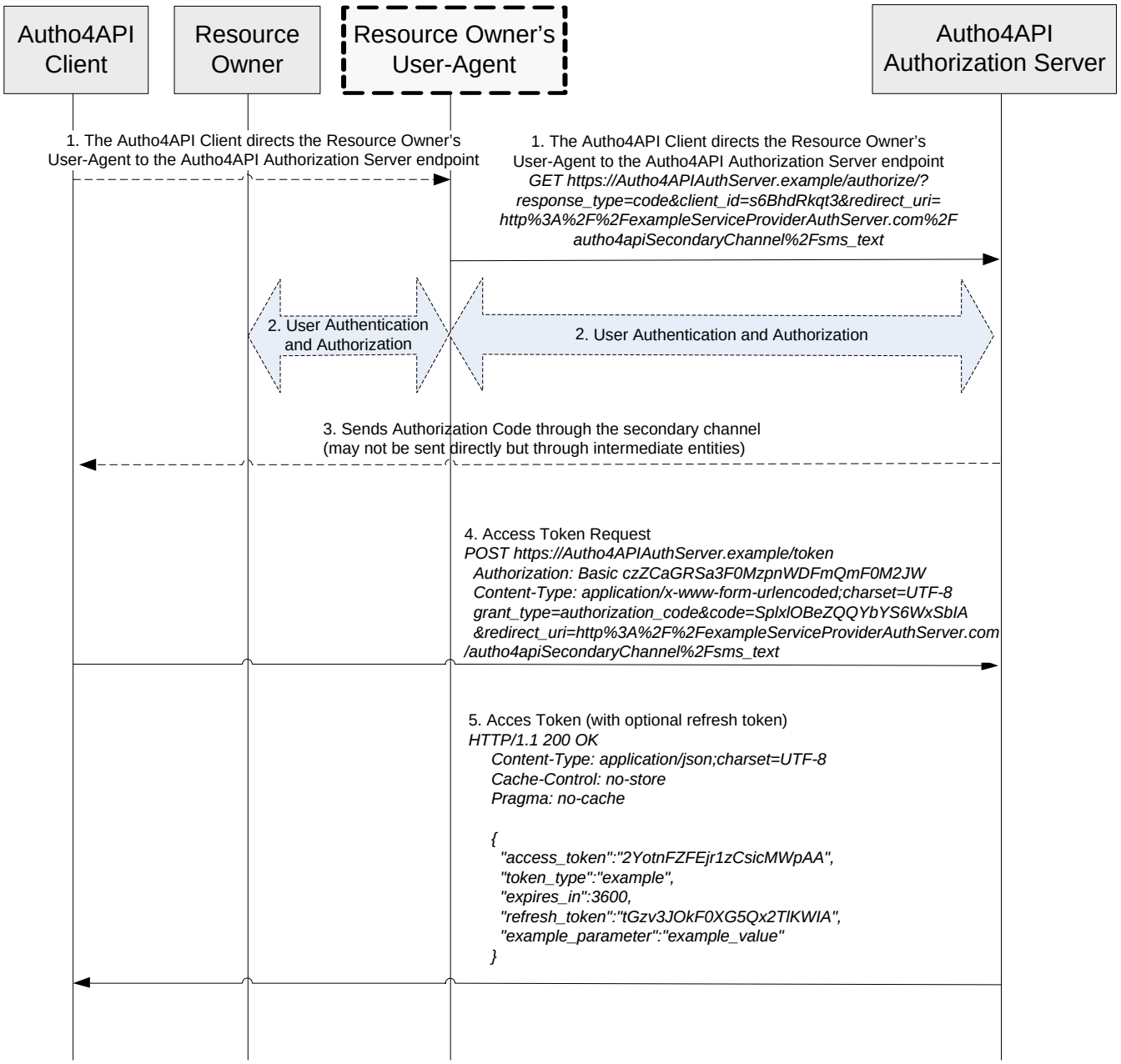

Figure 5: Obtaining Authorization using the Authorization Code grant type and a secondary channel: Detailed Protocol Flow

1. Autho4API Client directs the Resource Owner's User Agent to the Autho4API Authorization Server Endpoint, with the redirect_uri parameter set to http://exampleServiceProviderAuthServer.com/autho4apiSecondaryChannel/sms_text as specified in section 7.5.7.3 to request the Authorization Response to be delivered over textual SMS. For the rest of parameters, description in step 1 of Figure 3 is followed.
2. The Resource Owner is authenticated and grants to the Autho4API Client the access to the Resources. This step is the same as step 2 of Figure 3. How this step 2 is performed is out of scope of Autho4API.
3. The Autho4API Authorization Server sends the Authorization Response through the secondary channel. How this step 2 is performed is out of scope of Autho4API and can involve other entities such as SMSCs, etc.

4. The Autho4API Client sends an Access Token Request to the Autho4API Authorization Server. This step is the same as step 4 of Figure 3.
5. The Autho4API Authorization Server answers to the Autho4API Client, providing the Access Token and optionally the Refresh Token. This step is the same as step 5 of Figure 3.

When the response to Authorization Request is delivered over a secondary channel, the general Implicit Grant flow shown in Figure 4 is modified as follows:

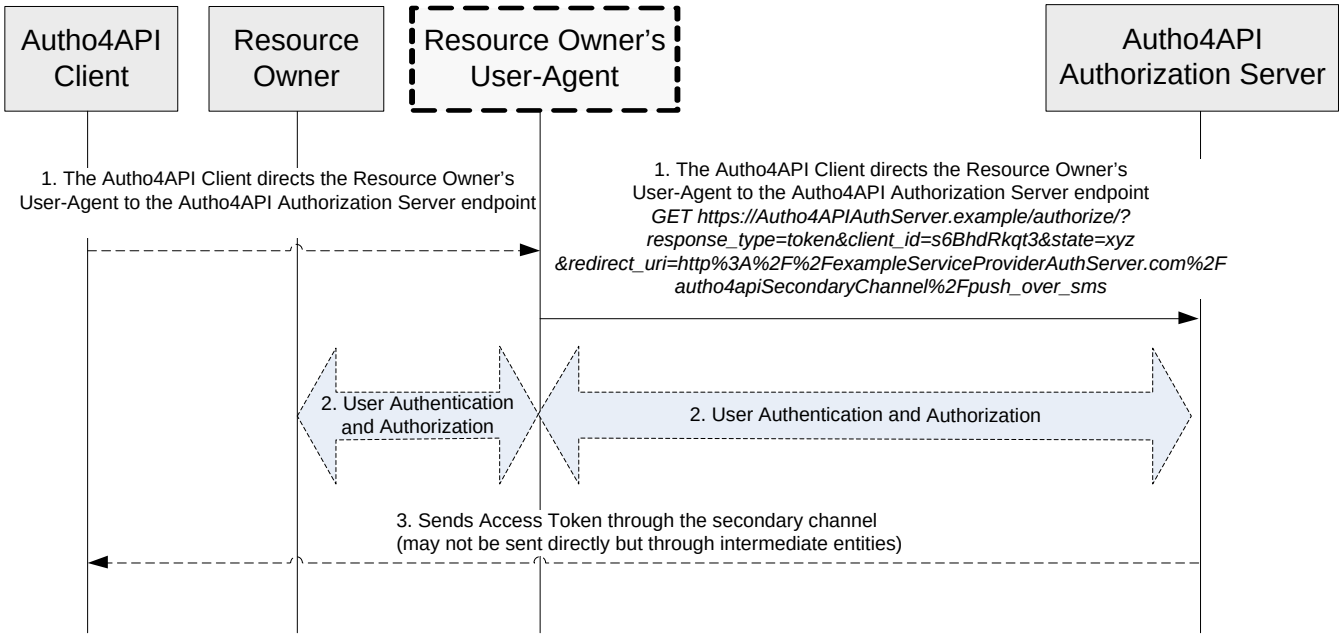

Figure 6: Obtaining Authorization using the Implicit grant type and a secondary channel: Detailed Protocol Flow

1. Autho4API Client directs the Resource Owner's User Agent to the Autho4API Authorization Server Endpoint with the redirect_uri parameter set to http://exampleServiceProviderAuthServer.com/autho4apiSecondaryChannel/push_over_sms as specified in section 7.5.7.3 to request the Authorization Response to be delivered over Connectionless Push over SMS. For the rest of parameters, description in step 1 of Figure 4 is followed.
2. The Resource Owner is authenticated and grants to the Autho4API Client the access to the Resources. This step is the same as step 2 of Figure 4. How this step 2 is performed is out of scope of Autho4API.
3. The Autho4API Authorization Server sends the Access Token Response through the secondary channel. How this step 2 is performed is out of scope of Autho4API and can involve other entities such as SMSCs, etc.

7.5.7.8 Security considerations

Depending on the environment (device, operating system) and on whether the response to Authorization Request is delivered to the same device where the Autho4API Client is running, secondary channel may not be considered secure.

> Note: Whether a secondary channel can be considered secure is out of scope of this specification, as a secondary channel can be secure or not depending on the environment

For the Authorization Code flow:

- If the secondary channel is not considered secure and the Autho4API Client is confidential according to section 2.1 of [RFC6749], the secondary channel MAY be used anyway, as (due to client authentication enforcement) capturing an Authorization Code is not enough to exchange it for an Access Token. Nevertheless, in order to enhance transport

security, the Autho4API Client SHOULD request the response to be encrypted as defined in section 7.5.7.3, thus making the channel secure.

- If the secondary channel is not considered secure and the Autho4API Client is public according to section 2.1 of [RFC6749], the Autho4API Client SHALL request the response to be encrypted as defined in section 7.5.7.3, thus making the channel secure.

For the Implicit Grant flow:

- If the secondary channel is not considered secure, the Autho4API Client SHALL request the response to be encrypted as defined in section 7.5.7.3, thus making the channel secure.

For the strategies where the Autho4API Client is able to retrieve the response without any Resource Owner interaction, it is highly recommended that the Autho4API Client uses the 'state' parameter defined in section 4.1 of [RFC6749] in order to verify that a response received over the secondary channel is matching a pending Authorization Request, and by this prevent cross-site forgery attacks.

7.6 Issuing an Access Token

In reply to Access Token Requests sent by Autho4API Client on Autho-2 interface, the Autho4API Authorization Server SHALL return either a successful or an error Access Token Response as specified in section 5 of [RFC6749].

7.7 Refreshing an Access Token

An Autho4API Client which has obtained an Access Token along with a Refresh Token, can later on refresh this Access Token when it expires by making a refresh request to Autho4API Authorization Server's token endpoint on Autho-2 interface, as specified in section 6 of [RFC6749].

7.8 Accessing Protected Resources

7.8.1 Overview

7.8.1.1 Access Token Validation

The Autho4API Access Control Server SHALL check the validity of an Access Token presented in the resource request before serving the request to the Autho4API Client, as per section 7 of [RFC6749]. The Autho4API Access Control Server SHALL in particular conclude to Access Token invalidity - and deny resource access - if any of the following is true:

- the Access Token is a one-time Access Token, and has already been successfully used once to serve any protected resource request in the scope of the Access Token
- the Access Token is expected to conform to an integrity-protected format, and is detected as altered
- the Access Token is expected to conform to a structured format, and is detected as malformed
- the Access Token is unknown to the Autho4API Authorization Server(s) which the Autho4API Access Control Server has a trust relationship with
- the Access Token has expired
- the Access Token – or the Refresh Token used to obtain the Access Token – has been revoked
- the Access Token has not a sufficient scope for this resource request

For the case of bearer tokens: if the protected resource request does not include any Access Token or does not contain a valid Access Token, the Autho4API Access Control Server SHALL return to the Autho4API Client an error response constructed according to section 3 of [RFC6750].

How and when the Autho4API Access Control Server interacts with the Autho4API Authorization Server to perform Access Token validity check is out of the scope of this specification. The possible approaches can be characterized as follows:

- security: the Autho4API Access Control Server never happens to accept an invalid Access Token
  - e.g. when this Access Token has just been revoked by Autho4API Authorization Server
- reliability: the Autho4API Access Control Server never happens to reject a valid Access Token
  - e.g. when this Access Token has just been issued by Autho4API Authorization Server
- scalability: the Autho4API Access Control Server is able to process a high number of resource requests (whether they contain no or invalid or valid Access Token)

One most secure and reliable approach is for the Autho4API Access Control Server to entirely delegate Access Token validity check to the Autho4API Authorization Server, upon each received resource request (meaning in particular that Autho4API Access Control Server does not cache Access Token validities). Other more scalable approaches (e.g. where the Autho4API Access Control Server can in some cases conclude on its own to the validity or invalidity of a presented Access Token) SHALL provide the same level of security.

7.8.1.2 One-time Access Tokens

OAuth 2.0 Access Tokens defined in [RFC6749] can be indefinitely used by the client to access the protected resource, till the Access Token expires or is revoked. Expiry time can be explicitly signaled to the client via the "expires_in" parameter attached to an issued Access Token in the Access Token Response.

There is on the other hand no IETF Access Token Response parameter specifying a limited count of use for an issued Access Token. For certain types of resources though (e.g. a payment transaction), it is critical for security reasons that the Access Token be used once only. The Autho4API Authorization Server and Access Control Server could mutually agree to enforce this one-time use in the back-end, but a client would nonetheless attempt to reuse such issued Access Token multiple times, unnecessarily overloading the Access Control Server and increasing the total time for a client to restart the authorization process.

One way of realizing this one-time use without creating a new Access Token Response extension is to specify by convention that some Scope Values of the Network API are defined for the issuing of one-time Access Tokens only. Whenever a Network API defines such a Scope Value, the Autho4API enabler SHALL support the one-time Access Token functionality in the following way:

- In the scope parameter included in the first protocol flow request, the Autho4API Client SHALL only include this Scope Value (i.e. it SHALL NOT include multiple Scope Values).
- The Autho4API Authorization Server SHALL NOT issue a Refresh Token along with a one-time Access Token
- The Autho4API Access Control Server SHALL permanently invalidate a one-time Access Token as soon as it has been successfully used once to access the protected resource.
- The Autho4API Client SHALL discard a one-time Access Token as soon as it has successfully used it once to access the protected resource.

The Autho4API enabler MAY otherwise support the one-time Access Token functionality, using methods not defined by this specification.

7.8.1.3 Self-contained Access Token formats

This specification does not restrict the possible formats usable for the construction of an Access Token, which can denote an identifier used to retrieve the authorization information, or else self-contain the authorization information itself or some Access Token properties (expiry time, one-time Access Token indication…), for the purpose e.g. of increasing the scalability of token validity check by Autho4API Access Control Servers. In the latter case, the following apply:

- An Autho4API Authorization Server issuing a self-contained Access Token:
  - SHALL use a format protecting the confidentiality of any sensitive information the Access Token may carry (e.g. related to Resource Owner's privacy)
  - SHALL use a format protecting the integrity of the Access Token

- An Autho4API Access Control Server checking the validity of a self-contained Access Token presented for the first time in a protected resource request:
  - SHALL NOT conclude to the validity of the Access Token based on the sole information it contains, but SHALL collaborate with the Autho4API Authorization Server to determine such validity.

7.8.2 Access Token Types

The Autho4API enabler SHALL support the bearer access token type defined in [RFC6750].

7.8.3 Bearer Tokens

Regarding the method for sending a bearer token to make authenticated requests to access protected resources:

- The Autho4API Client SHOULD support the method where the Access Token is sent in an "Authorization" request header field, as defined in section 2.1 of [RFC6750]. The Autho4API Client MAY support other sending methods defined in [RFC6750].
- The Autho4API Access Control Server SHALL support the method where the Access Token is sent in an "Authorization" request header field, as defined in section 2.1 of [RFC6750]. The Autho4API Access Control Server MAY support other sending methods defined in [RFC6750].

Regarding the transport layer security of protected resource requests sent with a bearer token:

- The Autho4API Access Control Server SHALL support TLS 1.1 [RFC4346] and SHOULD support TLS 1.2 [RFC5246].

7.9 Multi-Service Provider environments (Informative)

This section describes deployment options for Autho4API Enabler in multi-Service Provider environments. These environments present the following characteristics:

- More than one Service Provider is offering the same set of protected Resources.
- Each Resource Owner belongs to a certain Service Provider.
- A specific Autho4API Client MAY be used by Resource Owners belonging to any of the Service Providers.

There are several environments with the above characteristics. The common point for all of them is that Autho4API enabler SHALL make possible for the Autho4API Client to obtain authorization to access the protected resources of any Resource Owner, no matter which of the Service Providers the Resource Owner belong to.

The different environments, depicted in the following sections, are divided depending on the different implications to the involved Autho4API entities, and the way the scenario is technically solved.

These scenarios are though not mutually exclusive, as combinations of them are possible.

7.9.1 Autho4API Client discovering the specific Autho4API Authorization Server

This scenario presents the following characteristics in addition to the general ones described for the Multi-Service Provider environment:

- There are multiple Autho4API Authorization Servers each serving Resource Owners belonging to a Service Provider or to a set thereof.
- The Autho4API Client uses a Discovery process, out of scope of this specification, to identify the URLs of the Autho4API Authorization Server serving the corresponding Resource Owner.

Once the Autho4API Authorization Server has been identified the Autho4API Client proceeds to obtain authorization as described in section 7.4, targeting the Authorization Request to the URL retrieved as part of the Discovery procedure.

7.9.2 Autho4API Client requesting authorization through a single Autho4API Authorization Server

This scenario can be divided into the two following sub cases:

7.9.2.1 Single Autho4API Authorization Server serving multiple Service Providers

This scenario presents the following characteristics in addition to the general ones described for the Multi-Service Provider environment:

- There is a single Autho4API Authorization Server generating and managing Access Tokens for the Resource Owners from all Service Providers.

In this scenario the Autho4API Client proceeds to obtain authorization as described in section 7.4, targeting the Authorization Request to the single Autho4API Authorization Server.

Note: In this scenario the Autho4API Client can use a Discovery process, out of scope of this specification, to identify the URLs of the single Autho4API Authorization Server.

7.9.2.2 Multiple Autho4API Authorization Servers serving multiple Service Providers

This scenario presents the following characteristics in addition to the general ones described for the Multi-Service Provider environment:

- There are multiple Autho4API Authorization Servers, each generating and managing Access Tokens for Resource Owners belonging to a specific Service Provider.
- The Autho4API Client directs the Authorization Requests to a single Autho4API Authorization Server acting as entry point, i.e. the (Shared) Autho4API Authorization Server, that in turn interacts with the Autho4API Authorization Server serving each specific Resource Owner.
- The Autho4API Client does not need to be informed about the details of the Service Provider associated to the Resource Owner before completing the OAuth flow.

In this scenario the Autho4API Client proceeds to obtain authorization as described in section 7.4, targeting the Authorization Request to the single Autho4API Authorization Server acting as entry point.

Note: This Multi Service Provider environment maps with the deployment scenario described in section C.1.

Note: In this scenario the Autho4API Client can use a Discovery process, out of scope of this specification, to identify the URLs of the (Shared) Autho4API Authorization Server acting as entry point.

7.9.2.2.1 Client Registration

In this scenario the Autho4API Client registers in the (Shared) Autho4API Authorization Server as described in section 7.1.

Upon registration of the Autho4API Client, the (Shared) Autho4API Authorization Server registers on behalf of the Autho4API Client in the Autho4API Authorization Servers of the different Service Providers, with the following consideration. If the Autho4API Client registered a redirection URI in the (Shared) Autho4API Authorization Server, the redirection URI registered in the Autho4API Authorization Server of each Service Provider SHALL be the URI of the (Shared) Autho4API Authorization Server with the following query parameter containing the redirection URI registered by the Autho4API Client.

Name	Type/value	Optional	Description
client_redirect_uri	xsd:anyuri	NO	redirection URL registered by the Autho4API Client in the (Shared) Autho4API Authorization Server

Example:

https://sharedAuthServerURI.com/?client_redirect_uri=https%3A%2F%2FclientURI.com

7.9.2.2.2 Obtaining Authorization

For the obtaining authorization, the following considerations apply.

- The (Shared) Autho4API Authorization Server offering the authorization and token endpoints to Autho4API Client must be able to cache Authorization Codes and Access Tokens.
    - When Authorization Code grant is used, Authorization Codes are cached so the Autho4API Authorization Server offering the token endpoint can redirect Access Tokens requests to the Appropriate Autho4API Authorization Server of a Service Provider.
    - Access Tokens are cached so the Autho4API Access Control Server acting as entry point for resource requests can redirect them to the Appropriate Autho4API Access Control Server of a Service Provider.
- The (Shared) Autho4API Authorization Server offering the authorization and token endpoints to Autho4API Client must be able to redirect Resource Owner's User Agent to the Autho4API Authorization Server of a Service Provider.

    Note: To perform this step, the Autho4API Authorization Server offering the authorization and token endpoints will discover the Service Provider of the end user, by means out of scope of this specification.

- The (Shared) Autho4API Authorization Server offering the authorization and token endpoints to Autho4API Client must be able to indicate a redirection URI where the Resource Owner's User Agent will be sent back after User Authentication and Authorization. The redirection URI will be included by the (Shared) Autho4API Authorization Server in the redirection of the Resource Owner's User Agent to the Autho4API Authorization Server of a Service Provider only if the Autho4API Client included it in the initial Authorization Request. In that case, the redirect_uri URI will be the URI of the (Shared) Autho4API Authorization Server with a query parameter 'client_redirect_uri' containing the URI indicated by the Autho4API Client, which matches the one registered previously as described in section 7.9.2.2.1.
- The Autho4API Authorization Server offering the authorization and token endpoints to Autho4API Client must be able to cache the mapping between the following elements:
    - When Authorization Code grant is used, an Authorization Code with a certain Autho4API Authorization Server of a Service Provider
    - An Access Token with a certain Autho4API Authorization Server of a Service Provider

7.9.2.2.2.1 Detailed protocol flow – Authorization Code

The general flow shown in Figure 3 is modified as follows:

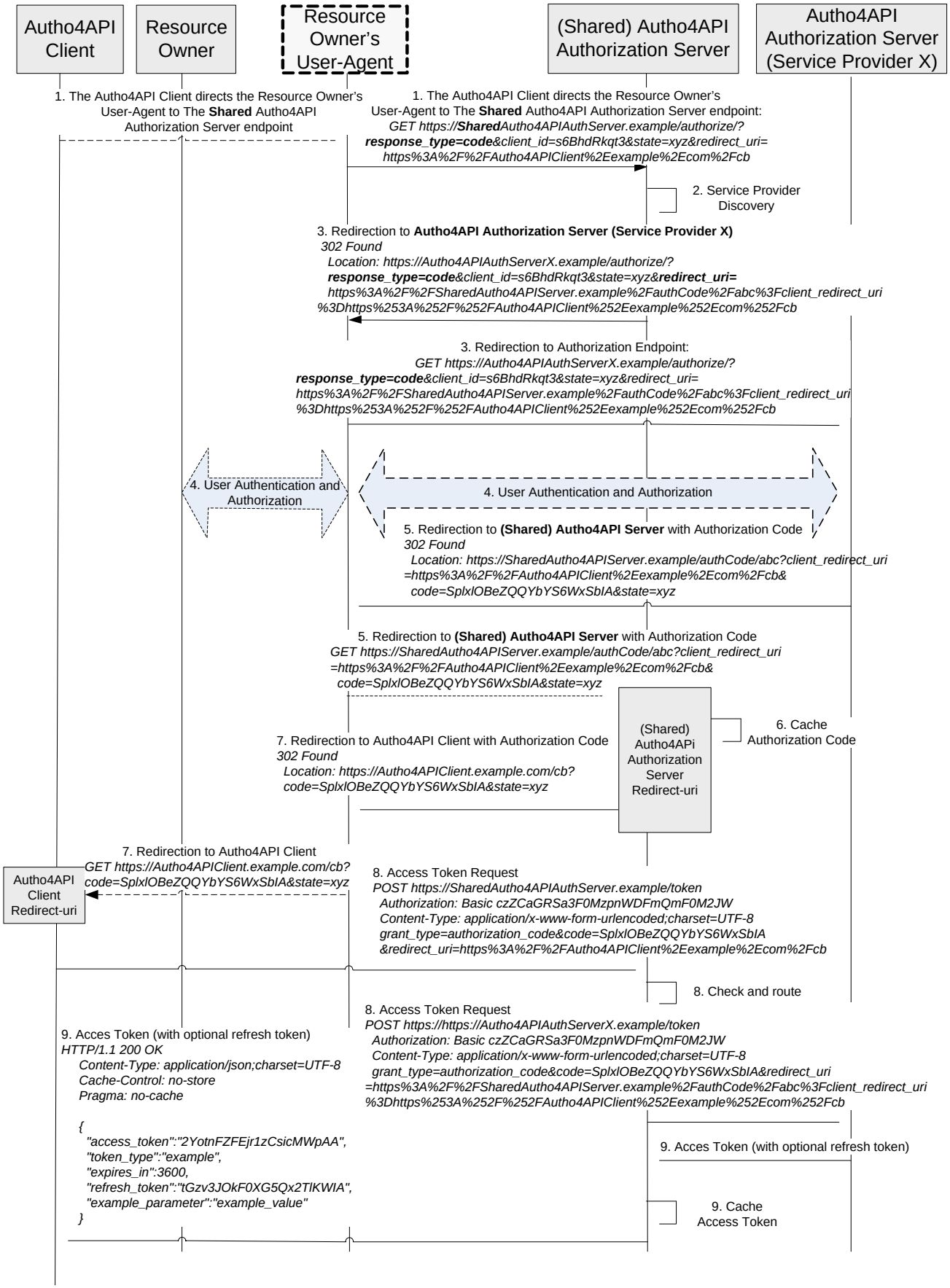

Figure 7: Obtaining Authorization using the Authorization Code grant type: Multiple Autho4API Authorization Servers serving multiple Service Providers detailed protocol flow

1. Autho4API Client directs the Resource Owner's User Agent to the (Shared) Autho4API Authorization Server authorization endpoint.
2. (Shared) Autho4API Authorization Server discovers the Resource Owner's Service Provider. There are several ways to discover the Service Provider, by means of network authentication or by other means. If none of the mechanisms are available, the Resource Owner can be requested to select her Service Provider.
3. Once the Service Provider is known, (Shared) Autho4API Authorization Server redirects the Resource Owner's User Agent to the Autho4API Authorization Server of Service Provider X Endpoint. (Shared) Autho4API Authorization Server provides an URI in the 'redirect_uri' parameter, so that the Resource Owner's User Agent can be redirected back to the (Shared) Autho4API Authorization Server later on. This provided URI is the URI of the (Shared) Autho4API Authorization Server including a query parameter 'client_redirect_uri' containing the 'redirect_uri' indicated by Autho4API Client in step 1.
4. The Resource Owner is authenticated and grants to the Autho4API Client the access to the Resources.
5. The Autho4API Authorization Server of Service Provider X answers to the request in step 3 redirecting the Resource Owner's User Agent to the redirection URI provided in step 3; the Resource Owner's User Agent sends the corresponding HTTP GET request to the URI indicated in the Location header of received HTTP 302 response.
6. (Shared) Autho4API Authorization Server caches the received Authorization Code and associates it with Service Provider X.
7. The (Shared) Autho4API Authorization Server answers to the GET request in step 5 redirecting the Resource Owner's User Agent to the redirection URI received in the 'client_redirect_uri' query parameter which matches the redirect_uri provided by the Autho4API Client in step 1; the Resource Owner's User Agent sends the corresponding HTTP GET request to the URI indicated in the Location header of received HTTP 302 response.
8. The Autho4API Client sends an Access Token Request to the (Shared) Autho4API Authorization Server. (Shared) Autho4API Authorization Server checks the validity of Authorization Code, which was cached in step 6, and routes the Access Token Request to Autho4API Authorization Server of Service Provider X. In case that the Autho4API Client includes a redirect_uri parameter in the Access Token Request the (Shared) Autho4API Authorization Server shall include a redirect_uri parameter in the Access Token Request that it subsequently sends to the Autho4API Authorization Server of Service Provider X. In that case, the redirect_uri URI will be the URI of the (Shared) Autho4API Authorization Server with a query parameter 'client_redirect_uri' containing the redirect_uri provided by the Autho4API Client in the Access Token Request (which matches the one registered previously as described in section 7.9.2.2.1 and thus the one used in step 3).
9. The Autho4API Authorization Server of Service Provider X answers to the Autho4API Client, providing the Access Token and optionally a Refresh Token. (Shared) Autho4API Authorization Server caches the Access Token and, if present, the Refresh Token, associates them with Service Provider X and forwards the response to Autho4API Client.

7.9.2.2.2.2 Detailed protocol flow – Implicit Grant

The general flow shown in Figure 4 is modified as follows:

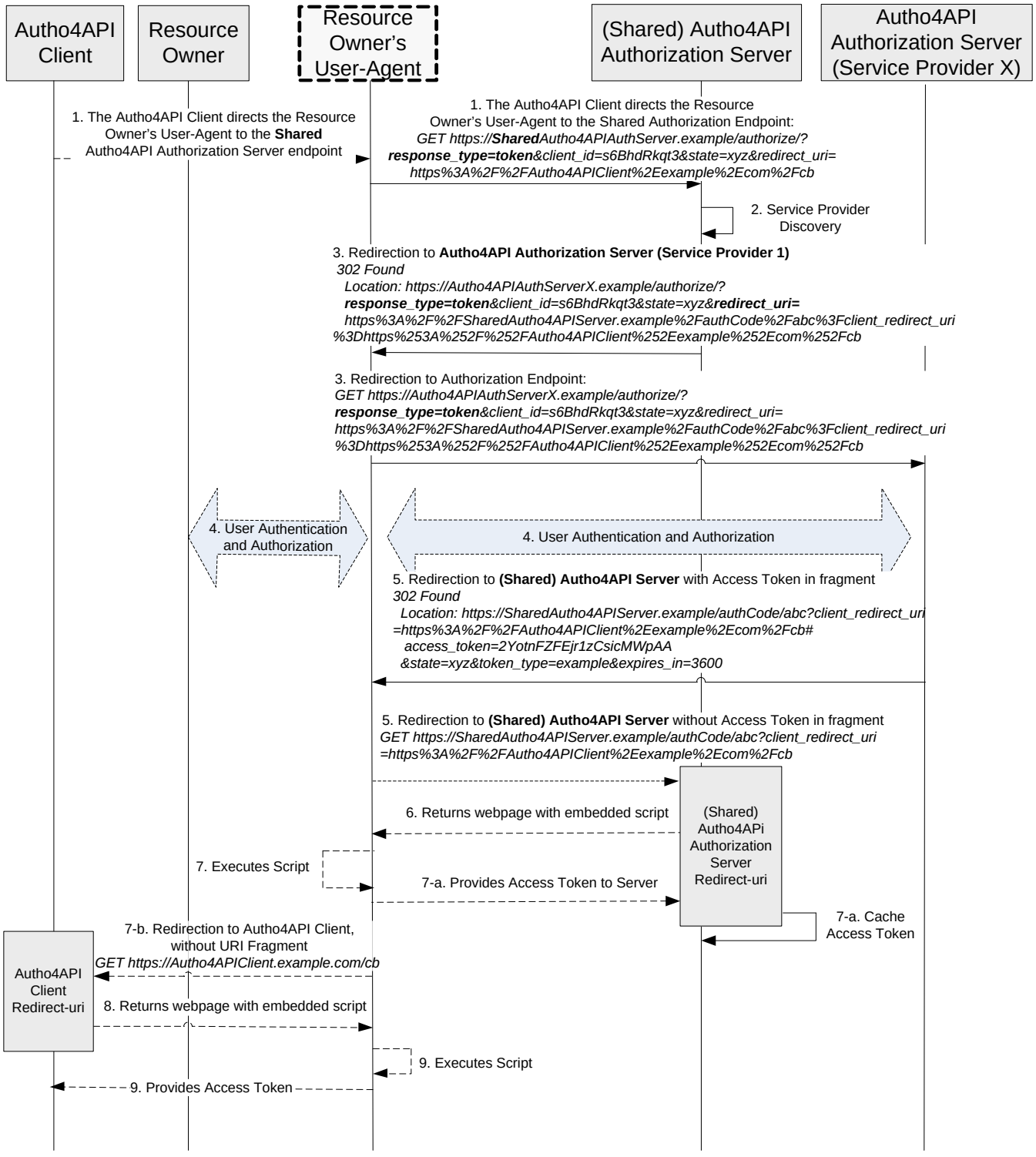

Figure 8: Obtaining Authorization using the Implicit Grant type: Multiple Autho4API Authorization Servers serving multiple Service Providers detailed protocol flow

1. Autho4API Client directs the Resource Owner's User Agent to the (Shared) Autho4API Authorization Server authorization endpoint.
2. (Shared) Autho4API Authorization Server discovers the Resource Owner's Service Provider. There are several ways to discover the Service Provider, by means of network authentication or by other means. If none of the mechanisms are available, the Resource Owner can be requested to select her Service Provider

3. Once the Service Provider is known, (Shared) Autho4API Authorization Server redirects the Resource Owner's User Agent to the Autho4API Authorization Server of Service Provider X Endpoint. (Shared) Autho4API Authorization Server provides an URI in the 'redirect_uri' parameter, so that the Resource Owner's User Agent can be redirected back to the (Shared) Autho4API Authorization Server later on. This provided URI is the URI of the (Shared) Autho4API Authorization Server including a query parameter 'client_redirect_uri' containing the 'redirect_uri' indicated by Autho4API Client in step 1.
4. The Resource Owner is authenticated and grants to the Autho4API Client the access to the Resources.
5. The Autho4API Authorization Server of Service Provider X answers to the request in step 3 redirecting the Resource Owner's User Agent to the redirection URI provided in step 3; the Access Token is provided in the URI as an URI fragment. The Resource Owner's User Agent sends the corresponding HTTP GET request to the URI indicated in the Location header of received HTTP 302 response, without including the URI Fragment.
6. The (Shared) Autho4API Authorization Server returns a web page (typically an HTML document with an embedded script).
7. The Resource Owner's User Agent executes the script provided by the web-hosted Autho4API Authorization Server, which instructs the Resource Owner's User Agent to send the URI fragment to the web server and to be redirected to the URI received in the 'client_redirect_uri' query parameter, which matches the redirect_uri provided by the Autho4API Client in step 1.

    a) Resource Owner's User Agent sends to the (Shared) Autho4API Authorization Server the Access Token; (Shared) Autho4API Authorization Server caches the Access Token and associates it with Service Provider X.

    b) Resource Owner's User Agent sends the corresponding HTTP GET request to the URI indicated in the script. The GET request does not include the previous URI fragment, which is retained in the Resource Owner's User Agent.

    Note: This redirection cannot be done by a HTTP 302, but can be easily performed for example with a script in JavaScript.

8. The Autho4API Client returns a web page (typically an HTML document with an embedded script) capable of accessing the full redirection URI including the fragment retained by the Resource Owner's User-Agent, and extracting the Access Token (and other parameters) contained in the fragment.
9. The Resource Owner's User Agent executes the script provided by the web-hosted Autho4API Client resource locally, which extracts the Access Token and passes it to the Autho4API Client.

7.9.2.2.3 Accessing Protected Resources

For the access to protected resources, the following considerations apply:

- The Autho4API Access Control Server acting as entry point for resource requests, based on cached Access Token, forwards the resource request to the Autho4API Access Control Server of the Service Provider that issued the Access Token.
- Alternatively, the Autho4API Authorization Server acting as entry point for resource requests offers a Resource Server Redirection endpoint described in 7.9.2.2.3.1 where
    - The Autho4API Client sends a request including the issued Access Token.
    - The Autho4API Authorization Server acting as entry point, based on cached Access Token, answers to the request with the set of resource endpoints valid for the Access Token, i.e.: the endpoint(s) of the Autho4API Access Control Server(s) of the Service Provider that issued the Access Token.

7.9.2.2.3.1 Resource Server Redirection endpoint

This endpoint is offered by the Autho4API Authorization Server. Autho4API Client uses this endpoint to query for the endpoint(s) where the Service Provider is exposing the resources granted for an Access Token.

The offered endpoint exposes a REST resource in an URL that follows OMA REST Common Resource URL Considerations [REST_NetAPI_Common]. This resource is named 'URL Prefixes for Granted resources' and is defined in Appendix E.

The REST resource is accessed by Autho4API Client as follows:

- Autho4API Client sends a GET request, using HTTPS, to the Resource Server Redirection endpoint offered by Autho4API Authorization Server.
- Autho4API Client includes in the Authorization request header the Access Token obtained in the Obtaining Authorization step.
- The Autho4API Authorization Server checks the association of the received Access Token with an Autho4API Access Control Server (e.g.: the Autho4API Access Control Server of the corresponding Service Provider) and responds with a list of endpoints of that specific Autho4API Access Control Server and the resources accessible in each of the endpoints.
    - The endpoints returned are URL Prefixes referring to the server root where the resources are exposed, i.e.: the URL Prefix has the form: *hostname+port+base path.*

      Example: http://example.com/ServerX/ResourceExposingPlatform
    - The resources accessible in each of the returned endpoints are optionally matched by using 'scope' parameter indicated by the Autho4API Client in the Obtaining Authorization Step.
    - For accessing the actual resources, the Autho4API Client takes the provided URL Prefix and appends the needed parts. For OMA Network APIs this means that the URL Prefixes match the {serverRoot} variable.

7.10 Security considerations

This specification builds upon the security considerations described in section 10 of [RFC6749] and in section 4 of [RFC6750] and extends them by the following aspects:

- Authentication of Autho4API Client and Resource Owner to mitigate impersonation attacks
- Authentication of Autho4API Authorization Server to guarantee endpoints authenticity
- Secure transport of Authorization Grants, Access Tokens, Refresh tokens and client credentials
- Secure transport of the response to Authorization Request, when the secondary channel is used
- Client-side security challenges when opting for a specific secondary channel delivery mechanism
- Security challenges specific to some multi-Service Provider environments

These aspects are detailed in the sections respectively describing these features.

8. Release Information

8.1 Supporting File Document Listing

Doc Ref	Permanent Document Reference	Description
Supporting Files		
[AUTHO4API _SUP_Resource_Server]	OMA-SUP-XSD_rest_autho4api_url_prefixes_for_granted_resources-V1_0-20141209-A	XML Schema Definition for the resource "URL Prefixes for Granted Resources" Working file in Schema directory: file: rest_autho4api_url_prefixes_for_granted_resources-v1_0.xsd path: http://www.openmobilealliance.org/tech/profiles

Table 1: Listing of Supporting Documents in Autho4API Release

8.2 OMNA Considerations

Autho4API 1.0 includes the following OMNA items:

1. URN-based namespace identifiers
    a. urn:oma:xml:rest:autho:redirectEndpoint:1 (see section E.2.1)
2. OMNA registries
    a. OMNA Autho4API Scope Value registry (see section 7.3.1.2.2)

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-ER-Autho4API-V1_0-20141209-A	09 Dec 2014	Status changed to Approved by TP TP Ref # OMA-TP-2014-0275-INP_Autho4API_V1_0_ERP_for_final_Approval

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

B.1 ERDEF for Autho4API - Client Requirements

This section is normative.

Item	Feature / Application	Requirement
OMA-ERDEF-CLI-C-001-M	Autho4API Client	Autho4API-CLI-C: MCF

Table 2: ERDEF for Autho4API Client-side Requirements

B.2 ERDEF for Autho4API - Server Requirements

This section is normative.

Item	Feature / Application	Requirement
OMA-ERDEF-AUTH-S-001-M	Autho4API Authorization Server	Autho4API-AUTH-S: MSF
OMA-ERDEF-ACCESS-S-001-M	Autho4API Access Control Server	Autho4API-ACCESS-S: MSF

Table 3: ERDEF for Autho4API Server-side Requirements

B.3 SCR for Autho4API Client

Item	Function	Reference	Requirement
Autho4API-CLI-C-001-M	Support for client registration	7.1	Autho4API-CLI-C-002-O OR Autho4API-CLI-C-003-O
Autho4API-CLI-C-002-O	Support for public client type	7.1.1	
Autho4API-CLI-C-003-O	Support for confidential client type	7.1.1	Autho4API-CLI-C-020-O
Autho4API-CLI-C-004-M	Support for OAuth 2.0 Authorization Grant	7.2.2.1	Autho4API-CLI-C-005-O OR Autho4API-CLI-C-006-O OR Autho4API-CLI-C-007-O OR Autho4API-CLI-C-008-O
Autho4API-CLI-C-005-O	Support for Authorization Code flow	7.5.1	Autho4API-CLI-C-010-O OR Autho4API-CLI-C-012-O OR Autho4API-CLI-C-013-O
Autho4API-CLI-C-006-O	Support for Implicit Grant flow	7.5.2	Autho4API-CLI-C-019-O OR Autho4API-CLI-C-021-O OR Autho4API-CLI-C-022-O
Autho4API-CLI-C-007-O	Support for Resource Owner Password Credentials flow	7.5.3	
Autho4API-CLI-C-008-O	Support for Client Credentials flow	7.5.4	
Autho4API-CLI-C-010-O	Support for HTTP redirection to a public client HTTP URL for the delivery of response to Authorization Request (Authorization Code flow)	7.5.1.1	

Item	Function	Reference	Requirement
Autho4API-CLI-C-011-O	Support for HTTP redirection to a public client HTTP URL: confidentiality protection using TLS (Authorization Code flow)	7.2.1	
Autho4API-CLI-C-012-O	Support for HTTP redirection to a private client URI, for the delivery of response to Authorization Request (Authorization Code flow)	7.5.6.1	
Autho4API-CLI-C-013-O	Support for secondary channel for the delivery of response to Authorization Request (Authorization Code flow)	7.5.7	Autho4API-CLI-C-015-O OR Autho4API-CLI-C-016-O OR Autho4API-CLI-C-017-O OR Autho4API-CLI-C-018
Autho4API-CLI-C-014-O	Support for secondary channel: confidentiality protection using response encryption (Authorization Code flow)	7.5.7	
Autho4API-CLI-C-015-O	Support for secondary channel: response display with Resource Owner interaction (Authorization Code flow)	7.5.7	
Autho4API-CLI-C-016-O	Support for secondary channel: response delivery over textual SMS with Resource Owner interaction (Authorization Code flow)	7.5.7	
Autho4API-CLI-C-017-O	Support for secondary channel: response delivery to User Agent with automatic client retrieval (Authorization Code flow)	7.5.7	
Autho4API-CLI-C-018-O	Support for secondary channel: response delivery using OMA Connectionless Push over SMS (Authorization Code flow)	7.5.7	
Autho4API-CLI-C-019-O	Support for HTTP redirection to a public client HTTP URL for the delivery of response to Authorization Request (Implicit Grant flow)	7.5.2.1	
Autho4API-CLI-C-020-O	Support for HTTP redirection to a public client HTTP URL: confidentiality protection using TLS (Implicit Grant flow)	7.2.1	
Autho4API-CLI-C-021-O	Support for HTTP redirection to a private client URI, for the delivery of response to Authorization Request (Implicit Grant flow)	7.5.6.1	
Autho4API-CLI-C-022-O	Support for secondary channel for the delivery of response to Authorization Request (Implicit Grant flow)	7.5.7	Autho4API-CLI-C-024-O OR Autho4API-CLI-C-025-O OR Autho4API-CLI-C-026-O OR Autho4API-CLI-C-027

Item	Function	Reference	Requirement
Autho4API-CLI-C-023-O	Support for secondary channel: confidentiality protection using response encryption (Implicit Grant flow)	7.5.7	
Autho4API-CLI-C-024-O	Support for secondary channel: response display with Resource Owner interaction (Implicit Grant flow)	7.5.7	
Autho4API-CLI-C-025-O	Support for secondary channel: response delivery over textual SMS with Resource Owner interaction (Implicit Grant flow)	7.5.7	
Autho4API-CLI-C-026-O	Support for secondary channel: response delivery to User Agent with automatic client retrieval (Implicit Grant flow)	7.5.7	
Autho4API-CLI-C-027-O	Support for secondary channel: response delivery using OMA Connectionless Push over SMS (Implicit Grant flow)	7.5.7	
Autho4API-CLI-C-028-O	Support for Access Token Request: confidentiality protection using TLS	7.2.2	
Autho4API-CLI-C-029-O	Support for Access Token Request: Client authentication with Autho4API Authorization Server	7.1.2	
Autho4API-CLI-C-030-O	Client authentication using HTTP Basic and client password	7.1.2	
Autho4API-CLI-C-031-O	Support for the resolution of resource location from an issued Access Token	7.9.2.2.2	
Autho4API-CLI-C-032-M	Support for Bearer Access Token	7.8.3	
Autho4API-CLI-C-033-O	Support for Bearer Token sending in protected resource request: Authorization Request header sending method	7.8.3	
Autho4API-CLI-C-034-O	Support for Bearer Token sending in protected resource request: other sending method	7.8.3	
Autho4API-CLI-C-035-O	Support for Bearer Token sending in protected resource request: confidentiality protection using TLS	7.8.3	
Autho4API-CLI-C-036-O	Support for one-time Access Token	7.8.1.1	
Autho4API-CLI-C-037-O	Support for Refresh Token	7.7	
Autho4API-CLI-C-038-O	Support for Access Token and Refresh Token revocation	7.2.3	
Autho4API-CLI-C-039-O	Support for subscoping mechanism	7.4	

B.4 SCR for Autho4API Authorization Server

Item	Function	Reference	Requirement
Autho4API-AUTH-S-001-M	Support for client registration		Autho4API-AUTH-S-002-O OR Autho4API-AUTH-S-003-O
Autho4API-AUTH-S-002-M	Support for public client type	7.1.1	
Autho4API-AUTH-S-003-M	Support for confidential client type	7.1.1	Autho4API-AUTH-S-020-O
Autho4API-AUTH-S-004-M	Support for OAuth 2.0 Authorization Grant	7.2.2.1	Autho4API-AUTH-S-005-M AND Autho4API-AUTH-S-006-M
Autho4API-AUTH-S-005-M	Support for Authorization Code flow	7.5.1	Autho4API-AUTH-S-011-O
Autho4API-AUTH-S-006-M	Support for Implicit Grant flow	7.5.2	Autho4API-AUTH-S-020-O
Autho4API-AUTH-S-007-O	Support for Resource Owner Password Credentials flow	7.5.3	
Autho4API-AUTH-S-008-O	Support for Client Credentials flow	7.5.4	
Autho4API-AUTH-S-010-M	Support for Authorization Request: confidentiality protection using TLS	7.2.1	
Autho4API-AUTH-S-011-O	Support for HTTP redirection to a public client HTTP URL for the delivery of response to Authorization Request (Authorization Code flow)	7.2.1, 7.5.1.1	
Autho4API-AUTH-S-012-O	Support for HTTP redirection to a private client URI, for the delivery of response to Authorization Request (Authorization Code flow)	7.5.6.1	
Autho4API-AUTH-S-013-O	Support for secondary channel for the delivery of response to Authorization Request (Authorization Code flow)	7.5.7	(Autho4API-AUTH-S-016-O OR Autho4API-AUTH-S-017-O OR Autho4API-AUTH-S-018-O OR Autho4API-AUTH-S-019-O) AND Autho4API-AUTH-S-015-O
Autho4API-AUTH-S-014-M	Support for detecting the request to use the secondary channel, based on 'redirect_uri' parameter value (Authorization Code flow)	7.5.7.5	
Autho4API-AUTH-S-015-O	Support for secondary channel: confidentiality protection using response encryption (Authorization Code flow)	7.5.7	
Autho4API-AUTH-S-016-O	Support for secondary channel: response display with Resource Owner interaction (Authorization Code flow)	7.5.7	
Autho4API-AUTH-S-017-O	Support for secondary channel: response delivery over textual SMS with Resource Owner interaction (Authorization Code flow)	7.5.7	

Item	Function	Reference	Requirement
Autho4API-AUTH-S-018-O	Support for secondary channel: response delivery to User Agent with automatic client retrieval (Authorization Code flow)	7.5.7	
Autho4API-AUTH-S-019-O	Support for secondary channel: response delivery using OMA Connectionless Push over SMS (Authorization Code flow)	7.5.7	
Autho4API-AUTH-S-020-O	Support for HTTP redirection to a public client HTTP URL for the delivery of response to Authorization Request (Implicit Grant flow)	7.5.2.1	
Autho4API-AUTH-S-021-O	Support for HTTP redirection to a private client URI, for the delivery of response to Authorization Request (Implicit Grant flow)	TBD	
Autho4API-AUTH-S-022-O	Support for secondary channel for the delivery of response to Authorization Request (Implicit Grant flow)	7.5.7	(Autho4API-AUTH-S-025-O OR Autho4API-AUTH-S-026-O OR Autho4API-AUTH-S-027-O OR Autho4API-AUTH-S-028-O) AND Autho4API-AUTH-S-024-O
Autho4API-AUTH-S-023-M	Support for detecting the request to use the secondary channel, based on 'redirect_uri' parameter value (Implicit Grant flow)	7.5.7.5	
Autho4API-AUTH-S-024-O	Support for secondary channel: confidentiality protection using response encryption (Implicit Grant flow)	7.5.7	
Autho4API-AUTH-S-025-O	Support for secondary channel: response display with Resource Owner interaction (Implicit Grant flow)	7.5.7	
Autho4API-AUTH-S-026-O	Support for secondary channel: response delivery over textual SMS with Resource Owner interaction (Implicit Grant flow)	7.5.7	
Autho4API-AUTH-S-027-O	Support for secondary channel: response delivery to User Agent with automatic client retrieval (Implicit Grant flow)	7.5.7	
Autho4API-AUTH-S-028-O	Support for secondary channel: response delivery using OMA Connectionless Push over SMS (Implicit Grant flow)	7.5.7	
Autho4API-AUTH-S-029-M	Support for Access Token Request: confidentiality protection using TLS	7.2.2	

Item	Function	Reference	Requirement
Autho4API-AUTH-S-030-O	Support for Access Token Request: Client authentication with Autho4API Authorization Server	7.1.2	
Autho4API-AUTH-S-031-O	Client authentication using HTTP Basic and client password	7.1.2	
Autho4API-AUTH-S-032-O	Support for the resolution of resource location from an issued Access Token	7.9.2.2.2	
Autho4API-AUTH-S-033-M	Support for Bearer Access Token	7.8.3	
Autho4API-AUTH-S-034-O	Support for Refresh Token	7.7	
Autho4API-AUTH-S-035-O	Support for Access Token and Refresh Token revocation	7.2.3	
Autho4API-AUTH-S-036-O	Support for acting as a (Shared) Authorization Server	7.9.2.2, C.1	
Autho4API-AUTH -S-037-O	Support for subscoping mechanism	7.4	

B.5 SCR for Autho4API Access Control Server

Item	Function	Reference	Requirement
Autho4API-ACCESS-S-001-M	Support for Bearer Access Token	7.8.3	
Autho4API-ACCESS-S-002-M	Support for Bearer Token sending in protected resource request: Authorization Request header sending method	7.8.3	
Autho4API-ACCESS-S-003-O	Support for Bearer Token sending in protected resource request: other sending method	7.8.3	
Autho4API-ACCESS-S-004-O	Support for Bearer Token sending in protected resource request: confidentiality protection using TLS	7.8.3	
Autho4API-ACCESS-S-005-O	Support for one-time Access Token	7.8.1.1	
Autho4API-ACCESS-S-006-O	Support for acting as a (Shared) Access Control Server	7.9.2.2.3, C.1	
Autho4API-ACCESS-S-007-O	Support for subscoping mechanism	7.4	

Appendix C. Deployment Diagrams (Informative)

The following figures present possible deployment scenarios:

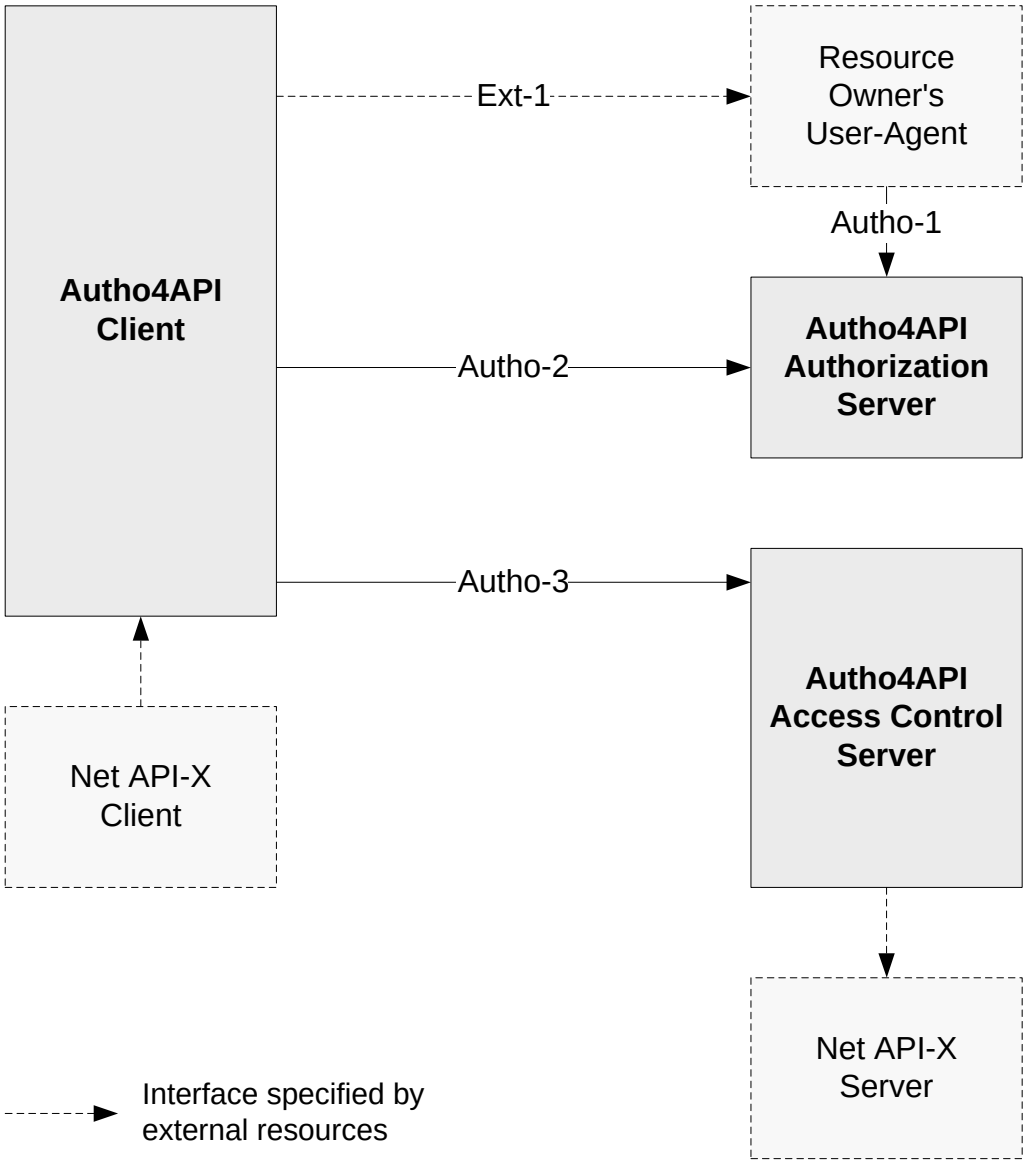

Figure 9: Autho-3 invoked by Autho4API Client

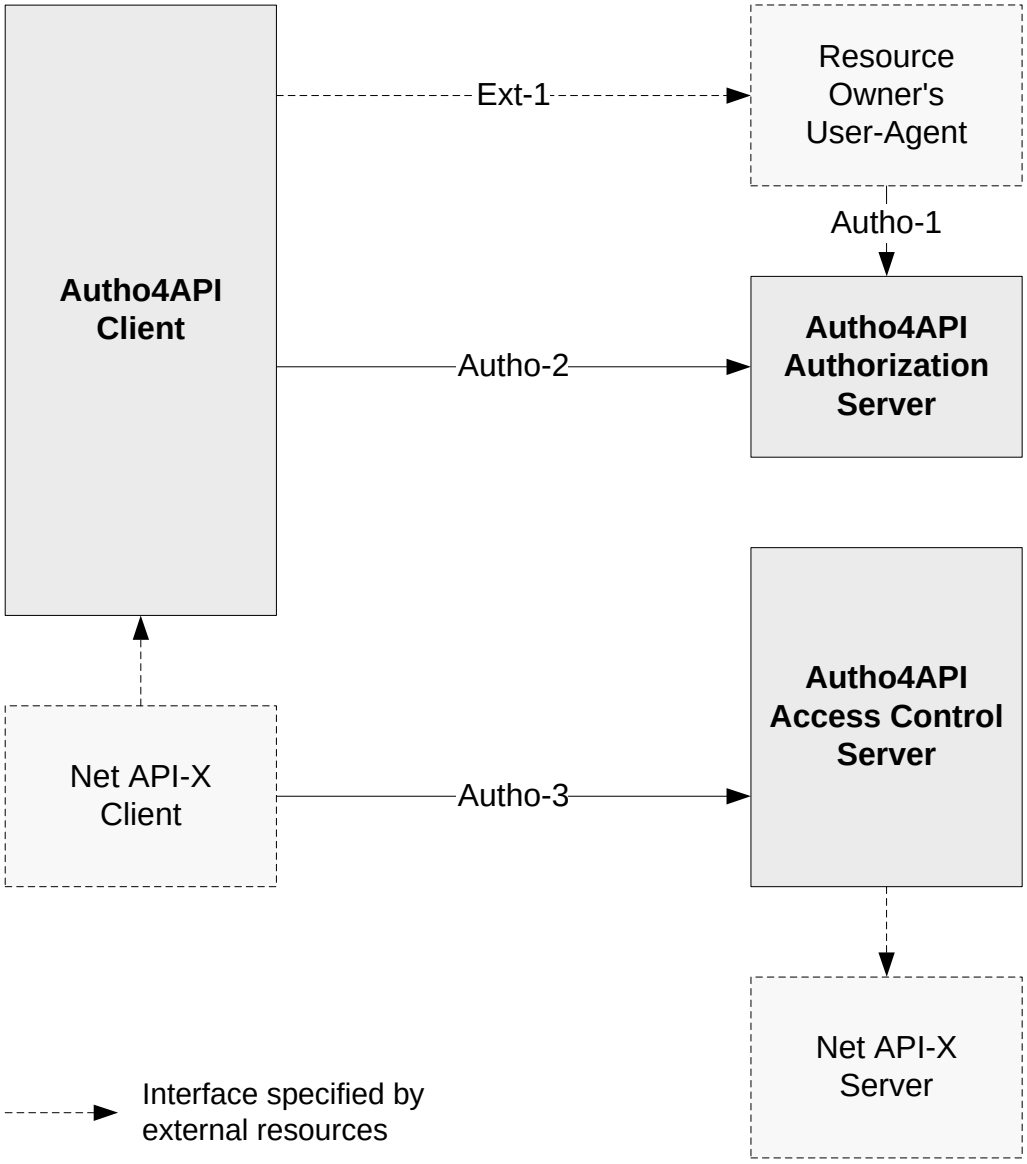

Figure 10: Autho-3 invoked by Net API-X Client

C.1 Multi Service Provider Scenario: Case Autho4API Client does not know the specific Service Provider

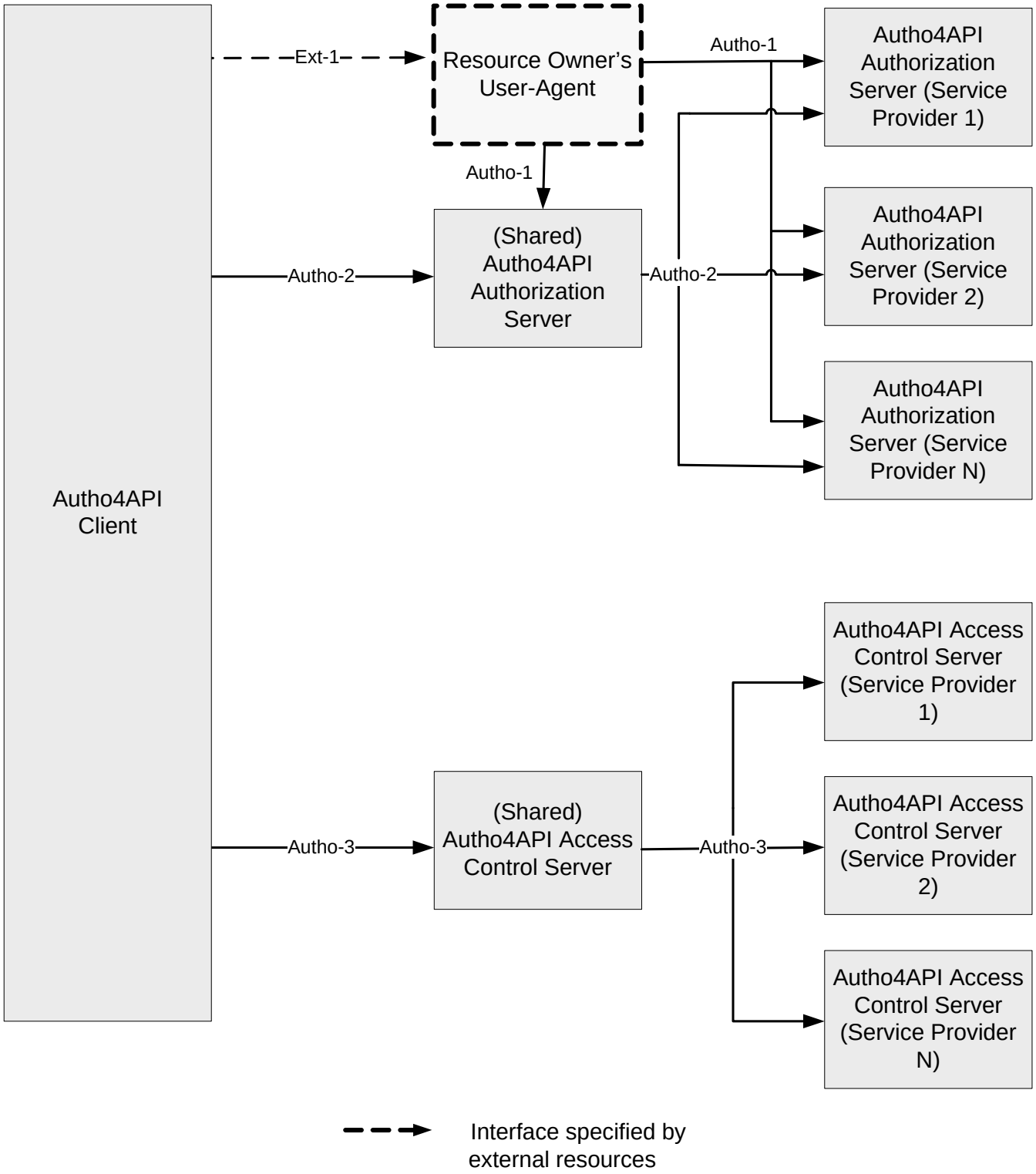

Figure 11: Multi Service Provider Scenario: Case Autho4API Client does not know the specific Service ProviderConsiderations for this deployment scenario

This scenario does not modify Autho4API Architecture described in section 6.2. As shown in the picture, in described scenario, the entities Autho4API Authorization Server and Autho4API Access Control Server will perform different functionalities among those described in section 6.3, as described below:

- **(Shared) Autho4API Authorization Server**: Is an Autho4API Authorization Server with limited functions, only those needed to support this scenario.
  - **Use of Autho-1 interface**: the (Shared) Autho4 API Authorization Server behaves as the Autho4API Authorization Server in communications with Resource Owner's User Agent with the consideration that it can redirect Resource Owner's User Agent to the Autho4API Authorization Server of a certain Service Provider, redirect it back to itself in order to cache the Authorization Code / Access Token and then redirect it back to the Autho4API Client.
  - **Use of Autho-2 interface**: the (Shared) Autho4API Authorization Server behaves as the Autho4API Authorization Server in communications with Autho4API Client and acts on behalf of Autho4API Client in communications with the Autho4API Authorization Server of a certain Service Provider.
- **Autho4API Authorization Server of a certain Service Provider:** behaves as a regular Autho4API Authorization Server.
- **(Shared) Autho4API Access Control Server**: Is an Autho4API Access Control Server with limited functions, only those needed to support this scenario.
  - **Use of Autho-3 interface**: the (Shared) Autho4API Access Control Server behaves as the Autho4API Access Control Server in communications with Autho4API Client and acts on behalf of Autho4API Client in communications with the Autho4API Access Control Server of a certain Service Provider.
- Autho4API Access Control Server of a certain Service Provider: behaves as a regular Autho4API Access Control Server.

This scenario assumes trusted relationship between (Shared) Autho4 API Authorization Server and the Autho4 API Authorization Servers of the different Service Providers involved and between (Shared) Autho4 API Access Control Server and the Autho4 API Access Control Servers of those Service Providers.

Appendix D. Defining Scope Values (Informative)

This appendix provides guidance on the definition of Scope Values for a Network API which uses the Autho4API authorization framework.

D.1 Summary

The definition of Scope Values for a given Network API should specify the following:

- Syntax and semantics of Scope Values;
- Scope Value registration procedure if applicable;
- Construction of the endpoint URL to which Autho4API Client sends the first protocol request (which is either Authorization Request on Autho-1 or Token Request on Autho-2 depending on the flow);
    - In particular if this endpoint URL is bound or not to a given Service Provider and/or Network API (organization, API, profile, version, etc.);
- Semantics of 'scope' parameter omitted in client request;
- Downscoping order of processing;
- Returned scope when 'scope' parameter includes multiple Scope Values, where at least one value is defined for one-time Access Token issuing;
- Relationship between Scope Value definitions (exclusive, inclusive, overlapping);
- Whether downscoping is supported, and if so how it applies in case of inclusive definitions;
- For each Scope Value:
    - Mapping to the authorized operations on resources of the Network API;
    - Whether this Scope Value is for the issuing of one-time Access Tokens;
    - Eventual recommendation with regard to expiry time.

D.2 Considerations

This section provides considerations on the task of defining Scope Values.

D.2.1 Expressing access scope

The actual access scope represented by an issued Access Token is functionally characterized by all the following:

- identity of Service Provider which exposes resources through the Network API;
- in the scope of this Service Provider, identity of Resource Owner;
- in the scope of this Service Provider, identity of Autho4API Client;
- Network API identification, characterized for instance by:
    - organization defining the API;
    - API itself (distinguished from other APIs defined by the organization);
    - profile of the API if applicable;
    - version.
- in the scope of this Network API, identification of the (REST operations on) resources.

One first element which can play a role in the definition of access scope is the endpoint URL to which the first protocol request is sent, which is:

- for the flows involving explicit Resource Owner's authorization (i.e. Authorization Code flow, Implicit Grant flow), the Authorization Request sent on Autho-1 to authorization endpoint URL; or
- for the other flows, the Token Request sent on Autho-2 to token endpoint URL.

This endpoint URL, simply called "endpoint URL" further in this appendix can be bound to some scoping information (such as Service Provider and Network API identification), binding which can be done for instance:

- Dynamically when the endpoint URL is obtained by Autho4API client through a discovery process where some scoping information like Service Provider or Network API identification is provided by client side or inferred by server side;
- Statically when the endpoint URL has a publicly documented construction, with URL components conveying Service Provider or network API identification, etc.

In addition, given the endpoint URL, the Autho4API Client can explicitly narrow down the access scope by including in the very first request of the protocol:

- the OAuth 'scope' parameter, listing a number of Scope Values;
- some endpoint extension parameters.

The more the endpoint URL is bound to access scope information, the less Scope Values need to convey access scope information. As opposite examples:

1. For an endpoint URL not bound to any scoping information at all (like a URL shared by multiple Service Providers exposing multiple APIs defined by multiple organizations), the first protocol request could convey via Scope Values and eventually extension parameters the identification of Service Provider and Network API (organization, API, version…).

2. For an endpoint URL bound to much scoping information (like a URL specific to one Service Provider exposing a single version of a unique API), the first protocol request would include simple Scope Values identifying resources without further context information. Ultimately for very simple APIs, Scope Values could even not be needed at all.

D.2.2 Syntax and semantics

D.2.2.1 Scope Value syntax

Scope Values are encoded as a space-delimited list in the scope parameter, which can be present as a percent-encoded value in:

- the Authorization Request;
- the Token Request;
- the Token Response of Implicit Grant flow;
- (for bearer tokens) in the unsuccessful response to a protected resource request, when Scope represented by the Access Token is insufficient.

The 'scope' parameter can also be present as a JSON string in:

- the Token Response of flows other than Implicit Grant flow.

Therefore there are theoretically no restrictions with regard to the set of characters usable to construct Scope Values, which can thus be:

- A free-text value;
- A text value conforming to a well-known textual format (URN, URI, JSON object, etc.).

Overlong Scope Values may cause transport problems though, especially when 'scope' parameter is carried as a URI query parameter (i.e. Authorization Request).

D.2.2.2 Scope Value semantics

[RFC6749] does not constrain the semantics of content conveyed in a Scope Value:

- Typically it consists of a string token identifying by convention a set of authorized operations on some network resources. It can also include information scoping the resources such as Network API identification.
- Alternatively, and to avoid the publication of string token semantics, a Scope Value could contain the URI identifying the resource on which authorized access is requested. But this technique has its downsides:
  - For an API exposing many resources, 'scope' parameter size could grow up very quickly (i.e. list of many Resource URIs);
  - On this Resource URI, granularity of access finer that "full access" cannot be granted.

D.2.3 Overlapping and conflicting definitions

D.2.3.1 Overlapping Scope Values

For a given API, the mapping between (operations on) resources and Scope Values can be:

- exclusive: each (operation on a) resource is mapped to one Scope Value at most;
- overlapping: some (operation on a) resource is mapped to several Scope Values;
- inclusive: some (operation on a) resource is mapped to several Scope Values, but in that case Scope Values have a strict superset/subset relationship with each other.

These mapping models are all usable, with the reservation that overlapping Scope Values present limitations with regard to downscoping functionality (see section D.2.5).

D.2.3.2 Conflicting Scope Values

Collision of conflicting Scope Values occur when at the same time:

- for different APIs (defined by same or different organizations), the same Scope Value is defined;
- for these different APIs, the same endpoint is serving the requests for authorization.

In this situation, the Autho4API Authorization Server could incorrectly grant access to resources not actually requested by the client. This inconsistent behavior could be exploited by malware clients.

D.2.3.3 API versioning

Another conflicting situation can arise when at the same time:

- different versions of the same API happen to be deployed in Autho4API clients and servers;
- some Scope Values are reused from one API version to another, with some changes of mapping between Scope Values and resources;
- Scope Values do not convey API version information;
- endpoint URLs are not bound to a specific API version.

In this situation, the Autho4API Authorization Server could grant access to less or more resources than those actually requested by the client.

D.2.4 Omission of requested Scope

The absence of Scope in the first protocol request can be given the following semantics:

1. It is not allowed, the Service Provider mandates the inclusion of 'scope' parameter on this endpoint URL;

2. It is allowed, and the requested Scope is implicitly understood as a basic access to some resources of the API, according to a documented convention;

3. It is allowed, and the requested Scope is implicitly understood as a full access to all resources of the API.

The allowed omission of Scope can coexist with well-defined Scope Values. As an example, in the requested conforming to case 3 above, downscoping could occur, i.e. the response could contain a 'scope' parameter listing well-known Scope Value(s).

The allowed omission of Scope likely constrains the endpoint URL to convey Scope information (e.g. dedicated URL for serving Authorization Requests of a single API).

D.2.5 Downscoping

D.2.5.1 Principles

Subsequently to the Autho4API Client request including a specific Scope, one or both of the following steps can take place:

- The Autho4API Authorization Server can narrow down the requested Scope, based on e.g. security considerations;
- The Resource Owner can narrow down the requested Scope, when prompted for authorization.

Depending on how Scope Values are defined, the downscoping operation can result for the Autho4API client in:

- The granting of less functional access, when each Scope Value maps to a functional group of resources;
- The granting of less privileged access, when each Scope Value maps to privileged-specific operations on the same resources.

As a result, enabling downscoping can help reduce the damages caused by leaked bearer tokens.

The Autho4API Authorization Server can signal to the client that downscoping has occurred, by including a 'scope' parameter in the response delivering the Access Token. This response is:

- For the Implicit Grant flow, the Token Response subsequent to Authorization Request;
- For the other flows, the Token Response subsequent to Token Request.

  Note: consequently for the Authorization Code flow, downscoping signaling is not done in the response to the Authorization Request (where requested Scope is included), but later on in the response to the Token Request.

D.2.5.2 Order of processing

Whether the Autho4API Authorization Server downscoping occurs before or after Resource Owner downscoping is affecting end-user experience. Taking the example where the Autho4API client requests "read write" access scope and the server intends to downscope to "read":

- The server downscopes before Resource Owner authorization; then the Resource Owner is prompted to authorize the Application for "read" access; or
- The Resource Owner authorizes the Application for "read write" access; then the server downscopes this authorization to "read" only, and the Resource Owner could later on wonder why the Application is only able to perform read-only access to the resource.

D.2.5.3 Returned Scope Values

[RFC6749] does not constrain the Scope resulting from downscoping to be a strict subset of the requested Scope Values. Downscoping is on the other hand constrained by the way Scope Values relate to each other. Specifically:

- In case of exclusive definitions, the narrower Scope can consist of a strict subset of the values listed in the request (e.g. "sms mms" is requested, and "sms" is returned);

- In case of inclusive definitions, the narrower Scope can consist of other values than those listed in the request (e.g. "messaging" is requested, and "sms" is returned);
- In case of overlapping definitions, the narrower Scope cannot be signaled and the Authorization Request has to be rejected.

In any case it is critical for the Autho4API client to understand the returned Scope, so to behave consistently with regard to further resource access. In particular, attempts to access to protected resources with an Access Token of insufficient Scope result in poor end-user experience (as the Resource Owner will be directed again for authorization) and unduly overloads Autho4API servers.

D.2.6 Characteristics of Scope Values

D.2.6.1 Expiry time management

By the inclusion of "expires_in" parameter in the Token Response, the Autho4API Authorization Server can optionally signal to Autho4API clients that the issued Access Token has a certain lifetime. Clients supporting this parameter can thus anticipate Access Token expiry and avoid sending resource requests containing expired tokens.

In addition, this Access Token has been issued for a given list of Scope Values negotiated by the Autho4API client during the authorization process, meaning that the same expiry time is attached to each of these "granted" Scope Values.

Finally, the Autho4API Authorization Server may have internally defined some computation rules for expiry time of issued Access Tokens, depending on for instance on:

- The type of Autho4API client (public or confidential);
- The type of Authorization Grant (implicit, Authorization Code, assertion….);
- The method of end-user authentication on Autho-1 (if applicable), as some are more secure than others;
- The method of client authentication on Autho-2 (if applicable);
- Whether this a first Access Token issuing, or an Access Token refresh;
- Resources and operations on these resources, meaning Scope Values.

Depending on Autho4API Authorization Server policy, each Access Token can be issued with variable expiry time depending on the list of Scope Values negotiated by Autho4API client during the authorization process. For example, if this list contains:

- Scope Values defined for critical access privileges, the Access Token would be issued with a short lifetime (e.g. a few minutes);
- Scope Values defined for non-critical access privileges, the Access Token would be issued with a long lifetime (e.g. a few weeks);
- Scope Values defined for critical access privileges *and* Scope Values defined for non-critical access privileges, the Access Token would likely be issued with a short lifetime (e.g. for security reasons).

The last point raises the issue that the Access Token by expiry will not be usable very soon even if the Autho4API client only intends later on to access the non-critical resources. Consequently:

- When refresh tokens are not usable (e.g. Implicit Grant flow) or not issued by the Autho4API Authorization Server, the Resource Owner will be frequently directed for authorization renewal;
- When Refresh Tokens are used, the Autho4API Authorization Server will be frequently requested to refresh the Access Token.

This issue is caused by a current limitation of the OAuth 2.0 protocol: an Authorization Grant can only be exchanged for one single Access Token.

A possible workaround for the Network API is to specify for each Scope Value some order of magnitude of Access Token lifetime, so to enable the Autho4API client to request one specific authorization per group of Scope Values of similar

lifetime. However as described earlier in this section, the final expiry time chosen by Autho4API Authorization Server may take into account other parameters than Scope Values.

D.2.6.2 One-time Access Tokens

By design, Access Tokens can be used multiple times till their expiry or revocation, and the OAuth 2.0 protocol does not define any Token Response parameter declaring a limited count of use for an issued Access Token.

It is possible though by documentation to explicitly indicate that a Scope Value is defined for the issuing of one-time Access Token only. Autho4API clients taking this indication into consideration can avoid attempting to use the obtained Access Token more than once.

In practice an Autho4API client would only list a single Scope Value in the 'scope' parameter, when this value is for one-time Access Token issuing. It is worth clarifying though what Scope the Autho4API Authorization Server would grant when the client lists in the 'scope' parameter:

- Several Scope Values defined for one-time Access Token;
- A Scope Value defined for one-time Access Token, and some Scope Values defined for multiple-time Access Token.

Some possible behaviors could be to reject the request, or else to downscope to one of the Scope Values defined for one-time Access Token.

D.2.7 Granularity of Scope Values

In terms of resource access granularity, Scope Value definition for a given Network API can theoretically range from no Scope Value defined at all, to one Scope Value defined per allowed operation on each resource of the API. In practice, the granularity needs to be a compromise:

- If access granularity is too coarse (i.e. too few Scope Values), the following may not be enabled:
  - Feature-driven downscoping, useful when the API consists of functionally distinct groups of resources;
  - Security-driven downscoping, useful when the API happens to define different access privileges on the same resources. In particular, the Autho4API Authorization Server may want to force security-driven downscoping for the Autho4API clients identified as "public" and not "confidential" (as per [RFC6749] definitions).
  - Fine-grained management of Access Token expiry time (see section D.2.6.1).
- If access granularity is too fine (i.e. too many Scope Values), then:
  - In the case where the Authorization Request prompted to Resource Owner is representing each Scope Value as one option to check on or off, the Resource Owner could be overwhelmed by user-interface complexity;
  - Autho4API client side development is complexified.

Appendix E. Definition of 'URL Prefixes for Granted Resources' resource (Informative)

This section is organized to support a comprehensive understanding of the 'URL Prefixes for Granted Resources' resource, defined following [REST_NetAPI_Common] recommendations. It specifies the definition of the resource, data structures, and definitions of all operations permitted on the resource.

E.1 Resource Summary

The following tables give a detailed overview of the resource 'URL Prefixes for Granted Resources', the data type of their representation and the allowed HTTP methods.

Purpose: Retrieving URL Prefixes for the Resources granted for a given Access Token

Resource	URL Base URL: https://{serverRoot}/autho4api/{version}	Data Structures	HTTP verbs			
			GET	PUT	POST	DELETE
URL Prefixes for Granted Resources	/resourcesURLPrefixes	RedirectEndpointList	read one or more URL prefixes for resources granted to an Access Token	no	no	no

E.2 Data Types

E.2.1 XML Namespaces

The namespace for the 'URL Prefixes for Granted Resources' resource is:

urn:oma:xml:rest:autho:redirectEndpoint:1

The 'xsd' namespace is used in the present document to refer to the XML Schema data types defined in XML Schema [XMLSchema1, XMLSchema2]. The 'common' namespace is used in the present document to refer to the data types defined in [REST_NetAPI_Common]. The use of the names 'xsd' and 'common' is not semantically significant.

The XML schema for the data structures defined in the section below is given in [AUTHO4API _SUP_Resource_Server].

Applications requiring the usage of Resource Server Redirection Endpoint defined in section 7.9.2.2.3.1 SHALL use the namespace urn:oma:xml:rest:autho:redirectEndpoint:1

E.2.2 Structures

The subsections of this section define the data structures used in the 'URL Prefixes for Granted Resources' resource.

Some of the structures can be instantiated as so-called root elements.

E.2.2.1 Type: RedirectEndpointList

List of Endpoints

Element	Type	Optional	Description
endpoint	Endpoint [1..unbounded]	No	An array of endpoints associated to provided Access Token

A root element named redirectEndpointList of type RedirectEndpointList is allowed in response bodies.

E.2.2.2 Type: Endpoint

Individual endpoint

Element	Type	Optional	Description
url	xsd:anyURI	No	URL Prefix referring to the server root where the resources are exposed, i.e.: the URL Prefix has the form: *hostname+port+base path*. Example: http://example.com/ServerX/ResourceExposingPlatform
scope	xsd:string [0..unbounded]	Yes	List of Scope Values referring to the resources granted for indicated Access token in the URL Prefix indicated in parameter above. If this parameter does not appear, it means that the URL Prefix indicated in parameter above exposes all the resources granted for provided Access Token. If more than one endpoint is provided, the 'scope' parameter SHALL appear in each of the endpoints.

E.3 Detailed specification of the resource

The following applies to the resource defined in this Appendix regardless of the representation format (i.e. XML, JSON):

- Reserved characters in URL variables (parts of a URL denoted below by a name in curly brackets) SHALL be percent-encoded according to [RFC3986]. Note that this always applies, no matter whether the URL is used as a Request URL or inside the representation of a resource (such as in "endpoint" elements).
- For responses that have a body, the following applies: The Server SHALL return either JSON or XML encoded parameters in the response body, according to the result of the content type negotiation as specified in [REST_NetAPI_Common]. The generation and handling of the JSON representations SHALL follow the rules for JSON encoding in HTTP Requests/Responses as specified in [REST_ NetAPI_Common].

E.3.1 Resource: 'URL Prefixes for Granted Resources'

The resource used is:

https://{serverRoot}/autho4api/{version}/resourcesURLPrefixes/

This resource is for polling for the endpoint(s) where the Service Provider is exposing the resources granted for an Access Token.

E.3.1.1 Request URL variables

The following request URL variables are common for all HTTP commands:

Name	Description
serverRoot	server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI Note: Please notice that serverRoot will be the same as the serverRoot for Authorization and Token endpoints defined in chapters 7.2.1 and 7.2.2, respectively.
version	version of the API client wants to use. The value of this variable is v1.

See section E.3 for a statement on the escaping of reserved characters in URL variables.

E.3.1.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

When the request provides an expired, revoked, malformed, or invalid Access Token, the response shall be an HTTP 401 (Unauthorized) with the error code 'invalid_token', as defined for Bearer Tokens in section 3.1 of [RFC6750]

E.3.1.3 GET

This operation is used for reliable retrieval of the endpoint(s) where the Service Provider is exposing the resources granted for an Access Token

Request URL parameters are: none

The request for polling for the endpoint must include the Access Token, in the same manner as when Accessing Protected Resources, as specified in section 7.8.

E.3.1.3.1 Examples 1: Retrieve endpoint(s) for granted resources for a given Access Token. XML requested

E.3.1.3.1.1 Request

The following is an example of a request to this endpoint:

```
GET exampleAPI/autho4api/v1/resourcesURLPrefixes HTTP/1.1
Accept: application/xml
Host: server.example.com
Authorization: Bearer vF9dft4qmT
```

E.3.1.3.1.2 Response

The following is an example of a valid response from this endpoint to previous request:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: nnnn
Date: Thu, 18 Nov 2011 02:51:59 GMT
```

```
<?xml version="1.0" encoding="UTF-8"?>
<autho:redirectEndpointList xmlns:autho="urn:oma:xml:rest:autho:redirectEndpoint:1">
        <endpoint>
                <url>https://Autho4APIAccessControlServer_1.serviceproviderExample.com</url>
                <scope>oma_rest_messaging.in_regist</scope>
                <scope>oma_rest_messaging.in_subscr</scope>
        </endpoint>
        <endpoint>
                <url>https://Autho4APIAccessControlServer_2.serviceproviderExample.com</url>
                <scope>oma_rest_messaging.out</scope>
        </endpoint>
</autho:redirectEndpointList>
```

E.3.1.3.2 Examples 2: Retrieve endpoint(s) for granted resources for a given invalid Access Token

E.3.1.3.2.1 Request

The following is an example of a request to this endpoint:

```
GET exampleAPI/autho4api/v1/resourcesURLPrefixes HTTP/1.1
Accept: application/xml
Host: server.example.com
Authorization: Bearer FakevF9dft4qmT
```

E.3.1.3.2.2 Response

The following is an example of a valid response from this endpoint to previous request:

```
HTTP/1.1 401 Unauthorized
Date: Thu, 18 Nov 2011 02:51:59 GMT


WWW-Authenticate: Bearer realm=" server.example.com ",
        error="invalid_token",
        error_description="The Access Token is not valid"
```

E.3.1.3.3 Examples 3: Retrieve endpoint(s) for granted resources for a given Access Token. JSON requested

E.3.1.3.3.1 Request

The following is an example of a request to this endpoint:

```
GET exampleAPI/autho4api/v1/resourcesURLPrefixes HTTP/1.1
Accept: application/json
Host: server.example.com
Authorization: Bearer vF9dft4qmT
```

E.3.1.3.3.2 Response

The following is an example of a valid response from this endpoint to previous request:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
Content-Length: nnnn
Date: Thu, 18 Nov 2011 02:51:59 GMT

{"redirectEndpointList": {"endpoint": [
   {
      "scope": [
         "oma_rest_messaging.in_regist",
         "oma_rest_messaging.in_subscr"
      ],
      "url": "https://Autho4APIAccessControlServer_1.serviceproviderExample.com"
   },
   {
      "scope": "oma_rest_messaging.out",
      "url": "https://Autho4APIAccessControlServer_2.serviceproviderExample.com"
   }
]}}
```

E.3.1.4 PUT

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET’ field in the response as per section 14.7 of [RFC 2616].

E.3.1.5 POST

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET’ field in the response as per section 14.7 of [RFC 2616].

E.3.1.6 DELETE

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: GET’ field in the response as per section 14.7 of [RFC 2616].