OAuth 2.0授权框架

1 OAuth 2.0授权框架

摘要：
OAuth 2.0授权框架允许第三方应用程序通过如下任意一种方式获取有限制的访问：

1. 第三方应用代表资源拥有者发起在资源拥有者和HTTP服务之间的互动。
2. 第三方应用通过其身份来获取访问权限。

本文取代并淘汰了在RFC 5849中所描述的的OAuth 1.0协议。

本备忘录的状态：
这是Internet标准跟踪文档。

本文档是Internet工程任务组（IETF）的产品。它代表了IETF社区的共识。它有获得公众审查，并已获得`互联网工程指导小组（IESG）`批准发布。有关Internet标准的更多信息，请参见RFC 5741的第2节。
有关本文档当前状态，任何勘误以及如何提供反馈的信息，请访问http://www.rfc-editor.org/info/rfc6749。

1.1 1. Introduction

在传统的C/S身份验证模型中，客户端通过使用资源所有者的凭据向服务器进行身份验证来请求服务器上的访问受限资源（受保护资源）。 为了向第三方应用程序能够访问受限资源，资源所有者需要与第三方共享其凭据。 这会产生一些问题和局限：

- 为了将来的需要，第三方程序需要存储资源拥有者的凭据（通常为明文密码）
- 即使密码验证存在安全漏洞，服务器仍然需要支持它
- 第三方程序对资源拥有者的受保护资源拥有过于宽泛的权限，同时资源拥有者也没有能力对第三方程序进行限制（如限制第三方程序仅访问部分资源，或限制第三方程序的访问时间等）
- 资源拥有者必须通过更改密码来撤销第三方应用的权限。并且不能对单个第三方应用撤权（一旦更改密码，所有之前授予权限的第三方应用程序都要重新授权）
- 任意第三方应用的泄密都会导致终端用户的密码和受该密码保护的所有数据泄密。

OAuth通过引入授权层并将客户端的角色与资源所有者的角色分开来解决这些问题。在OAuth中，客户端请求访问由资源所有者拥有并由资源服务器托管的资源，并向其颁发与资源所有者不同的凭据集。

客户端通过获取访问令牌而不是使用资源拥有者的凭据来访问受限资源。访问令牌是一个表示特定范围，生命周期和其他访问属性的字符串。授权服务器在资源所有者的批准下才会向第三方客户端颁发访问令牌。客户端使用访问令牌来访问资源服务器托管的受保护资源。

例如，一个终端用户（资源拥有者）可以授权打印服务（客户端）访问存储在照片共享服务（资源服务器）中的受保护照片，而无需与打印服务共享其用户名和密码。相反，她直接使用照片共享服务信任的服务器（授权服务器）进行身份验证，该服务器向打印服务发放特定凭证（访问令牌）。

此规范旨在与HTTP（[RFC2616]）一起使用。在HTTP之外的任何协议上使用OAuth都超出了本规范的范围。

作为信息文档发布的OAuth 1.0协议（[RFC5849]）是一个小型临时社区工作的结果。此标准跟踪规范建立在OAuth 1.0部署经验的基础上，以及从更广泛的IETF社区收集的其他用例和可扩展性要求。
OAuth 2.0 协议与OAuth 1.0不向后兼容。这两个版本可以在网络上共存，并且实现可以选择支持两者。但是，本规范的目的是所有的新系统均采用OAuth2.0，OAuth1.0仅用于支持已经部署的系统。
OAuth 2.0协议与OAuth 1.0协议共享的实现细节很少，所以熟悉OAuth 1.0的实施者不应该对本规范的结构和细节进行臆测。

1.1.1 1.1 角色

OAuth定义了四个角色：

- **资源所有者（resource owner）：** 能够对受保护资源授予访问权限的实体。当资源所有者是一个人时，它被称为终端用户。
- **资源服务器（resource server）：** 托管受保护资源的服务器，能够接受和响应通过令牌对受保护的资源的请求。
- **客户端（client）：** 代表资源所有者及其授权进行受保护资源请求的应用程序。术语"客户端"并不暗示任何特定的实现特征（例如，应用程序是在服务器，台式机还是其他设备上执行）。
- **授权服务器（authorization server）：** 成功后，服务器向客户端发出访问令牌验证资源所有者并获得授权。

授权服务器和资源服务器之间的交互超出了本规范的范围。授权服务器可以是与资源服务器相同的服务器或单独的实体。单个授权服务器可以发出可以被多个资源服务器接受的访问令牌。

1.1.2 1.2 Protocol Flow

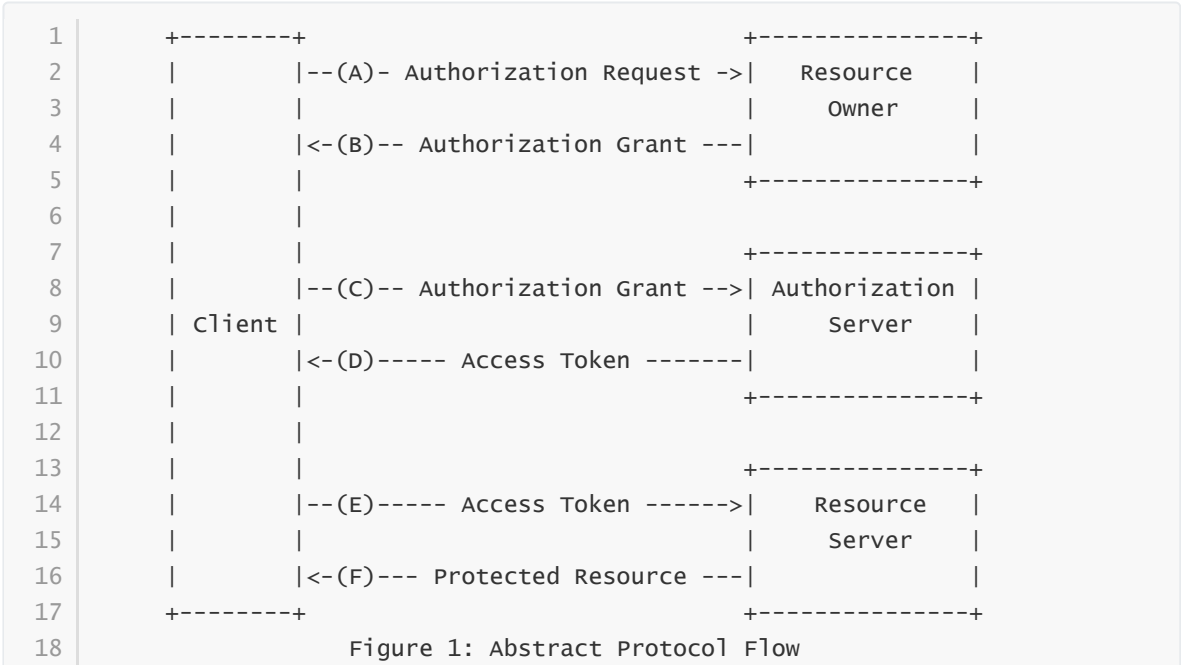

Figure 1: Abstract Protocol Flow

Figure 1中所示的抽象OAuth 2.0流程描述了四个角色之间的交互。包括以下步骤：
（A）客户端请求资源所有者授权。该授权请求可以直接直接呈现给资源拥有者，也可间接地通过授权服务器进行（例如跳转到授权服务器）。

（B）客户端收到授权许可，即表示资源所有者授权的凭证，使用本规范中定义的四种授权类型之一或使用扩展授权类型表示。授权许可类型取决于客户端使用何种方法请求授权服务器，以及授权服务器支持哪些授权类型。

（C）客户端通过向授权服务器进行认证并呈现用户赋予的权限来请求`access token`。

（D）授权服务器验证客户端并验证用户赋予的权限，如果有效，则颁发`access token`。

（E）客户端从资源服务器请求受保护资源，并通过呈现`access token`进行身份验证。

（F）资源服务器验证`access token`，如果有效，则为该请求提供服务。

客户端从资源所有者获得授权授权的首选方法（如步骤（A）和（B）所示）是使用授权服务器作为中介，如第4.1节中的图3所示。

1.1.3 1.3. Authorization Grant

权限授予（ Authorization Grant）是资源拥有者同意授权请求（访问受保护资源）的凭据，客户端可以用它来获取`access token`。 本规范定义了四种授权(grant)类型 - `授权码模式(authorization code)`，`简化模式（implicit）`，`密码模式（resource owner password credentials）`和`客户端模式（client credentials）`，以及用于定义其他类型的`可扩展性机制`。

> 这个 Authorization(n.) Grant(v.)不太好翻译，可以理解为权限授予，授权授予
> 本节仅对这四种授权模式进行简单介绍，将在第4节对这些模式进行详细的介绍

1.1.3.1 1.3.1 授权码模式（Authorization Code）

授权码是通过授权服务器来获得的，授权服务器是客户端和资源拥有者之间的媒介。与客户端直接向资源拥有者申请权限不同，客户端通过将资源拥有者引向授权服务器（通过[RFC2616]中定义的`user-agent`），然后授权服务器反过来将资源拥有者redirect到client（附带上`authorization code`）。

在将资源拥有者redirect到client（附带`authorization code`）之前，授权服务器验证资源拥有者并获取授权。因为资源拥有者仅与授权服务器进行身份验证，所以资源拥有者的凭据（用户名、密码等）永远不会泄露给客户端（尤其是第三方客户端）。

授权码有一些重要的安全优势，比如验证client的能力，比如直接将`access token`传送给client而不是通过资源拥有者的`user-agent`（可能会将token泄露给第三方）。

1.1.3.2 1.3.2 简化模式（implicit）

简化模式是为在浏览器中使用诸如JavaScript之类的脚本语言而优化的一种简化的授权码流程。在简化模式中，直接将`access token`而不是`authorization code`颁发给client（通过资源拥有者的授权）。grant类型为implicit，所以没有中间环节（比如用来在稍后获取`access token`的`authorization code`）

在简化模式中颁发`access token`时，授权服务器没有对client进行验证。在某些情况下，可以通过用来获取`access token`的重定向URI来验证client。`access token`可以通过访问资源拥有者的`user-agent`暴露给资源拥有者或者其他的应用。

由于简化模式减少了获取`access token`的往返次数，所以可以提高某些客户端的响应能力和效率（比如一个运行在浏览器中的应用）。但是，应该权衡使用简化模式所带来的便捷性与其带来的安全隐患之间的利害关系（在10.3和10.16中有描述），尤其是授权码模式可用时。

1.1.3.3 1.3.3 密码模式（resource owner password credentials）

资源拥有者密码凭据（如用户名和密码）可以用来直接用来当做一种获取`access token`的权限授予方式。凭据仅应当在资源拥有者高度信任client时使用（比如，应用是设备操作系统的一部分，或有较高权限的应用），并且其他授权模式（比如授权码模式）不可用时。

尽管这种授权类型需要client直接接触资源拥有者的凭据，资源拥有者的凭据仅被用于单次的获取`access token`的请求。通过使用用户凭据来交换具有较长寿命的`access token`或者`refresh token`，这种授权模式可消除client在将来需要授权时对资源拥有者凭据的需求（就是说，这次通过用户凭据获取了`access token`，以后就可以直接通过`access token`而不是用户凭据来访问受限资源了）。

1.1.3.4 1.3.4 客户端模式（client credentials）

当授权范围限于客户端控制下的受保护资源或先前与授权服务器一起安排的受保护资源时，client凭据（或其他形式的客户端身份验证）可用作权限授予。客户端凭证通常是在客户端代表自己（客户端也是资源所有者）或基于先前与授权服务器一起安排的授权请求访问受保护资源时用作权限授予。

1.1.4 1.4 Access Token

`access token`是用来访问受限资源的凭据。`access token`是一个代表授予client的权限的字符串。该字符串通常对client不透明。token表示特定范围和持续时间的访问权限，由资源所有者授予，由资源服务器和授权服务器执行。

令牌可以表示用于检索授权信息的标识符，或者可以以可验证的方式自包含授权信息（即，由一些数据和签名组成的令牌串）。client可能需要额外的身份验证凭据（超出本规范的范围）来使用令牌。

访问令牌提供一个使用单个的资源服务器可以理解的令牌来替换其他不同的身份验证方式（如用户名+密码方式）的抽象层。这种抽象使得颁发访问令牌比用于获取它们的权限授予更具限制性，并且消除了资源服务器理解各种不同身份验证方法的需要。

访问令牌可以具有基于资源服务器安全性要求的不同格式，结构和使用方法（例如，加密属性）。访问令牌属性和用于访问受保护资源的方法超出了本规范的范围，并由协同规范（如[RFC6750]）定义。

1.1.5 1.5. Refresh Token

refresh token是用于获取access token的凭据。refresh token由授权服务器颁发给client，用于在当前访问令牌变为无效或过期时获取新的访问令牌，或者获取具有相同或更窄范围的其他访问令牌（访问令牌可能具有更短的生命周期和权限少于资源所有者授权的权限。根据授权服务器的判断，发出刷新令牌是可选的。如果授权服务器发出刷新令牌，则在发出访问令牌时包括它（即图1中的步骤（D））。

刷新令牌是表示资源所有者授予客户端的权限的字符串。该字符串通常对客户端不透明。令牌表示用于检索 授权信息的标识符。与访问令牌不同，刷新令牌仅用于授权服务器，不会发送到资源服务器。

```
+--------+                                       +----------------+
|        |--(A)------- Authorization Grant --------->|                |
|        |                                       |                |
|        |<-(B)----------- Access Token -------------|                |
|        |               & Refresh Token             |                |
|        |                                       |                |
|        |                            +----------+   |                |
|        |--(C)---- Access Token ---->|          |   |                |
|        |                            |          |   |                |
|        |<-(D)- Protected Resource --| Resource |   | Authorization |
| Client |                            |  Server  |   |     Server     |
|        |--(E)---- Access Token ---->|          |   |                |
|        |                            |          |   |                |
|        |<-(F)- Invalid Token Error -|          |   |                |
|        |                            +----------+   |                |
|        |                                       |                |
```

```
   |        |                                           |               |
   |        |--(G)----------- Refresh Token ----------->|               |
   |        |                                           |               |
   |        |<-(H)----------- Access Token -------------|               |
   +--------+           & Optional Refresh Token        +---------------+

                Figure 2: Refreshing an Expired Access Token
```

图2所示的流程包括以下步骤：

（A）客户端通过向授权服务器进行认证、发起权限授予来获取access token。
（B）授权服务器验证客户端并验证权限授予授权，如果有效，则颁发访问令牌和刷新令牌。
（C）客户端通过呈现访问令牌向资源服务器发出受保护的资源请求。
（D）资源服务器验证访问令牌，如果有效，则为请求提供服务。
（E）重复步骤（C）和（D）直到访问令牌到期。如果客户端知道访问令牌已过期，则跳到步骤（G）；否则，它会生成另一个受保护的资源请求
（F）由于访问令牌无效，资源服务器返回无效的令牌错误。
（G）客户端通过向授权服务器进行身份验证并显示刷新令牌来请求新的访问令牌。该客户端身份验证的要求是基于客户端类型和授权服务器策略。
（H）授权服务器验证客户端并验证刷新令牌，如果有效，则发出新的访问令牌（以及可选的新刷新令牌）。

1.1.6 1.6. TLS Version

由于广泛的部署和已知的安全性漏洞，当本规范使用安全传输层协议（TLS）时可能存在不同的适用版本。在本协议发表时，TLS v1.2[RFC5246]是最新版本，但是部署基础非常有限，可能无法实现。TLS v1.0 [RFC2246]是最广泛的部署版本并将提供最广泛的互操作性。
实现还可以支持满足其安全要求的其他传输层安全机制。

1.1.7 1.7. HTTP Redirections

在client话说授权服务器将user-agent导向另一个目的地时，本规范广泛地使用了HTTP重定向。虽然本规范中的示例使用HTTP 302状态代码进行重定向，但是允许其他的实现通过其他方法实现重定向，这也被认为是实现细节的一部分。

1.1.8 1.8. Interoperability

OAuth 2.0提供了一个具有明确定义的具有丰富的安全属性的授权框架。但是，作为一个具有许多可选组件的丰富且高度可扩展的框架，该规范本身可能会产生各种不可互操作的实现。

此外，对于一些组件，本规范仅有部分定义或完全未定义（例如，客户端注册，授权服务器功能，endpoint发现）。如果没有这些组件，客户端必须专门手动地针对特定授权服务器和资源服务器进行配置以进行互操作。

该框架的设计明确期望未来的工作将定义实现完整的Web级互操作性所必需的规范性配置文件和扩展。

1.1.9 1.9. Notational Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”,“SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this specification are to be interpreted as described in [RFC2119].

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234]. Additionally, the rule URI-reference is included from “Uniform Resource Identifier (URI): Generic Syntax” [RFC3986].

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, “attack”, “authentication”, “authorization”, “certificate”, “confidentiality”, “credential”, “encryption”, “identity”, “sign”, “signature”, “trust”, “validate”, and “verify”.

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

> 这几个词在翻译的时候已尽量按照其含义进行翻译，但没有在文中进行标注（如加粗等），留坑等以后填吧。。

1.2 2. Client Registration

在启动协议之前，client向授权服务器注册。client注册的方式使用授权服务器超出了本规范的范围，但通常涉及终端用户与HTML注册表单的交互。

客户端注册不需要客户端和授权服务器之间的直接交互。当授权服务器支持时，注册可以依赖于其他方式来建立信任并获得所需的客户端属性（例如，重定向URI，客户端类型）。例如，可以使用自发布或第三方发布的断言来完成注册，或者通过使用 可信通道执行客户端发现的授权服务器来完成注册。

注册客户端时，客户端开发人员应该：

- 指定如第2.1节所述的客户端类型，
- 提供如第3.1.2节所述的client重定向URI， 以及
- 包含授权服务器所需的任何其他信息（例如，应用程序名称，网站，描述，徽标图像，所接受的法律条款）。

1.2.1 2.1. Client Types

OAuth根据其与授权服务器进行安全身份验证的能力定义了两种客户端类型（即，保证其客户凭证的机密性的能力）：

- **机密：** 客户端能够维护其凭证的机密性（例如，在具有对客户端凭证具有受限访问的安全服务器上实现的客户端），或能够使用其他方式进行安全的客户端认证。
- **公开：** 客户端无法维护其凭据的机密性（例如，在资源所有者使用的设备上执行的客户端，例如已安装的本机应用程序或基于Web 浏览器的应用程序），并且无法通过任何其他方式进行安全的客户端身份验证。

客户端类型标识基于授权服务器的安全身份验证定义及其可接受的客户端凭据暴露级别。授权服务器不应该对客户端类型做出假设。

客户端可以被实现为分布式组件集，每个组件具有不同的客户端类型和安全性上下文（例如，具有基于机密服务器的组件和基于公共浏览器的组件的分布式客户端）。如果授权服务器不提供对此类客户端的支持或不提供有关其注册的指导，则客户端应该将每个组件注册为单独的客户端。

此规范是围绕以下客户端配置设计的：

- **Web应用程序** Web应用程序是在Web 服务器上运行的机密客户端。资源所有者通过在资源所有者使用的设备上的用户代理中呈现的HTML用户界面来访问客户端。客户端凭据以及发布到客户端的任何访问令牌都存储在Web服务器上，不会向资源所有者公开或访问。
- **基于用户代理的应用程序** 基于用户代理的应用程序是公共客户端，其中客户端代码从web服务器下载并在资源所有者使用的设备上的用户代理（例如，web浏览器）内执行。协议数据和凭证可以轻松访问（并且通常可见）资源所有者。由于此类应用程序驻留在用户代理中，因此它们可以在请求授权时无缝使用用户代理功能。
- **本机应用程序** 本机应用程序是在资源所有者使用的设备上安装和执行的公共客户端。资源所有者可以访问协议数据和凭证。这是假设的可以提取应用程序中包含的任何客户端身份验证凭据。另一方面，动态发布的凭证（例如访问令牌或刷新令牌）可以获得可接受的保护级别。至少，这些凭据

受到保护，从而免受应用程序可能与之交互的恶意服务器的影响。在某些平台上，可能会保护这些凭据免受驻留在同一设备上的其他应用程序的影响。

1.2.2 2.2. Client Identifier

授权服务器向已注册的client颁发client identifier–一个代表该client注册信息的唯一字符串。client identifier不需要保密，它被暴露给资源拥有者并且禁止单独用于client认证。客户端标识符对于授权服务器是唯一的。
本规范未定义client identifier字符串的大小。客户端应避免对标识符大小进行假设。授权服务器应该记录它发出的任何标识符的大小。

1.2.3 2.3. Client Authentication

如果客户端类型是机密的，则客户端和授权服务器建立适合授权服务器的安全性要求的客户端认证方法。授权服务器可以接受满足其安全要求的任何形式的客户端身份验证。

机密客户端通常被颁发（或建立）用于与授权服务器进行认证的一组客户机凭证（例如，密码，公钥/私钥对）。

授权服务器可以与公共客户端建立客户端身份验证方法。但是，授权服务器不得依赖公共客户端身份验证来识别客户端。

客户端在每个请求中最多使用一种身份验证方法。

1.2.3.1 2.3.1. Client Password

拥有客户端密码的客户端可以使用[RFC2617]中定义的HTTP Basic身份验证方案向授权服务器进行身份验证。使用附录B中的`application/x-www-form-urlencoded`编码算法对客户端标识符进行编码，并将编码value用作username;客户端密码使用相同的算法进行编码并用作password。授权服务器必须支持HTTP基本身份验证方案，以便对发出客户端密码的客户端进行身份验证。

例如：`Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3`

或者，授权服务器可以选择支持在请求体中包含如下参数的客户端凭据：

- `client_id`： REQUIRED。在2.2节描述的注册过程中发给客户端的客户端标识符。
- `client_secret`： REQUIRED. The client secret. 如果客户端密钥是空字符串，则客户端可以省略该参数。

使用这两个参数在请求体中包含客户端凭证是不推荐的，并且应该仅限于无法直接使用HTTP基本身份验证方案（或其他基于密码的HTTP身份验证方案）的客户端。参数只能在请求体中传输，绝不能包含在请求URI中。

例如，使用body参数刷新访问令牌（第6节）的HTTP请求（额外换行符仅用于排版目的）：

```
    POST /token HTTP/1.1
    Host: server.example.com
    Content-Type: application/x-www-form-urlencoded

    grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
    &client_id=s6BhdRkqt3&client_secret=7FjfpOZBr1KtDRbnfVdmIw
```

当使用密码验证发送请求时，授权服务器必须要求使用如1.6节所述的TLS。
由于此客户端身份验证方法涉及密码，因此授权服务器必须保护使用它的任何endpoint 免受穷举攻击。

1.2.3.2 2.3.2. Other Authentication Methods

授权服务器可以支持符合其安全要求的任何合适的HTTP认证方案。使用其他身份验证方法时，授权服务器必须定义客户端标识符（注册记录）和身份验证方案之间的映射。

1.2.4 2.4. Unregistered Clients

此规范不排除使用未注册的客户端。但是，此类客户端的使用超出了本规范的范围，需要进行额外的安全性分析并检查其互操作性影响。

1.3 3. Protocol Endpoints

授权过程使用两个授权服务器端点（HTTP资源）：

- **Authorization endpoint:** 客户端使用该端点通过用户代理重定向从资源所有者获取授权。
- **Token endpoint:** 客户端用于通过user-agent redirection从资源所有者获取授权。

以及一个客户端端点：

- **Redirection endpoint** 授权服务器用于通过资源所有者 user-agent将包含授权凭据的响应返回给客户端。

并非每种授权授权类型都使用两个端点。
扩展授权类型可以根据需要定义其他端点。

1.3.1 3.1. Authorization Endpoint

授权终端用于与资源所有者交互并获得权限授予。 授权服务器必须首先验证资源所有者的身份。 授权服务器验证资源所有者的方式（例如，用户名和密码登录，会话cookie）超出了本规范的范围。

客户端获取授权端点位置的方法超出了本规范的范围，因为这个位置通常由服务文档提供。

端点URI可以包括`application/x-www-form-urlencoded`格式（根据附录B）的查询组件（[RFC3986]第3.4节），并且在添加其他查询参数时该组件必须保留。终端URI绝不能包含片段组件。

由于对授权端点的请求导致用户身份验证和凭据的明文传输（在HTTP响应中），在向授权端点发送请求时，授权服务器必须使用第1.6节中所述的TLS。对于没有值的参数，必须当作在请求中省略了该参数。授权服务器必须忽略无法识别的请求参数。请求和响应参数不得被包含多次。

授权服务器必须支持对授权端点使用HTTP“GET”方法[RFC2616]，并且也可以支持使用“POST”方法。

1.3.1.1 3.1.1. Response Type

授权终端由授权代码模式和简化授权模式的工作流中使用。客户端使用以下参数通知授权服务器所需的授权类型：

response_type: REQUIRED. 值必须是用于请求授权代码的“code”之一，如第4.1.1节所述，“token”用于请求访问令牌（简化授权），如4.2.1节所述，或者注册的扩展值，如第8.4节。

扩展响应类型可以包含空格（%x20）分隔的值列表，其中值的顺序无关紧要（例如，响应类型“a b”与“b a”相同）。这种复合响应类型的含义由它们各自的规范定义。

如果授权请求缺少“response_type”参数，或者不理解响应类型，授权服务器必须返回如第4.1.2.1节所述的错误响应。

1.3.1.2 3.1.2. Redirection Endpoint

完成与资源所有者的交互后，授权服务器将资源所有者的用户代理指向客户端。在用户注册过程中或在发出授权请求时，授权服务器将user-agent重定向到先前与授权服务器建立的客户端重定向端点。
重定向端点URI必须是[RFC3986]第4.3节定义的绝对URI。端点URI可以包括`application/x-www-form-urlencoded`格式的（附录B）查询组件（[RFC3986]第3.4节），并且在添加其他查询参数时必须保留

该组件。端点URI绝不能包含片段组件。

1.3.1.2.1 3.1.2.1. Endpoint Request Confidentiality

当请求的响应类型是“code”或“token”时，或者当重定向请求将导致在开放网络上传输敏感凭证时，重定向端点应该使用如第1.6节所述的TLS。此规范并未强制要求使用TLS，因为在撰写本文时，要求客户端部署TLS对许多客户端开发人员来说是一个重大障碍。如果TLS不可用，授权服务器应该在重定向之前警告资源所有者关于不安全端点（例如，在授权请求期间显示消息）。

缺乏TLS会严重影响客户端及其授权访问的受保护资源的安全性。当授权过程以委托的终端用户授权的形式被用作客户端（例如，第三方登录服务）使用时，TLS的使用尤其重要。

1.3.1.2.2 3.1.2.2. Registration Requirements

授权服务器必须要求以下客户端注册其重定向端点：

- 公共客户。
- 使用简化授权类型的机密客户端。

授权服务器应该在使用授权端点之前要求所有客户端注册其重定向端点。

授权服务器应该要求客户端提供完整的重定向URI（客户端可以使用“state”请求参数来实现按请求定制）如果要求注册完整的重定向URI是不可能的，授权服务器应该要求注册URI方案，权限和路径（允许客户端在请求授权时仅动态改变重定向URI的查询组件）。

授权服务器可以允许客户端注册多个重定向端

若无重定向注册，攻击者可能使用授权端点用作开放重定向器（如第10.15节中所述）。

1.3.1.2.3 3.1.2.3. Dynamic Configuration

如果已注册了多个重定向URI，或者只注册了部分重定向URI，或者没有注册重定向URI，则客户端必须使用“redirect_uri”请求参数包含带有授权请求的重定向URI。

当授权请求中包含重定向URI时，如果注册了任一重定向URI，授权服务器必须将接收到的值与按照[RFC3986]第6节中定义的重定向URI列表中至少一个已注册重定向URI（或URI组件）进行匹配。如果客户端注册包含完整重定向URI，则授权服务器必须使用[RFC3986]第6.2.1节中定义的简单字符串比较法来比较两个URI。

1.3.1.2.4 3.1.2.4. Invalid Endpoint

如果授权请求由于重定向URI的缺少，无效或不匹配的而未通过验证，则授权服务器应当通知资源所有者该错误，并且不得自动将用户代理重定向到无效的重定向URI。

1.3.1.2.5 3.1.2.5. Endpoint Content

对客户端点的重定向请求通常会导致由user-agent处理的HTML文档响应（就是通常会返回一个网页）。如果HTML响应直接作为重定向请求的结果提供，则HTML文档中包含的任何脚本都将以完全访问重定向URI及其包含的凭据的方式执行。

客户端不应在重定向端点响应中包含任何第三方脚本（例如，第三方分析，社交插件，广告网络）。相反，它应该从URI中提取凭据并将用户代理再次重定向到另一个端点，而不暴露凭证（在URI或其他地方）。如果包含第三方脚本，客户端必须确保首先执行自己的脚本（用于从URI中提取和删除凭据）

1.3.2 3.2. Token Endpoint

客户端使用令牌端点通过呈现其权限授予或刷新令牌来获取访问令牌。除了简化授权类型之外，令牌端点与每个权限授予一起使用（因为访问令牌被直接发出）。

客户端获取令牌端点位置的方法超出了本规范的范围，因为这通常由服务文档提供。

端点URI可以包括`application/x-www-form-urlencoded`格式的（附录B）查询组件（[RFC3986]第3.4节），并且在添加其他查询参数时必须保留该组件。端点URI绝不能包含片段组件。

由于对token端点的请求导致用户身份验证和凭据的明文传输（在HTTP响应中），在向授权端点发送请求时，授权服务器必须使用第1.6节中所述的TLS。对于没有值的参数，必须当作在请求中省略了该参数。授权服务器必须忽略无法识别的请求参数。请求和响应参数不得被包含多次。

1.3.2.1 3.2.1. Client Authentication

在向令牌端点发出请求时，Confidential clients或其他clients发出的客户端凭证必须使用授权服务器进行身份验证，如第2.3节所述。客户端身份验证用于：

- 强制将刷新令牌和授权码绑定到发给它们的客户端。当授权代码通过不安全的通道传输到重定向端点或者重定向URI尚未完整注册时，客户端身份验证至关重要。
- 通过禁用客户端或更改其凭据从受感染的客户端恢复，从而防止攻击者滥用被盗的刷新令牌。更改单组客户端凭据比撤消整组刷新令牌要快得多。
- 实施身份验证管理最佳实践，这需要定期进行凭据轮换。旋转整组刷新令牌可能具有挑战性，而单组客户端凭证的轮换则更加容易。

客户端可以在向令牌端点发送请求时使用“client_id”请求参数来对自身进行标识。在对token令牌端口的“authorization_code”“grant_type”请求中，未经身份验证的客户端必须发送其“client_id”以防止自己无意中接受用于具有不同“client_id”的客户端的代码。这可以保护客户端不会替换身份验证代码。 （它不为受保护资源提供额外的安全性。）

1.3.3 3.3. Access Token Scope

授权和令牌端点允许客户端使用“范围”请求参数指定访问请求的范围。反过来，授权服务器使用“范围”响应参数来通知客户端发出的访问令牌的范围。

scope参数的值表示为以空格分隔的区分大小写的字符串列表。字符串由授权服务器定义。如果该值包含多个以空格分隔的字符串，则它们的顺序无关紧要，并且每个字符串都会为请求的范围添加其他访问范围。

```
scope       = scope-token *( SP scope-token )
scope-token = 1*( %x21 / %x23-5B / %x5D-7E )
```

根据授权服务器策略或资源所有者的指示，授权服务器可以完全或部分忽略客户端请求的范围。如果发布的访问令牌范围与客户端请求的范围不同，则授权服务器必须包含“scope”响应参数，以通知客户端授予的实际范围。

如果客户端在请求授权时省略了scope参数，则授权服务器必须使用预定义的默认值处理请求，或者使请求失败，指示范围无效。授权服务器应该记录其范围要求和默认值（如果已定义）。

1.4 4. Obtaining Authorization

要请求访问令牌，客户端需从资源所有者处获取授权。授权以`authorization grant`的形式表示，客户端使用该`authorization grant`来请求访问令牌。OAuth定义了四种授权类型`authorization code`，`implicit`，`resource owner password credentials`，和`client credentials`。它还提供了一种用于定义其他授权类型的扩展机制。

1.4.1 4.1. 授权码模式（Authorization Code Grant）

`authorization code grant type`通常用于获取`access tokens`和`refresh tokens`，并且针对`confidential clients`进行了优化。由于这是一个基于重定向的流程，客户端必须有能力和资源拥有者的user-agent（通常为WEB浏览器）进行交互，并且有能力接收来自授权服务器的重定向请求。

```
     +----------+
     | Resource |
     |   Owner  |
     |          |
     +----------+
          ^
          |
         (B)
     +----|-----+          Client Identifier      +---------------+
     |         -+----(A)-- & Redirection URI ---->|               |
     |  User-   |                                 | Authorization |
     |  Agent  -+----(B)-- User authenticates --->|     Server    |
     |          |                                 |               |
     |         -+----(C)-- Authorization Code ---<|               |
     +-|----|---+                                 +---------------+
       |    |                                         ^      v
      (A)  (C)                                        |      |
       |    |                                         |      |
       ^    v                                         |      |
     +---------+                                      |      |
     |         |>---(D)-- Authorization Code ---------'      |
     |  Client |          & Redirection URI                  |
     |         |                                             |
     |         |<---(E)----- Access Token -------------------'
     +---------+       (w/ Optional Refresh Token)

   Note: The lines illustrating steps (A), (B), and (C) are broken into
   two parts as they pass through the user-agent.

                     Figure 3: Authorization Code Flow
```

Figure 3中的图表包含如下步骤：

（A）client通过将资源拥有者重定向到授权服务器来初始化这个流程。client需要在请求中包含client identifier, requested scope, local state, and a redirection URI，这些内容在client被赋予（或者被拒绝）权限后也会被发送回来。

（B）授权服务器通过user-agent来对资源拥有者进行身份验证，并确定资源所有者是否授予或拒绝客户端的访问请求。

（C）假设资源所有者授予访问权限，授权服务器使用先前提供的重定向URI（在请求中或在客户端注册期间）将用户代理重定向回客户端。重定向URI包含授权代码和客户端先前提供的任一本地状态。

（D）客户端通过包含在上一步骤中接收的授权代码来请求来自授权服务器的令牌端点的访问令牌。发出请求时，客户端使用授权服务器进行身份验证。客户端通过包含用于获取验证授权码的重定向URI进行 验证。

（E）授权服务器对客户端进行身份验证，验证授权代码，并确保收到的重定向URI与步骤（C）中用于重定向客户端的URI相匹配。

1.4.1.1 4.1.1. 授权请求（Authorization Request）

客户端通过使用`application/x-www-form-urlencoded`格式将以下参数添加到授权端点URI的查询组件来构造请求URI（附录B）：

- **response_type**： REQUIRED. Value MUST be set to “code”.
- **client_id**： REQUIRED. The client identifier as described in Section 2.2.
- **redirect_uri**： OPTIONAL. As described in Section 3.1.2.

- **scope**： OPTIONAL. The scope of the access request as described by Section 3.3.
- **state**： RECOMMENDED. 客户端用于维护请求和回调之间状态的不透明值。当授权服务器在将用户代理重定向回客户端时包含此值。这个参数应当被用来方式CSRF攻击 （参见10.12节）。

客户端使用HTTP重定向响应或通过用户代理可用的其他方式将资源所有者定向到构造的URI。

例如，client使用TLS将user-agent定向到如下HTTP请求：

```
1    GET /authorize?
response_type=code&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A%2F%2F
client%2Eexample%2Ecom%2Fcb HTTP/1.1
2    Host: server.example.com
```

为确保所有必须的参数都被呈现并且合法，授权服务器需要验证请求。如果请求合法，授权服务器对资源所有者进行身份验证并获取授权决策（通过询问资源所有者或通过其他方式建立批准）。

建立决策时，授权服务器使用HTTP重定向响应或通过用户代理可用的其他方式将用户代理指向提供的客户端重定向URI。

1.4.1.2 4.1.2. 授权响应（Authorization Response）

如果资源所有者授予访问请求，则授权服务器通过使用`application/x-www-form-urlencoded`格式将以下参数添加到重定向URI的查询组件来发布授权代码并将其传递给客户端（参见附录B）：

- **code**： REQUIRED. 授权服务器生成的授权码。授权代码必须在发布后尽快过期，以减少泄漏风险。建议最长授权代码生存期为10分钟。客户端不得多次使用授权码。如果授权代码被多次使用，授权服务器必须拒绝该请求，并且应该（如果可能）撤销先前基于该授权代码发出的所有令牌。授权代码与客户端标识符和重定向URI绑定。
- **state**： REQUIRED if the “state” parameter was present in the client authorization request. The exact value received from the client.

例如，授权服务器通过发送如下HTTP响应来重定向user-agent：

```
1     HTTP/1.1 302 Found
2     Location: https://client.example.com/cb?
code=SplxlOBeZQQYbYS6WxSbIA&state=xyz
```

客户端必须忽略无法识别的响应参数。本规范未定义授权代码字符串大小。客户端应避免对代码值大小进行假设。授权服务器应该记录由它发出的任何码值的大小。

1.4.1.2.1 4.1.2.1. 异常响应（Error Response）

如果因为缺失、无效或不匹配的URI，或者客户端的identifier 缺失或无效而导致请求失败 authorization server应当将这些错误通知给资源拥有者而不能自动将user-agent重定向到无效的URI。

如果资源拥有者拒绝了访问请求，或因为其他原因失败，authorization server 应当以附录B所示的格式，以`application/x-www-form-urlencoded`的编码添加如下参数：

- **error**： REQUIRED. 一个单个ASCII错误码。值域如下：
  - **invalid_request**： The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.
  - **unauthorized_client**： The client is not authorized to request an authorization code using this method.
  - **access_denied**： The resource owner or authorization server denied the request.
  - **unsupported_response_type**： The authorization server does not support obtaining an authorization code using this method.

    - **invalid_scope**： The requested scope is invalid, unknown, or malformed.
    - **server_error**： The authorization server encountered an un condition that prevented it from fulfilling the request. (This error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect.)
    - **temporarily_unavailable**： The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server. (This error code is needed because a 503 Service Unavailable HTTP status code cannot be returned to the client via an HTTP redirect.)

Values for the “error” parameter MUST NOT include characters outside the set `%x20-21 / %x23-5B / %x5D-7E`.

- **error_description**： OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the “error_description” parameter MUST NOT include characters outside the set `%x20-21 / %x23-5B / %x5D-7E`.
- **error_uri**： OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the “error_uri” parameter MUST conform to the URI-reference syntax and thus MUST NOT include characters outside the set `%x21 / %x23-5B / %x5D-7E`.
- **state**： REQUIRED if a “state” parameter was present in the client authorization request. The exact value received from the client.

  For example, the authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=access_denied&state=xyz
```

1.4.1.3 4.1.3. 授权令牌请求（Access Token Request）

client通过向token终端发送如下HTTP请求体（`application/x-www-form-urlencoded`格式、UTF-8编码的参数）

- **grant_type**： REQUIRED. Value MUST be set to “authorization_code”.
- **code**： REQUIRED. The authorization code received from the authorization server.
- **redirect_uri**： REQUIRED, if the “redirect_uri” parameter was included in the authorization request as described in Section 4.1.1, and their values MUST be identical.
- **client_id**： REQUIRED, if the client is not authenticating with the authorization server as described in Section 3.2.1.

If the client type is confidential or the client was issued client credentials (or assigned other authentication requirements), the client MUST authenticate with the authorization server as described in Section 3.2.1.

For example, the client makes the following HTTP request using TLS (with extra line breaks for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

The authorization server MUST:

o require client authentication for confidential clients or for any client that was issued client credentials (or with other authentication requirements),

o authenticate the client if client authentication is included,

o ensure that the authorization code was issued to the authenticated confidential client, or if the client is public, ensure that the code was issued to “client_id” in the request,

o verify that the authorization code is valid, and

o ensure that the “redirect_uri” parameter is present if the “redirect_uri” parameter was included in the initial authorization request as described in Section 4.1.1, and if included ensure that their values are identical.

1.4.1.4 4.1.4. 访问令牌响应（Access Token Response）

If the access token request is valid and authorized, the authorization server issues an access token and optional refresh token as described in Section 5.1. If the request client authentication failed or is invalid, the authorization server returns an error response as described in Section 5.2.

An example successful response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

1.4.2 4.2. 简化模式（Implicit Grant）

The implicit grant type is used to obtain access tokens (it does not support the issuance of refresh tokens) and is optimized for public clients known to operate a particular redirection URI. These clients are typically implemented in a browser using a scripting language such as JavaScript.

Since this is a redirection-based flow, the client must be capable of interacting with the resource owner’s user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

Unlike the authorization code grant type, in which the client makes separate requests for authorization and for an access token, the client receives the access token as the result of the authorization request.

The implicit grant type does not include client authentication, and relies on the presence of the resource owner and the registration of the redirection URI. Because the access token is encoded into the redirection URI, it may be exposed to the resource owner and other applications residing on the same device.

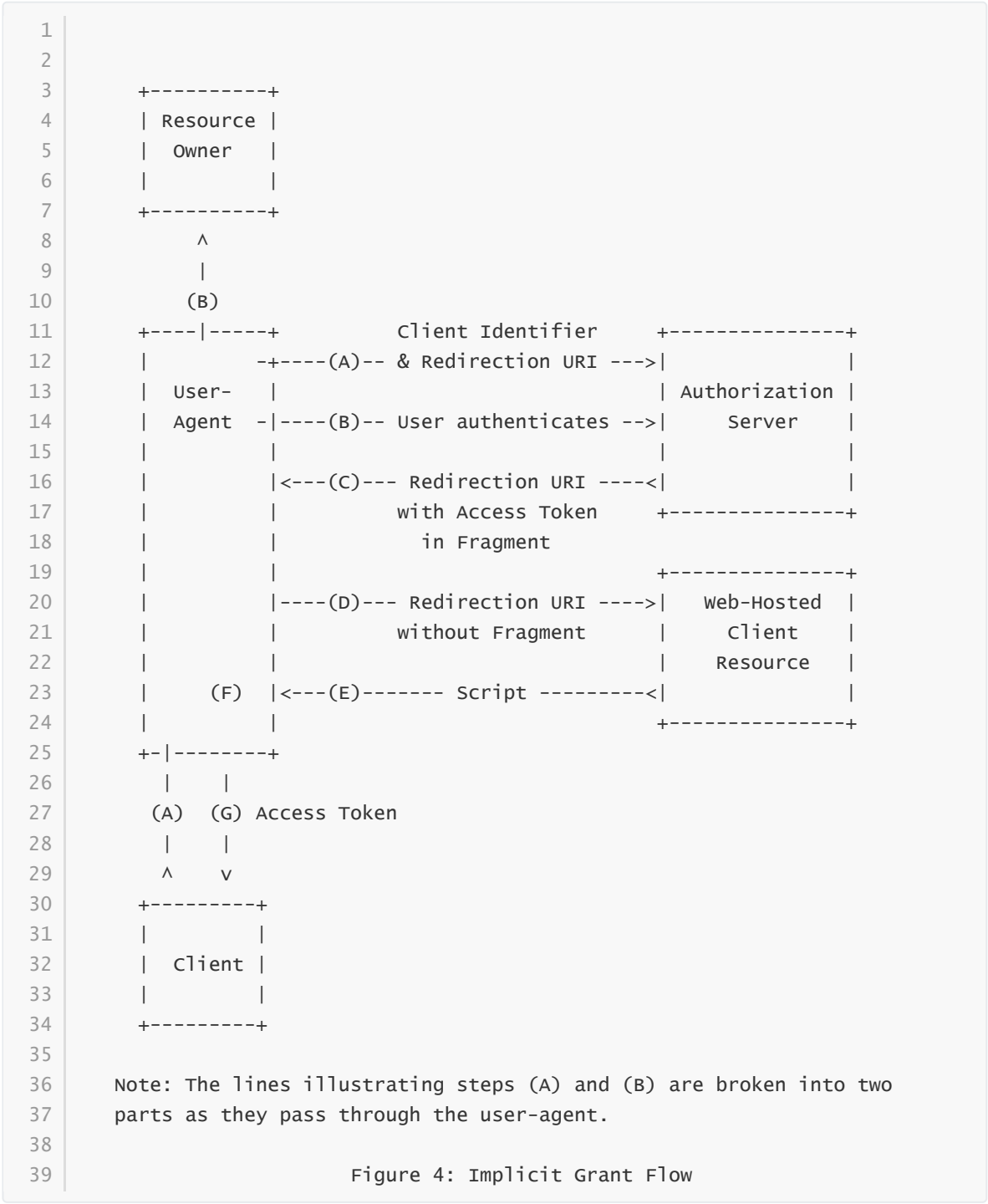

Figure 4: Implicit Grant Flow

The flow illustrated in Figure 4 includes the following steps:

(A) The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

(B) The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

© Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment.

(D) The user-agent follows the redirection instructions by making a request to the web-hosted client resource (which does not include the fragment per [RFC2616]). The user-agent retains the fragment information locally.

(E) The web-hosted client resource 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 user-agent, and extracting the access token (and other parameters) contained in the fragment.

(F) The user-agent executes the script provided by the web-hosted client resource locally, which extracts the access token.

(G) The user-agent passes the access token to the client.

See Sections 1.3.2 and 9 for background on using the implicit grant.
See Sections 10.3 and 10.16 for important security considerations when using the implicit grant.

1.4.2.1 4.2.1. Authorization Request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format, per Appendix B:

- **response_type**： REQUIRED. Value MUST be set to "token".
- **client_id**： REQUIRED. The client identifier as described in Section 2.2.
- **redirect_uri**： OPTIONAL. As described in Section 3.1.2.
- **scope**： OPTIONAL. The scope of the access request as described by Section 3.3.
- **state**： RECOMMENDED. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter SHOULD be used for preventing cross-site request forgery as described in Section 10.12.

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the user-agent to make the following HTTP request using TLS (with extra line breaks for display purposes only):

```
1    GET /authorize?response_type=token&client_id=s6BhdRkqt3&state=xyz
2        &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
3    Host: server.example.com
```

The authorization server validates the request to ensure that all required parameters are present and valid. The authorization server MUST verify that the redirection URI to which it will redirect the access token matches a redirection URI registered by the client as described in Section 3.1.2.

If the request is valid, the authorization server authenticates the resource owner and obtains an authorization decision (by asking the resource owner or by establishing approval via other means).

When a decision is established, the authorization server directs the user-agent to the provided client redirection URI using an HTTP redirection response, or by other means available to it via the user-agent.

1.4.2.2 4.2.2. Access Token Response

If the resource owner grants the access request, the authorization server issues an access token and delivers it to the client by adding the following parameters to the fragment component of the redirection URI using the “application/x-www-form-urlencoded” format, per Appendix B:

- **access_token**： REQUIRED. The access token issued by the authorization server.
- **token_type**： REQUIRED. The type of the token issued as described in Section 7.1. Value is case insensitive.
- **expires_in**： RECOMMENDED. The lifetime in seconds of the access token. For example, the value “3600” denotes that the access token will expire in one hour from the time the response was generated. If omitted, the authorization server SHOULD provide the expiration time via other means or document the default value.
- **scope**： OPTIONAL, if identical to the scope requested by the client; otherwise, REQUIRED. The scope of the access token as described by Section 3.3.
- **state**： REQUIRED if the “state” parameter was present in the client authorization request. The exact value received from the client.

The authorization server MUST NOT issue a refresh token.

For example, the authorization server redirects the user-agent by sending the following HTTP response (with extra line breaks for display purposes only):

```
    HTTP/1.1 302 Found
    Location: http://example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA
              &state=xyz&token_type=example&expires_in=3600
```

Developers should note that some user-agents do not support the inclusion of a fragment component in the HTTP “Location” response header field. Such clients will require using other methods for redirecting the client than a 3xx redirection response – for example, returning an HTML page that includes a ‘continue’ button with an action linked to the redirection URI.

The client MUST ignore unrecognized response parameters. The access token string size is left undefined by this specification. The client should avoid making assumptions about value sizes. The authorization server SHOULD document the size of any value it issues.

1.4.2.2.1 4.2.2.1. Error Response

If the request fails due to a missing, invalid, or mismatching redirection URI, or if the client identifier is missing or invalid, the authorization server SHOULD inform the resource owner of the error and MUST NOT automatically redirect the user-agent to the invalid redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the fragment component of the redirection URI using the “application/x-www-form-urlencoded” format, per Appendix B:

- **error**： REQUIRED. A single ASCII [USASCII] error code from the following:
    - **invalid_request**： The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.
    - **unauthorized_client**： The client is not authorized to request an access token using this method.

- **access_denied**： The resource owner or authorization server denied the request.
- **unsupported_response_type**： The authorization server does not support obtaining an access token using this method.
- **invalid_scope**： The requested scope is invalid, unknown, or malformed.
- **server_error**： The authorization server encountered an unexpected condition that prevented it from fulfilling the request. (This error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect.)
- **temporarily_unavailable**： The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server. (This error code is needed because a 503 Service Unavailable HTTP status code cannot be returned to the client via an HTTP redirect.)

Values for the “error” parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

- **error_description**： OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the “error_description” parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.
- **error_uri**： OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the “error_uri” parameter MUST conform to the URI-reference syntax and thus MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E.
- **state**： REQUIRED if a “state” parameter was present in the client authorization request. The exact value received from the client.

  For example, the authorization server redirects the user-agent by sending the following HTTP response:

```
   HTTP/1.1 302 Found
   Location: https://client.example.com/cb#error=access_denied&state=xyz
```

1.4.3 4.3. Resource Owner Password Credentials Grant

The resource owner password credentials grant type is suitable in cases where the resource owner has a trust relationship with the client, such as the device operating system or a highly privileged

application. The authorization server should take special care when enabling this grant type and only allow it when other flows are not viable.

This grant type is suitable for clients capable of obtaining the resource owner’s credentials (username and password, typically using an interactive form). It is also used to migrate existing clients using direct authentication schemes such as HTTP Basic or Digest authentication to OAuth by converting the stored credentials to an access token.

```
     +----------+
     | Resource |
     |  Owner   |
     |          |
     +----------+
          v
          |    Resource Owner
```

```
          (A) Password Credentials
           |
           v
      +---------+                                  +---------------+
      |         |>--(B)---- Resource Owner ------->|               |
      |         |         Password Credentials     | Authorization |
      | Client  |                                  |     Server    |
      |         |<--(C)---- Access Token ---------<|               |
      |         |    (w/ Optional Refresh Token)   |               |
      +---------+                                  +---------------+

             Figure 5: Resource Owner Password Credentials Flow
```

The flow illustrated in Figure 5 includes the following steps:

(A) The resource owner provides the client with its username and password.

(B) The client requests an access token from the authorization server's token endpoint by including the credentials received from the resource owner. When making the request, the client authenticates with the authorization server.

© The authorization server authenticates the client and validates the resource owner credentials, and if valid, issues an access token.

1.4.3.1 4.3.1. Authorization Request and Response

The method through which the client obtains the resource owner credentials is beyond the scope of this specification. The client MUST discard the credentials once an access token has been obtained.

1.4.3.2 4.3.2. Access Token Request

The client makes a request to the token endpoint by adding the following parameters using the "application/x-www-form-urlencoded" format per Appendix B with a character encoding of UTF-8 in the HTTP request entity-body:

- `grant_type` : REQUIRED. Value MUST be set to "password".
- `username` : REQUIRED. The resource owner username.
- `password` : REQUIRED. The resource owner password.
- `scope` : OPTIONAL. The scope of the access request as described by Section 3.3.

If the client type is confidential or the client was issued client credentials (or assigned other authentication requirements), the client MUST authenticate with the authorization server as described in Section 3.2.1.

For example, the client makes the following HTTP request using transport-layer security (with extra line breaks for display purposes only):

```
    POST /token HTTP/1.1
    Host: server.example.com
    Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
    Content-Type: application/x-www-form-urlencoded

    grant_type=password&username=johndoe&password=A3ddj3w
```

The authorization server MUST:

o require client authentication for confidential clients or for any client that was issued client credentials (or with other authentication requirements),

o authenticate the client if client authentication is included, and

o validate the resource owner password credentials using its existing password validation algorithm.

Since this access token request utilizes the resource owner's password, the authorization server MUST protect the endpoint against brute force attacks (e.g., using rate-limitation or generating alerts).

1.4.3.3 4.3.3. Access Token Response

If the access token request is valid and authorized, the authorization server issues an access token and optional refresh token as described in Section 5.1. If the request failed client authentication or is invalid, the authorization server returns an error response as described in Section 5.2.

An example successful response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

1.4.4 4.4. Client Credentials Grant

The client can request an access token using only its client credentials (or other supported means of authentication) when the client is requesting access to the protected resources under its control, or those of another resource owner that have been previously arranged with the authorization server (the method of which is beyond the scope of this specification).

The client credentials grant type MUST only be used by confidential clients.

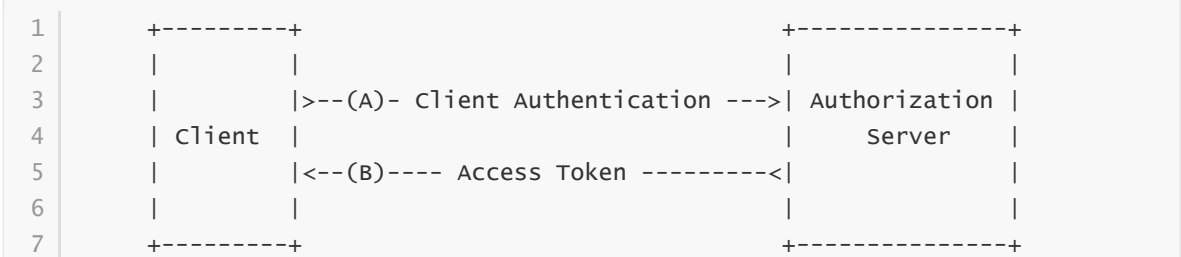

Figure 6: Client Credentials Flow

The flow illustrated in Figure 6 includes the following steps:

(A) The client authenticates with the authorization server and requests an access token from the token endpoint.

(B) The authorization server authenticates the client, and if valid, issues an access token.

1.4.4.1 4.4.1. Authorization Request and Response

Since the client authentication is used as the authorization grant, no additional authorization request is needed.

1.4.4.2 4.4.2. Access Token Request

The client makes a request to the token endpoint by adding the following parameters using the “application/x-www-form-urlencoded” format per Appendix B with a character encoding of UTF-8 in the HTTP request entity-body:

- `grant_type` ： REQUIRED. Value MUST be set to “client_credentials”.
- `scope` ： OPTIONAL. The scope of the access request as described by Section 3.3.

  The client MUST authenticate with the authorization server as described in Section 3.2.1.

  For example, the client makes the following HTTP request using transport-layer security (with extra line breaks for display purposes only):

```
    POST /token HTTP/1.1
    Host: server.example.com
    Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
    Content-Type: application/x-www-form-urlencoded

    grant_type=client_credentials

  The authorization server MUST authenticate the client.
```

1.4.4.3 4.4.3. Access Token Response

If the access token request is valid and authorized, the authorization server issues an access token as described in Section 5.1. A refresh token SHOULD NOT be included. If the request failed client authentication or is invalid, the authorization server returns an error response as described in Section 5.2.

An example successful response:

```
    HTTP/1.1 200 OK
    Content-Type: application/json;charset=UTF-8
    Cache-Control: no-store
    Pragma: no-cache

    {
      "access_token":"2YotnFZFEjr1zCsicMWpAA",
      "token_type":"example",
      "expires_in":3600,
      "example_parameter":"example_value"
    }
```

1.4.5 4.5. Extension Grants

The client uses an extension grant type by specifying the grant type using an absolute URI (defined by the authorization server) as the value of the “grant_type” parameter of the token endpoint, and by adding any additional parameters necessary.

For example, to request an access token using a Security Assertion Markup Language (SAML) 2.0 assertion grant type as defined by [OAuth-SAML2], the client could make the following HTTP request using TLS (with extra line breaks for display purposes only):

```
    POST /token HTTP/1.1
    Host: server.example.com
    Content-Type: application/x-www-form-urlencoded

    grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-
    bearer&assertion=PEFzc2Vyd[...omitted for brevity...]Q-PC9Bc3NlcnRpb24-
```

If the access token request is valid and authorized, the authorization server issues an access token and optional refresh token as described in Section 5.1. If the request failed client authentication or is invalid, the authorization server returns an error response as described in Section 5.2.

1.5 5. Issuing an Access Token

If the access token request is valid and authorized, the authorization server issues an access token and optional refresh token as described in Section 5.1. If the request failed client authentication or is invalid, the authorization server returns an error response as described in Section 5.2.

1.5.1 5.1. Successful Response

The authorization server issues an access token and optional refresh token, and constructs the response by adding the following parameters to the entity-body of the HTTP response with a 200 (OK) status code:

- `access_token` : REQUIRED. The access token issued by the authorization server.
- `token_type` : REQUIRED. The type of the token issued as described in Section 7.1. Value is case insensitive.
- `expires_in` : RECOMMENDED. The lifetime in seconds of the access token. For example, the value “3600” denotes that the access token will expire in one hour from the time the response was generated. If omitted, the authorization server SHOULD provide the expiration time via other means or document the default value.
- `refresh_token` : OPTIONAL. The refresh token, which can be used to obtain new access tokens using the same authorization grant as described in Section 6.
- `scope` : OPTIONAL, if identical to the scope requested by the client; otherwise, REQUIRED. The scope of the access token as described by Section 3.3.

  The parameters are included in the entity-body of the HTTP response using the “application/json” media type as defined by [RFC4627]. The parameters are serialized into a JavaScript Object Notation (JSON) structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. The order of parameters does not matter and can vary.

  The authorization server MUST include the HTTP “Cache-Control” response header field [RFC2616] with a value of “no-store” in any response containing tokens, credentials, or other sensitive information, as well as the “Pragma” response header field [RFC2616] with a value of “no-cache”.

For example:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

The client MUST ignore unrecognized value names in the response. The sizes of tokens and other values received from the authorization server are left undefined. The client should avoid making assumptions about value sizes. The authorization server SHOULD document the size of any value it issues.

1.5.2 5.2. Error Response

The authorization server responds with an HTTP 400 (Bad Request) status code (unless specified otherwise) and includes the following parameters with the response:

- `error` : REQUIRED. A single ASCII [USASCII] error code from the following:
  - `invalid_request` : The request is missing a required parameter, includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.
  - `invalid_client` : Client authentication failed (e.g., unknown client, no client authentication included, or unsupported authentication method). The authorization server MAY return an HTTP 401 (Unauthorized) status code to indicate which HTTP authentication schemes are supported. If the client attempted to authenticate via the "Authorization" request header field, the authorization server MUST respond with an HTTP 401 (Unauthorized) status code and include the "WWW-Authenticate" response header field matching the authentication scheme used by the client.
  - `invalid_grant` : The provided authorization grant (e.g., authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, does not match the redirection URI used in the authorization request, or was issued to another client.
  - `unauthorized_client` : The authenticated client is not authorized to use this authorization grant type.
  - `unsupported_grant_type` : The authorization grant type is not supported by the authorization server.
  - `invalid_scope` : The requested scope is invalid, unknown, malformed, or exceeds the scope granted by the resource owner.

Values for the "error" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

- `error_description` : OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the "error_description" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

- `error_uri` ： OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the “error_uri” parameter MUST conform to the URI-reference syntax and thus MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E.
  The parameters are included in the entity-body of the HTTP response using the “application/json” media type as defined by [RFC4627]. The parameters are serialized into a JSON structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. The order of parameters does not matter and can vary.

  For example:

```
    HTTP/1.1 400 Bad Request
    Content-Type: application/json;charset=UTF-8
    Cache-Control: no-store
    Pragma: no-cache

    {
      "error":"invalid_request"
    }
```

1.6 6. Refreshing an Access Token

If the authorization server issued a refresh token to the client, the client makes a refresh request to the token endpoint by adding the following parameters using the “application/x-www-form-urlencoded” format per Appendix B with a character encoding of UTF-8 in the HTTP request entity-body:

- `grant_type` ： REQUIRED. Value MUST be set to “refresh_token”.
- `refresh_token` ： REQUIRED. The refresh token issued to the client.
- `scope` ： OPTIONAL. The scope of the access request as described by Section 3.3. The requested scope MUST NOT include any scope not originally granted by the resource owner, and if omitted is treated as equal to the scope originally granted by the resource owner.

  Because refresh tokens are typically long-lasting credentials used to request additional access tokens, the refresh token is bound to the client to which it was issued. If the client type is confidential or the client was issued client credentials (or assigned other authentication requirements), the client MUST authenticate with the authorization server as described in Section 3.2.1.

  For example, the client makes the following HTTP request using transport-layer security (with extra line breaks for display purposes only):

```
     POST /token HTTP/1.1
     Host: server.example.com
     Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
     Content-Type: application/x-www-form-urlencoded

     grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
```

The authorization server MUST:

o require client authentication for confidential clients or for any client that was issued client credentials (or with other authentication requirements),

o authenticate the client if client authentication is included and ensure that the refresh token was issued to the authenticated client, and

o validate the refresh token.

If valid and authorized, the authorization server issues an access token as described in Section 5.1. If the request failed verification or is invalid, the authorization server returns an error response as described in Section 5.2.

The authorization server MAY issue a new refresh token, in which case the client MUST discard the old refresh token and replace it with the new refresh token. The authorization server MAY revoke the old refresh token after issuing a new refresh token to the client. If a new refresh token is issued, the refresh token scope MUST be identical to that of the refresh token included by the client in the request.

1.7 7. Accessing Protected Resources

The client accesses protected resources by presenting the access token to the resource server. The resource server MUST validate the access token and ensure that it has not expired and that its scope covers the requested resource. The methods used by the resource server to validate the access token (as well as any error responses) are beyond the scope of this specification but generally involve an interaction or coordination between the resource server and the authorization server.

The method in which the client utilizes the access token to authenticate with the resource server depends on the type of access token issued by the authorization server. Typically, it involves using the HTTP “Authorization” request header field [RFC2617] with an authentication scheme defined by the specification of the access token type used, such as [RFC6750].

1.7.1 7.1. Access Token Types

The access token type provides the client with the information required to successfully utilize the access token to make a protected resource request (along with type-specific attributes). The client MUST NOT use an access token if it does not understand the token type.

For example, the “bearer” token type defined in [RFC6750] is utilized by simply including the access token string in the request:

```
    GET /resource/1 HTTP/1.1
    Host: example.com
    Authorization: Bearer mF_9.B5f-4.1JqM
```

while the “mac” token type defined in [OAuth-HTTP-MAC] is utilized by issuing a Message Authentication Code (MAC) key together with the access token that is used to sign certain components of the HTTP requests:

```
    GET /resource/1 HTTP/1.1
    Host: example.com
    Authorization: MAC id="h480djs93hd8",
                       nonce="274312:dj83hs9s",
                       mac="kDZvddkndxvhGRXZhvuDjEWhGeE="
```

The above examples are provided for illustration purposes only. Developers are advised to consult the [RFC6750] and [OAuth-HTTP-MAC] specifications before use.

Each access token type definition specifies the additional attributes (if any) sent to the client together with the “access_token” response parameter. It also defines the HTTP authentication method used to include the access token when making a protected resource request.

1.7.2 7.2. Error Response

If a resource access request fails, the resource server SHOULD inform the client of the error. While the specifics of such error responses are beyond the scope of this specification, this document establishes a common registry in Section 11.4 for error values to be shared among OAuth token authentication schemes.

New authentication schemes designed primarily for OAuth token authentication SHOULD define a mechanism for providing an error status code to the client, in which the error values allowed are registered in the error registry established by this specification.

Such schemes MAY limit the set of valid error codes to a subset of the registered values. If the error code is returned using a named parameter, the parameter name SHOULD be “error”.

Other schemes capable of being used for OAuth token authentication, but not primarily designed for that purpose, MAY bind their error values to the registry in the same manner.

New authentication schemes MAY choose to also specify the use of the “error_description” and “error_uri” parameters to return error information in a manner parallel to their usage in this specification.

1.8 8. Extensibility

1.8.1 8.1. Defining Access Token Types

Access token types can be defined in one of two ways: registered in the Access Token Types registry (following the procedures in Section 11.1), or by using a unique absolute URI as its name.

Types utilizing a URI name SHOULD be limited to vendor-specific implementations that are not commonly applicable, and are specific to the implementation details of the resource server where they are used.

All other types MUST be registered. Type names MUST conform to the type-name ABNF. If the type definition includes a new HTTP authentication scheme, the type name SHOULD be identical to the HTTP authentication scheme name (as defined by [RFC2617]). The token type “example” is reserved for use in examples.

```
type-name  = 1*name-char
name-char  = "-" / "." / "_" / DIGIT / ALPHA
```

1.8.2 8.2. Defining New Endpoint Parameters

New request or response parameters for use with the authorization endpoint or the token endpoint are defined and registered in the OAuth Parameters registry following the procedure in Section 11.2.

Parameter names MUST conform to the param-name ABNF, and parameter values syntax MUST be well-defined (e.g., using ABNF, or a reference to the syntax of an existing parameter).

```
param-name  = 1*name-char
name-char   = "-" / "." / "_" / DIGIT / ALPHA
```

Unregistered vendor-specific parameter extensions that are not commonly applicable and that are specific to the implementation details of the authorization server where they are used SHOULD utilize a vendor-specific prefix that is not likely to conflict with other registered values (e.g., begin with 'companyname_').

1.8.3 8.3. Defining New Authorization Grant Types

New authorization grant types can be defined by assigning them a unique absolute URI for use with the "grant_type" parameter. If the extension grant type requires additional token endpoint parameters, they MUST be registered in the OAuth Parameters registry as described by Section 11.2.

1.8.4 8.4. Defining New Authorization Endpoint Response Types

New response types for use with the authorization endpoint are defined and registered in the Authorization Endpoint Response Types registry following the procedure in Section 11.3. Response type names MUST conform to the response-type ABNF.

```
response-type  = response-name *( SP response-name )
response-name  = 1*response-char
response-char  = "_" / DIGIT / ALPHA
```

If a response type contains one or more space characters (%x20), it is compared as a space-delimited list of values in which the order of values does not matter. Only one order of values can be registered, which covers all other arrangements of the same set of values.

For example, the response type "token code" is left undefined by this specification. However, an extension can define and register the "token code" response type. Once registered, the same combination cannot be registered as "code token", but both values can be used to denote the same response type.

1.8.5 8.5. Defining Additional Error Codes

In cases where protocol extensions (i.e., access token types, extension parameters, or extension grant types) require additional error codes to be used with the authorization code grant error response (Section 4.1.2.1), the implicit grant error response (Section 4.2.2.1), the token error response (Section 5.2), or the resource access error response (Section 7.2), such error codes MAY be defined.

Extension error codes MUST be registered (following the procedures in Section 11.4) if the extension they are used in conjunction with is a registered access token type, a registered endpoint parameter, or an extension grant type. Error codes used with unregistered extensions MAY be registered.

Error codes MUST conform to the error ABNF and SHOULD be prefixed by an identifying name when possible. For example, an error identifying an invalid value set to the extension parameter "example" SHOULD be named "example_invalid".

```
error      = 1*error-char
error-char = %x20-21 / %x23-5B / %x5D-7E
```

1.9 9. Native Applications

Native applications are clients installed and executed on the device used by the resource owner (i.e., desktop application, native mobile application). Native applications require special consideration related to security, platform capabilities, and overall end-user experience.

The authorization endpoint requires interaction between the client and the resource owner's user-agent. Native applications can invoke an external user-agent or embed a user-agent within the application. For example:

o External user-agent - the native application can capture the response from the authorization server using a redirection URI with a scheme registered with the operating system to invoke the client as the handler, manual copy-and-paste of the credentials, running a local web server, installing a user-agent extension, or by providing a redirection URI identifying a server-hosted resource under the client's control, which in turn makes the response available to the native application.

o Embedded user-agent - the native application obtains the response by directly communicating with the embedded user-agent by monitoring state changes emitted during the resource load, or accessing the user-agent's cookies storage.

When choosing between an external or embedded user-agent, developers
should consider the following:

o An external user-agent may improve completion rate, as the resource owner may already have an active session with the authorization server, removing the need to re-authenticate. It provides a familiar end-user experience and functionality. The resource owner may also rely on user-agent features or extensions to assist with authentication (e.g., password manager, 2-factor device reader).

o An embedded user-agent may offer improved usability, as it removes the need to switch context and open new windows.

o An embedded user-agent poses a security challenge because resource owners are authenticating in an unidentified window without access to the visual protections found in most external user-agents. An embedded user-agent educates end-users to trust unidentified requests for authentication (making phishing attacks easier to execute).

When choosing between the implicit grant type and the authorization
code grant type, the following should be considered:

o Native applications that use the authorization code grant type SHOULD do so without using client credentials, due to the native application's inability to keep client credentials confidential.

o When using the implicit grant type flow, a refresh token is not returned, which requires repeating the authorization process once the access token expires.

1.10 10. Security Considerations

As a flexible and extensible framework, OAuth's security considerations depend on many factors. The following sections provide implementers with security guidelines focused on the three client profiles described in Section 2.1: web application, user-agent-based application, and native application.

A comprehensive OAuth security model and analysis, as well as background for the protocol design, is provided by [OAuth-THREATMODEL].

1.10.1 10.1. Client Authentication

The authorization server establishes client credentials with web application clients for the purpose of client authentication. The authorization server is encouraged to consider stronger client authentication means than a client password. Web application clients MUST ensure confidentiality of client passwords and other client credentials.

The authorization server MUST NOT issue client passwords or other client credentials to native application or user-agent-based application clients for the purpose of client authentication. The authorization server MAY issue a client password or other credentials for a specific installation of a native application client on a specific device.

When client authentication is not possible, the authorization server SHOULD employ other means to validate the client's identity – for example, by requiring the registration of the client redirection URI or enlisting the resource owner to confirm identity. A valid redirection URI is not sufficient to verify the client's identity when asking for resource owner authorization but can be used to prevent delivering credentials to a counterfeit client after obtaining resource owner authorization.

The authorization server must consider the security implications of interacting with unauthenticated clients and take measures to limit the potential exposure of other credentials (e.g., refresh tokens) issued to such clients.

1.10.2 10.2. Client Impersonation

A malicious client can impersonate another client and obtain access to protected resources if the impersonated client fails to, or is unable to, keep its client credentials confidential.

The authorization server MUST authenticate the client whenever possible. If the authorization server cannot authenticate the client due to the client's nature, the authorization server MUST require the registration of any redirection URI used for receiving authorization responses and SHOULD utilize other means to protect resource owners from such potentially malicious clients. For example, the authorization server can engage the resource owner to assist in identifying the client and its origin.

The authorization server SHOULD enforce explicit resource owner authentication and provide the resource owner with information about the client and the requested authorization scope and lifetime. It is up to the resource owner to review the information in the context of the current client and to authorize or deny the request.

The authorization server SHOULD NOT process repeated authorization requests automatically (without active resource owner interaction) without authenticating the client or relying on other measures to ensure that the repeated request comes from the original client and not an impersonator.

1.10.3 10.3. Access Tokens

Access token credentials (as well as any confidential access token attributes) MUST be kept confidential in transit and storage, and only shared among the authorization server, the resource servers the access token is valid for, and the client to whom the access token is issued. Access token credentials MUST only be transmitted using TLS as described in Section 1.6 with server authentication as defined by [RFC2818].

When using the implicit grant type, the access token is transmitted in the URI fragment, which can expose it to unauthorized parties.

The authorization server MUST ensure that access tokens cannot be generated, modified, or guessed to produce valid access tokens by unauthorized parties.

The client SHOULD request access tokens with the minimal scope necessary. The authorization server SHOULD take the client identity into account when choosing how to honor the requested scope and MAY issue an access token with less rights than requested.

This specification does not provide any methods for the resource server to ensure that an access token presented to it by a given client was issued to that client by the authorization server.

1.10.4 10.4. Refresh Tokens

Authorization servers MAY issue refresh tokens to web application clients and native application clients.

Refresh tokens MUST be kept confidential in transit and storage, and shared only among the authorization server and the client to whom the refresh tokens were issued. The authorization server MUST maintain the binding between a refresh token and the client to whom it was issued. Refresh tokens MUST only be transmitted using TLS as described in Section 1.6 with server authentication as defined by [RFC2818].

The authorization server MUST verify the binding between the refresh token and client identity whenever the client identity can be authenticated. When client authentication is not possible, the authorization server SHOULD deploy other means to detect refresh token abuse.

For example, the authorization server could employ refresh token rotation in which a new refresh token is issued with every access token refresh response. The previous refresh token is invalidated

but retained by the authorization server. If a refresh token is compromised and subsequently used by both the attacker and the legitimate client, one of them will present an invalidated refresh token, which will inform the authorization server of the breach.

The authorization server MUST ensure that refresh tokens cannot be generated, modified, or guessed to produce valid refresh tokens by unauthorized parties.

1.10.5 10.5. Authorization Codes

The transmission of authorization codes SHOULD be made over a secure channel, and the client SHOULD require the use of TLS with its redirection URI if the URI identifies a network resource. Since authorization codes are transmitted via user-agent redirections, they could potentially be disclosed through user-agent history and HTTP referrer headers.

Authorization codes operate as plaintext bearer credentials, used to verify that the resource owner who granted authorization at the authorization server is the same resource owner returning to the client to complete the process. Therefore, if the client relies on the authorization code for its own resource owner authentication, the client redirection endpoint MUST require the use of TLS.

Authorization codes MUST be short lived and single-use. If the authorization server observes multiple attempts to exchange an authorization code for an access token, the authorization server SHOULD attempt to revoke all access tokens already granted based on the compromised authorization code.

If the client can be authenticated, the authorization servers MUST authenticate the client and ensure that the authorization code was issued to the same client.

1.10.6 10.6. Authorization Code Redirection URI Manipulation

When requesting authorization using the authorization code grant type, the client can specify a redirection URI via the “redirect_uri” parameter. If an attacker can manipulate the value of the redirection URI, it can cause the authorization server to redirect the resource owner user-agent to a URI under the control of the attacker with the authorization code.

An attacker can create an account at a legitimate client and initiate the authorization flow. When the attacker’s user-agent is sent to the authorization server to grant access, the attacker grabs the authorization URI provided by the legitimate client and replaces the

client’s redirection URI with a URI under the control of the attacker. The attacker then tricks the victim into following the manipulated link to authorize access to the legitimate client.

Once at the authorization server, the victim is prompted with a normal, valid request on behalf of a legitimate and trusted client, and authorizes the request. The victim is then redirected to an endpoint under the control of the attacker with the authorization code. The attacker completes the authorization flow by sending the authorization code to the client using the original redirection URI provided by the client. The client exchanges the authorization code with an access token and links it to the attacker’s client account, which can now gain access to the protected resources authorized by the victim (via the client).

In order to prevent such an attack, the authorization server MUST ensure that the redirection URI used to obtain the authorization code is identical to the redirection URI provided when exchanging the authorization code for an access token. The authorization server MUST require public clients and SHOULD require confidential clients to register their redirection URIs. If a redirection URI is provided in the request, the authorization server MUST validate it against the registered value.

1.10.7 10.7. Resource Owner Password Credentials

The resource owner password credentials grant type is often used for legacy or migration reasons. It reduces the overall risk of storing usernames and passwords by the client but does not eliminate the need to expose highly privileged credentials to the client.

This grant type carries a higher risk than other grant types because it maintains the password anti-pattern this protocol seeks to avoid. The client could abuse the password, or the password could unintentionally be disclosed to an attacker (e.g., via log files or other records kept by the client).

Additionally, because the resource owner does not have control over the authorization process (the resource owner’s involvement ends when it hands over its credentials to the client), the client can obtain access tokens with a broader scope than desired by the resource owner. The authorization server should consider the scope and lifetime of access tokens issued via this grant type.

The authorization server and client SHOULD minimize use of this grant type and utilize other grant types whenever possible.

1.10.8 10.8. Request Confidentiality

Access tokens, refresh tokens, resource owner passwords, and client credentials MUST NOT be transmitted in the clear. Authorization codes SHOULD NOT be transmitted in the clear.

The “state” and “scope” parameters SHOULD NOT include sensitive client or resource owner information in plain text, as they can be transmitted over insecure channels or stored insecurely.

1.10.9 10.9. Ensuring Endpoint Authenticity

In order to prevent man-in-the-middle attacks, the authorization server MUST require the use of TLS with server authentication as defined by [RFC2818] for any request sent to the authorization and token endpoints. The client MUST validate the authorization server's TLS certificate as defined by [RFC6125] and in accordance with its requirements for server identity authentication.

1.10.10 10.10. Credentials-Guessing Attacks

The authorization server MUST prevent attackers from guessing access tokens, authorization codes, refresh tokens, resource owner passwords, and client credentials.

The probability of an attacker guessing generated tokens (and other credentials not intended for handling by end-users) MUST be less than or equal to 2^(-128) and SHOULD be less than or equal to 2^(-160).

The authorization server MUST utilize other means to protect credentials intended for end-user usage.

1.10.11 10.11. Phishing Attacks

Wide deployment of this and similar protocols may cause end-users to become inured to the practice of being redirected to websites where they are asked to enter their passwords. If end-users are not careful to verify the authenticity of these websites before entering their credentials, it will be possible for attackers to exploit this practice to steal resource owners' passwords.

Service providers should attempt to educate end-users about the risks phishing attacks pose and should provide mechanisms that make it easy for end-users to confirm the authenticity of their sites. Client developers should consider the security implications of how they interact with the user-agent (e.g., external, embedded), and the ability of the end-user to verify the authenticity of the authorization server.

To reduce the risk of phishing attacks, the authorization servers MUST require the use of TLS on every endpoint used for end-user interaction.

1.10.12 10.12. Cross-Site Request Forgery

Cross-site request forgery (CSRF) is an exploit in which an attacker causes the user-agent of a victim end-user to follow a malicious URI (e.g., provided to the user-agent as a misleading link, image, or redirection) to a trusting server (usually established via the presence of a valid session cookie).

A CSRF attack against the client's redirection URI allows an attacker to inject its own authorization code or access token, which can result in the client using an access token associated with the attacker's protected resources rather than the victim's (e.g., save the victim's bank account information to a protected resource controlled by the attacker).

The client MUST implement CSRF protection for its redirection URI. This is typically accomplished by requiring any request sent to the redirection URI endpoint to include a value that binds the request to the user-agent's authenticated state (e.g., a hash of the session cookie used to authenticate the user-agent). The client SHOULD utilize the "state" request parameter to deliver this value to the authorization server when making an authorization request.

Once authorization has been obtained from the end-user, the authorization server redirects the end-user's user-agent back to the client with the required binding value contained in the "state" parameter. The binding value enables the client to verify the validity of the request by matching the binding value to the user-agent's authenticated state. The binding value used for CSRF protection MUST contain a non-guessable value (as described in Section 10.10), and the user-

agent's authenticated state (e.g., session cookie, HTML5 local storage) MUST be kept in a location accessible only to the client and the user-agent (i.e., protected by same-origin policy).

A CSRF attack against the authorization server's authorization endpoint can result in an attacker obtaining end-user authorization for a malicious client without involving or alerting the end-user.

The authorization server MUST implement CSRF protection for its authorization endpoint and ensure that a malicious client cannot obtain authorization without the awareness and explicit consent of the resource owner.

1.10.13 10.13. Clickjacking

In a clickjacking attack, an attacker registers a legitimate client and then constructs a malicious site in which it loads the authorization server's authorization endpoint web page in a transparent iframe overlaid on top of a set of dummy buttons, which are carefully constructed to be placed directly under important buttons on the authorization page. When an end-user clicks a misleading visible button, the end-user is actually clicking an invisible button on the authorization page (such as an "Authorize" button). This allows an attacker to trick a resource owner into granting its client access without the end-user's knowledge.

To prevent this form of attack, native applications SHOULD use external browsers instead of embedding browsers within the application when requesting end-user authorization. For most newer browsers, avoidance of iframes can be enforced by the authorization server using the (non-standard) "x-frame-options" header. This header can have two values, "deny" and "sameorigin", which will block any framing, or framing by sites with a different origin, respectively. For older browsers, JavaScript frame-busting techniques can be used but may not be effective in all browsers.

1.10.14 10.14. Code Injection and Input Validation

A code injection attack occurs when an input or otherwise external variable is used by an application unsanitized and causes modification to the application logic. This may allow an attacker to gain access to the application device or its data, cause denial of service, or introduce a wide range of malicious side-effects.

The authorization server and client MUST sanitize (and validate when possible) any value received – in particular, the value of the "state" and "redirect_uri" parameters.

1.10.15 10.15. Open Redirectors

The authorization server, authorization endpoint, and client redirection endpoint can be improperly configured and operate as open redirectors. An open redirector is an endpoint using a parameter to automatically redirect a user-agent to the location specified by the parameter value without any validation.

Open redirectors can be used in phishing attacks, or by an attacker to get end-users to visit malicious sites by using the URI authority component of a familiar and trusted destination. In addition, if the authorization server allows the client to register only part of the redirection URI, an attacker can use an open redirector operated by

the client to construct a redirection URI that will pass the authorization server validation but will send the authorization code or access token to an endpoint under the control of the attacker.

1.10.16 10.16. Misuse of Access Token to Impersonate Resource Owner in Implicit

```
    Flow
```

For public clients using implicit flows, this specification does not provide any method for the client to determine what client an access token was issued to.

A resource owner may willingly delegate access to a resource by granting an access token to an attacker's malicious client. This may be due to phishing or some other pretext. An attacker may also steal a token via some other mechanism. An attacker may then attempt to impersonate the resource owner by providing the access token to a legitimate public client.

In the implicit flow (response_type=token), the attacker can easily switch the token in the response from the authorization server, replacing the real access token with the one previously issued to the attacker.

Servers communicating with native applications that rely on being passed an access token in the back channel to identify the user of the client may be similarly compromised by an attacker creating a compromised application that can inject arbitrary stolen access tokens.

Any public client that makes the assumption that only the resource owner can present it with a valid access token for the resource is vulnerable to this type of attack.

This type of attack may expose information about the resource owner at the legitimate client to the attacker (malicious client). This will also allow the attacker to perform operations at the legitimate client with the same permissions as the resource owner who originally granted the access token or authorization code.

Authenticating resource owners to clients is out of scope for this specification. Any specification that uses the authorization process as a form of delegated end-user authentication to the client (e.g., third-party sign-in service) MUST NOT use the implicit flow without additional security mechanisms that would enable the client to determine if the access token was issued for its use (e.g., audience- restricting the access token).

1.11 11. IANA Considerations

1.11.1 11.1. OAuth Access Token Types Registry

This specification establishes the OAuth Access Token Types registry.

Access token types are registered with a Specification Required ([RFC5226]) after a two-week review period on the oauth-ext-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests must be sent to the oauth-ext-review@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request for access token type: example").

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

IANA must only accept registry updates from the Designated Expert(s) and should direct all requests for registration to the review mailing list.

1.11.1.1 11.1.1. Registration Template

Type name:
The name requested (e.g., “example”).

Additional Token Endpoint Response Parameters:
Additional response parameters returned together with the “access_token” parameter. New parameters MUST be separately registered in the OAuth Parameters registry as described by Section 11.2.

HTTP Authentication Scheme(s):
The HTTP authentication scheme name(s), if any, used to authenticate protected resource requests using access tokens of this type.

Change controller:
For Standards Track RFCs, state “IETF”. For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification document(s): Reference to the document(s) that specify the parameter, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

1.11.2 11.2. OAuth Parameters Registry

This specification establishes the OAuth Parameters registry.

Additional parameters for inclusion in the authorization endpoint request, the authorization endpoint response, the token endpoint request, or the token endpoint response are registered with a Specification Required ([RFC5226]) after a two-week review period on the oauth-ext-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests must be sent to the oauth-ext-review@ietf.org mailing list for review and comment, with an appropriate subject (e.g., “Request for parameter: example”).

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

IANA must only accept registry updates from the Designated Expert(s) and should direct all requests for registration to the review mailing list.

1.11.2.1 11.2.1. Registration Template

Parameter name:
The name requested (e.g., “example”).

Parameter usage location:
The location(s) where parameter can be used. The possible locations are authorization request, authorization response, token request, or token response.

Change controller:
For Standards Track RFCs, state “IETF”. For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification document(s):
Reference to the document(s) that specify the parameter, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

1.11.2.2 11.2.2. Initial Registry Contents

The OAuth Parameters registry's initial contents are:

o Parameter name: client_id
o Parameter usage location: authorization request, token request
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: client_secret
o Parameter usage location: token request
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: response_type
o Parameter usage location: authorization request
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: redirect_uri
o Parameter usage location: authorization request, token request
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: scope
o Parameter usage location: authorization request, authorization response, token request, token response
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: state
o Parameter usage location: authorization request, authorization response
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: code
o Parameter usage location: authorization response, token request
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: error_description
o Parameter usage location: authorization response, token response
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: error_uri
o Parameter usage location: authorization response, token response
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: grant_type
o Parameter usage location: token request
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: access_token
o Parameter usage location: authorization response, token response
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: token_type
o Parameter usage location: authorization response, token response
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: expires_in
o Parameter usage location: authorization response, token response
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: username
o Parameter usage location: token request
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: password
o Parameter usage location: token request
o Change controller: IETF
o Specification document(s): RFC 6749

o Parameter name: refresh_token
o Parameter usage location: token request, token response
o Change controller: IETF
o Specification document(s): RFC 6749

1.11.3 11.3. OAuth Authorization Endpoint Response Types Registry

This specification establishes the OAuth Authorization Endpoint Response Types registry.

Additional response types for use with the authorization endpoint are registered with a Specification Required ([RFC5226]) after a two-week review period on the oauth-ext-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests must be sent to the oauth-ext-review@ietf.org mailing list for review and comment, with an appropriate subject (e.g., “Request for response type: example”).

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

IANA must only accept registry updates from the Designated Expert(s) and should direct all requests for registration to the review mailing list.

1.11.3.1 11.3.1. Registration Template

Response type name:
The name requested (e.g., “example”).

Change controller:
For Standards Track RFCs, state “IETF”. For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification document(s):
Reference to the document(s) that specify the type, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

1.11.3.2 11.3.2. Initial Registry Contents

The OAuth Authorization Endpoint Response Types registry’s initial contents are:

o Response type name: code
o Change controller: IETF
o Specification document(s): RFC 6749

o Response type name: token
o Change controller: IETF
o Specification document(s): RFC 6749

1.11.4 11.4. OAuth Extensions Error Registry

This specification establishes the OAuth Extensions Error registry.

Additional error codes used together with other protocol extensions (i.e., extension grant types, access token types, or extension parameters) are registered with a Specification Required ([RFC5226]) after a two-week review period on the oauth-ext-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests must be sent to the oauth-ext-review@ietf.org mailing list for review and comment, with an appropriate subject (e.g., “Request for error code: example”).

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

IANA must only accept registry updates from the Designated Expert(s) and should direct all requests for registration to the review mailing list.

1.11.4.1 11.4.1. Registration Template

Error name:
The name requested (e.g., “example”). Values for the error name MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.

Error usage location:
The location(s) where the error can be used. The possible locations are authorization code grant error response (Section 4.1.2.1), implicit grant error response (Section 4.2.2.1), token error response (Section 5.2), or resource access error response (Section 7.2).

Related protocol extension:
The name of the extension grant type, access token type, or extension parameter that the error code is used in conjunction with.

Change controller:
For Standards Track RFCs, state “IETF”. For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification document(s):
Reference to the document(s) that specify the error code, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

1.12 12. References

1.12.1 12.1. Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997.

[RFC2246] Dierks, T. and C. Allen, “The TLS Protocol Version 1.0”, RFC 2246, January 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication”, RFC 2617, June 1999.

[RFC2818] Rescorla, E., “HTTP Over TLS”, RFC 2818, May 2000.

[RFC3629] Yergeau, F., “UTF-8, a transformation format of ISO 10646”, STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax”, STD 66, RFC 3986, January 2005.

[RFC4627] Crockford, D., “The application/json Media Type for JavaScript Object Notation (JSON)”, RFC 4627, July 2006.

[RFC4949] Shirey, R., “Internet Security Glossary, Version 2”, RFC 4949, August 2007.

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs”, BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF”, STD 68, RFC 5234, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2”, RFC 5246, August 2008.

[RFC6125] Saint-Andre, P. and J. Hodges, “Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)”, RFC 6125, March 2011.

[USASCII] American National Standards Institute, “Coded Character Set – 7-bit American Standard Code for Information Interchange”, ANSI X3.4, 1986.

[W3C.REC-html401-19991224]
Raggett, D., Le Hors, A., and I. Jacobs, “HTML 4.01 Specification”, World Wide Web Consortium Recommendation REC-html401-19991224, December 1999, http://www.w3.org/TR/1999/REC-html401-19991224.

[W3C.REC-xml-20081126]
Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, “Extensible Markup Language (XML) 1.0 (Fifth Edition)”, World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, http://www.w3.org/TR/2008/REC-xml-20081126.

1.12.2 12.2. Informative References

[OAuth-HTTP-MAC]
Hammer-Lahav, E., Ed., “HTTP Authentication: MAC Access Authentication”, Work in Progress, February 2012.

[OAuth-SAML2]
Campbell, B. and C. Mortimore, “SAML 2.0 Bearer Assertion Profiles for OAuth 2.0”, Work in Progress, September 2012.

[OAuth-THREATMODEL]
Lodderstedt, T., Ed., McGloin, M., and P. Hunt, “OAuth 2.0 Threat Model and Security Considerations”, Work in Progress, October 2012.

[OAuth-WRAP]
Hardt, D., Ed., Tom, A., Eaton, B., and Y. Goland, “OAuth Web Resource Authorization Profiles”, Work in Progress, January 2010.

[RFC5849] Hammer-Lahav, E., “The OAuth 1.0 Protocol”, RFC 5849, April 2010.

[RFC6750] Jones, M. and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage”, RFC 6750, October 2012.

1.13 Appendix A. Augmented Backus-Naur Form (ABNF) Syntax

This section provides Augmented Backus-Naur Form (ABNF) syntax descriptions for the elements defined in this specification using the notation of [RFC5234]. The ABNF below is defined in terms of Unicode code points [W3C.REC-xml-20081126]; these characters are typically encoded in UTF-8. Elements are presented in the order first defined.

Some of the definitions that follow use the “URI-reference” definition from [RFC3986].

Some of the definitions that follow use these common definitions:

```
VSCHAR      = %x20-7E
NQCHAR      = %x21 / %x23-5B / %x5D-7E
NQSCHAR     = %x20-21 / %x23-5B / %x5D-7E
UNICODECHARNOCRLF = %x09 /%x20-7E / %x80-D7FF /
                    %xE000-FFFD / %x10000-10FFFF
```

(The UNICODECHARNOCRLF definition is based upon the Char definition in Section 2.2 of [W3C.REC-xml-20081126], but omitting the Carriage Return and Linefeed characters.)

1.13.1 A.1. “client_id” Syntax

The “client_id” element is defined in Section 2.3.1:

```
client-id     = *VSCHAR
```

1.13.2 A.2. “client_secret” Syntax

The “client_secret” element is defined in Section 2.3.1:

```
client-secret = *VSCHAR
```

1.13.3 A.3. “response_type” Syntax

The “response_type” element is defined in Sections 3.1.1 and 8.4:

```
response-type = response-name *( SP response-name )
response-name = 1*response-char
response-char = "_" / DIGIT / ALPHA
```

1.13.4 A.4. “scope” Syntax

The “scope” element is defined in Section 3.3:

```
scope       = scope-token *( SP scope-token )
scope-token = 1*NQCHAR
```

1.13.5 A.5. “state” Syntax

The “state” element is defined in Sections 4.1.1, 4.1.2, 4.1.2.1, 4.2.1, 4.2.2, and 4.2.2.1:

```
state      = 1*VSCHAR
```

1.13.6 A.6. “redirect_uri” Syntax

The “redirect_uri” element is defined in Sections 4.1.1, 4.1.3, and 4.2.1:

```
redirect-uri      = URI-reference
```

1.13.7 A.7. “error” Syntax

The “error” element is defined in Sections 4.1.2.1, 4.2.2.1, 5.2, 7.2, and 8.5:

```
error             = 1*NQSCHAR
```

1.13.8 A.8. “error_description” Syntax

The “error_description” element is defined in Sections 4.1.2.1, 4.2.2.1, 5.2, and 7.2:

```
error-description = 1*NQSCHAR
```

1.13.9 A.9. “error_uri” Syntax

The “error_uri” element is defined in Sections 4.1.2.1, 4.2.2.1, 5.2, and 7.2:

```
error-uri         = URI-reference
```

1.13.10 A.10. “grant_type” Syntax

The “grant_type” element is defined in Sections 4.1.3, 4.3.2, 4.4.2, 4.5, and 6:

```
grant-type = grant-name / URI-reference
grant-name = 1*name-char
name-char  = "-" / "." / "_" / DIGIT / ALPHA
```

1.13.11 A.11. “code” Syntax

The “code” element is defined in Section 4.1.3:

```
code       = 1*VSCHAR
```

1.13.12 A.12. “access_token” Syntax

The “access_token” element is defined in Sections 4.2.2 and 5.1:

```
access-token = 1*VSCHAR
```

1.13.13 A.13. “token_type” Syntax

The “token_type” element is defined in Sections 4.2.2, 5.1, and 8.1:

```
token-type = type-name / URI-reference
type-name  = 1*name-char
name-char  = "-" / "." / "_" / DIGIT / ALPHA
```

1.13.14 A.14. “expires_in” Syntax

The “expires_in” element is defined in Sections 4.2.2 and 5.1:

```
expires-in = 1*DIGIT
```

1.13.15 A.15. “username” Syntax

The “username” element is defined in Section 4.3.2:

```
username = *UNICODECHARNOCRLF
```

1.13.16 A.16. “password” Syntax

The “password” element is defined in Section 4.3.2:

```
password = *UNICODECHARNOCRLF
```

1.13.17 A.17. “refresh_token” Syntax

The “refresh_token” element is defined in Sections 5.1 and 6:

```
refresh-token = 1*VSCHAR
```

1.13.18 A.18. Endpoint Parameter Syntax

The syntax for new endpoint parameters is defined in Section 8.2:

```
param-name = 1*name-char
name-char  = "-" / "." / "_" / DIGIT / ALPHA
```

1.14 Appendix B. Use of application/x-www-form-urlencoded Media Type

At the time of publication of this specification, the"application/x-www-form-urlencoded" media type was defined inSection 17.13.4 of [W3C.REC-html401-19991224] but not registered inthe IANA MIME Media Types registry(http://www.iana.org/assignments/media-types). Furthermore, thatdefinition is incomplete, as it does not consider non-US-ASCIIcharacters.

To address this shortcoming when generating payloads using this mediatype, names and values MUST be encoded using the UTF-8 characterencoding scheme [RFC3629] first; the resulting octet sequence thenneeds to be further encoded using the escaping rules defined in[W3C.REC-html401-19991224].

When parsing data from a payload using this media type, the names andvalues resulting from reversing the name/value encoding consequentlyneed to be treated as octet sequences, to be decoded using the UTF-8character encoding scheme.

For example, the value consisting of the six Unicode code points
(1) U+0020 (SPACE), (2) U+0025 (PERCENT SIGN),
(3) U+0026 (AMPERSAND), (4) U+002B (PLUS SIGN),
(5) U+00A3 (POUND SIGN), and (6) U+20AC (EURO SIGN) would be encoded into the octet sequence below (using hexadecimal notation):

```
20 25 26 2B C2 A3 E2 82 AC
```

and then represented in the payload as:

```
+%25%26%2B%C2%A3%E2%82%AC
```

1.15 Appendix C. Acknowledgements

The initial OAuth 2.0 protocol specification was edited by David Recordon, based on two previous publications: the OAuth 1.0 community specification [RFC5849], and OAuth WRAP (OAuth Web Resource Authorization Profiles) [OAuth-WRAP]. Eran Hammer then edited many of the intermediate drafts that evolved into this RFC. The Security Considerations section was drafted by Torsten Lodderstedt, Mark McGloin, Phil Hunt, Anthony Nadalin, and John Bradley. The section on use of the “application/x-www-form-urlencoded” media type was drafted by Julian Reschke. The ABNF section was drafted by Michael B. Jones.

The OAuth 1.0 community specification was edited by Eran Hammer and authored by Mark Atwood, Dirk Balfanz, Darren Bounds, Richard M. Conlan, Blaine Cook, Leah Culver, Breno de Medeiros, Brian Eaton, Kellan Elliott-McCrea, Larry Halff, Eran Hammer, Ben Laurie, Chris Messina, John Panzer, Sam Quigley, David Recordon, Eran Sandler, Jonathan Sergent, Todd Sieling, Brian Slesinsky, and Andy Smith.

The OAuth WRAP specification was edited by Dick Hardt and authored by Brian Eaton, Yaron Y. Goland, Dick Hardt, and Allen Tom.

This specification is the work of the OAuth Working Group, which includes dozens of active and dedicated participants. In particular, the following individuals contributed ideas, feedback, and wording that shaped and formed the final specification:

Michael Adams, Amanda Anganes, Andrew Arnott, Dirk Balfanz, Aiden Bell, John Bradley, Marcos Caceres, Brian Campbell, Scott Cantor, Blaine Cook, Roger Crew, Leah Culver, Bill de hOra, Andre DeMarre, Brian Eaton, Wesley Eddy, Wolter Eldering, Brian Ellin, Igor Faynberg, George Fletcher, Tim Freeman, Luca Frosini, Evan Gilbert, Yaron Y. Goland, Brent Goldman, Kristoffer Gronowski, Eran Hammer, Dick Hardt, Justin Hart, Craig Heath, Phil Hunt, Michael B. Jones, Terry Jones, John Kemp, Mark Kent, Raffi Krikorian, Chasen Le Hara, Rasmus Lerdorf, Torsten Lodderstedt, Hui-Lan Lu, Casey Lucas, Paul Madsen, Alastair Mair, Eve Maler, James Manger, Mark McGloin, Laurence Miao, William Mills, Chuck Mortimore, Anthony Nadalin, Julian Reschke, Justin Richer, Peter Saint-Andre, Nat Sakimura, Rob Sayre, Marius Scurtescu, Naitik Shah, Luke Shepard, Vlad Skvortsov, Justin Smith, Haibin Song, Niv Steingarten, Christian Stuebner, Jeremy Suriel, Paul Tarjan, Christopher Thomas, Henry S. Thompson, Allen Tom, Franklin Tse, Nick Walker, Shane Weeden, and Skylar Woodward.

This document was produced under the chairmanship of Blaine Cook, Peter Saint-Andre, Hannes Tschofenig, Barry Leiba, and Derek Atkins. The area directors included Lisa Dusseault, Peter Saint-Andre, and Stephen Farrell.

Author’s Address

Dick Hardt (editor)
Microsoft

EMail: dick.hardt@gmail.com
URI: http://dickhardt.org/