Rhino Accounts Overview

Table of Contents

About Rhino Accounts
Using Rhino Accounts for Authorization as a Client
Using Rhino Accounts for Authorization as a Resource Server
Using Rhino Accounts for Authentication as a Client
Appendix

About Rhino Accounts

What is Rhino Accounts?

An authorization *and* authentication framework that can be used by applications to delegate access to resources and verify the identity of users. With Rhino Accounts, the goal is to make it easy for multiple services within Rhino and related 3rd parties to share resources and authenticate users by using a centralized system. For the end-user, a single account is all that's needed to access all Rhino related services.

What is Rhino Accounts based on?

It is based on OAuth 2.0, per the RFC 6749 specifications. Rhino Accounts implements all the features of OAuth 2.0 that are prefixed with `REQUIRED` and `MUST`, and avoids implementing any behavior that is prefixed with `MUST NOT`. Every attempt is made to implement features that are prefixed with `SHOULD`. Likewise, behavior prefixed with `SHOULD NOT` is avoided when feasible, particularly if it has security implications. When deemed advantageous for Rhino, features labeled as `OPTIONAL` have been implemented. In general, the implementation should allow any 3rd party OAuth 2.0/OpenID Connect client to take advantage of the system with little or no modification.

Authentication vs Authorization

Rhino Accounts aims to satisfy both needs. The need for authentication occurs when a client needs to verify the identity of a user. For example, eLIS needs to know that a user claiming to be Marley Rhino is indeed Marley Rhino when creating a shipment. The need for authorization occurs when a client needs to access the user's protected resources. For example, the Remote API in eLIS might need an access token to perform certain actions on behalf of a user. Rhino Accounts, based on OAuth 2.0 and OpenID Connect, can perform either of these needs individually or simultaneously.

A note about security.

Rhino Accounts is designed with the best intentions as an authorization/authentication provider. There is extensive development time devoted to security. However, developers should note that comprehensive security can only be achieved by trust. In particular, the client implementation of

OAuth 2.0 as well as the user agent play a large role in the overall level of security achieved in an OAuth 2.0 ecosystem, and Rhino Accounts is reliant on them to provide a secure environment.

User Agent Requirements.

Rhino Accounts has slightly more stringent user agent requirements (browsers released in the last ~6 years) than most other OAuth 2.0 providers. This is done for 3 reasons:

1. Only browsers that are still supported by their vendors are supported. This is done to ensure that if any security holes are discovered in the browser, the vendor will be encouraged to fix it.
2. The UI is based on the same framework as eLIS (The *eds* JavaScript framework), which offers a more interactive user experience. The *eds* framework requires certain HTML5 features which are only available in more recent browsers.
3. Rhino Accounts takes advantage of several newer security features regarding cookies and TLS that unfortunately have only been implemented (or implemented correctly) on recent browsers.

Supported Browsers:

- Microsoft IE 10 or later (11 or later recommended; Edge is supported)
- Apple Safari 6 or later
- Google Chrome
- Mozilla Firefox
- Opera

Client Registration

Contact support@mcneel.com to register a client with Rhino Accounts.

Using Rhino Accounts for Authorization as a Client

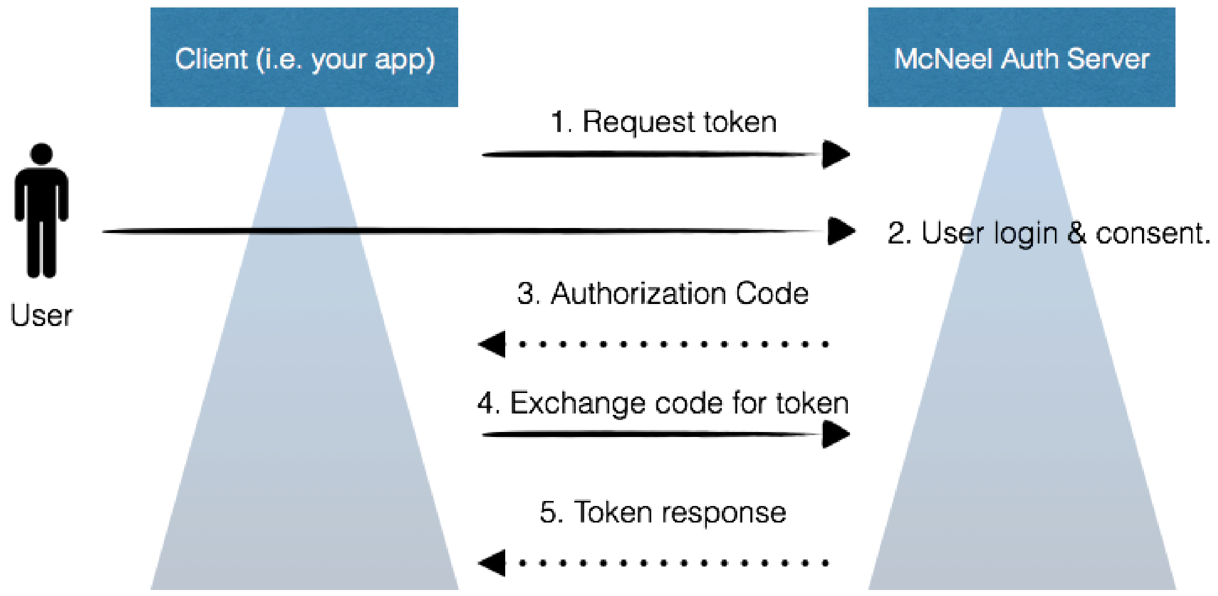
Overview

Rhino Accounts supports two different authorization workflows depending on the type of your application: *Authorization Code* flow and the *Implicit* flow. In general, applications that are *confidential,* such as a website hosted in a web server, should always use the *Authorization Code* flow for increased security. In contrast, applications that are *not confidential,* such as an application running on a user's device like Rhinoceros, should use the *Implicit* flow, which requires additional security considerations since they are not able to keep secrets confidential to the user or a malicious third party. In either case, your application will need to be registered as a client in Rhino Accounts before your application performs any requests. The documentation below explains in detail how a client interacts with Rhino Accounts–keep in mind that a reliable client library will implement most of the details for you if you choose to use one.

WARNING: Do not use any of the two authorization flows for authentication. Instead, see Using Rhino Accounts for Authentication as a Client

Using the Authorization Code flow:

The authorization sequence begins when your application redirects a browser to a Rhino URL; the URL includes query parameters that indicate the type of access being requested. Rhino Accounts handles the user authentication, session selection, and user consent. The result is an authorization code, which the client (i.e. your application) can exchange for an access token, that can be ultimately used to access the user's protected resources.



Preparing to start the flow:

Make sure you know your app's client ID and client secret, which you will have provided when registering your app as a client in Rhino Accounts. In addition, you will need to know the scopes (such as `profile` or `email`) that your app needs to request. See the documentation for the APIs your app uses for the required scopes.

Requesting a token

Generate either a `GET` or a `application/x-www-form-urlencoded POST` request to [`https://accounts.rhino3d.com/oauth2/auth`](https://accounts.rhino3d.com/oauth2/auth). This endpoint is accessible over HTTPS; plain HTTP connections are refused. To avoid capturing request data in browser caches or logs, you should strive to use a `POST` request over a `GET` request if possible. The request should have the following parameters:

Parameter	Values	Details
`response_type`	`code`	For the Authorization Code flow, clients should use `code`.
`client_id`	The client ID of your application	Identifies the client that is making the request. The value passed in this parameter must exactly match the value of your application.
`redirect_uri`	One of the `redirect_uri` values listed for your client.	Determines where the response is sent. The value of this parameter must exactly match one of the values listed for your client. (including the http or https scheme, case, and trailing '/').
`scope`	Space-delimited set of permissions that the application requests.	Optional. Identifies the API access that your client is requesting. The values passed in this parameter inform the consent screen that is shown to the user. If no scope is given, `profile` will be assigned.
`state`	Any string under 1024 chars.	Optional. Provides any state that might be useful to your application upon receipt of the response. The Rhino Accounts Server roundtrips this parameter, so your application receives the same value it sent. To mitigate against cross-site request forgery (CSRF), it is strongly recommended to include an anti-forgery token in the state, and confirm it in the response.

`prompt`	`none`
	`consent`
	`login`
		Optional. A space delimited list of string values that specifies whether the authorization server should attempt to prompt the user for reauthentication and consent. Note that the request is not guaranteed to be fulfilled. The possible values are:
		`none` The authorization server does not display any authentication or user consent screens; it will return an error if the user is not already authenticated and has not pre-configured consent for the requested scopes. You can use none to check for existing authentication and/or consent.
		`consent` The authorization server prompts the user for consent before returning information to the client.

		`login` The authorization server prompts the user to reauthenticate before returning information to the client.
`max_age`	A positive number denoting seconds.	Specifies the allowable elapsed time in seconds since the last time the user was actively authenticated by Rhino Accounts. If the elapsed time is greater than this value, the behavior is the same as specifying `login` for `prompt`.
`login_hint`	A string representing either a user's id (`sub`) or an email address associated with their account	If your client knows which user it is expecting to login, you may specify said user using this field preferably using the user's unique id (`sub`) or optionally an email address. Note that this field is merely a suggestion to Rhino Accounts and may not be heeded due to various security and/or technical reasons.

Exchanging an authorization code for a token

The Rhino Accounts server responds to your application's access request by using the URL specified in the request. If the user approves the access request, then the response contains an authorization code. If the user does not approve the request or the request is invalid, the response contains an error code based on RFC 6749 . All responses are returned to your web server on the query string, as shown below. Keep in mind that the authorization code is url safe, which means you don't have to url-encode it when sending it to the Rhino Accounts server (but you could).

An error response:
```
https://demo.appspot.com/auth?error=access_denied&state=MY_STATE_HERE
```

An authorization code response:
```
https://demo.appspot.com/auth?code=AUTH_CODE_HERE&state=MY_STATE_HERE
```

To exchange an authorization code for an access token, make a `application/x-www-form-urlencoded POST` request with Basic Authentication to `https://accounts.rhino3d.com/oauth2/token`. The `Authorization` header must have `client_id:client_secret` encoded in base64 as its value, where `client_id` is the id of your application and `client_secret` is the secret registered with your application in Rhino Accounts. The request must have the following parameters:

Parameter	Values
`code`	The authorization code returned from the initial request.
`redirect_uri`	One of the `redirect_uri` values listed for your client. The value here must exactly match the value passed in for the initial authorization code request.
`grant_type`	As defined in the OAuth 2.0 specification, this field must contain a value of `authorization_code`.

An example request:

```
POST
Authorization: Basic AKedpskdasdfw==
https://accounts.rhino3d.com/oauth2/token?
code=ACCESS_CODE_HERE&
redirect_uri=https://demo.appspot.com&
grant_type=authorization_code
```

A successful response to this request contains the following JSON response, where `access_token` is the token that can be used to access the user's protected resources, and `expires_in` is the remaining lifetime of the `access_token` in seconds:

```
{
  "access_token":ACCESS_TOKEN_HERE,
  "expires_in":3920,
  "scope":”profile”,
  "token_type":”bearer”
}
```

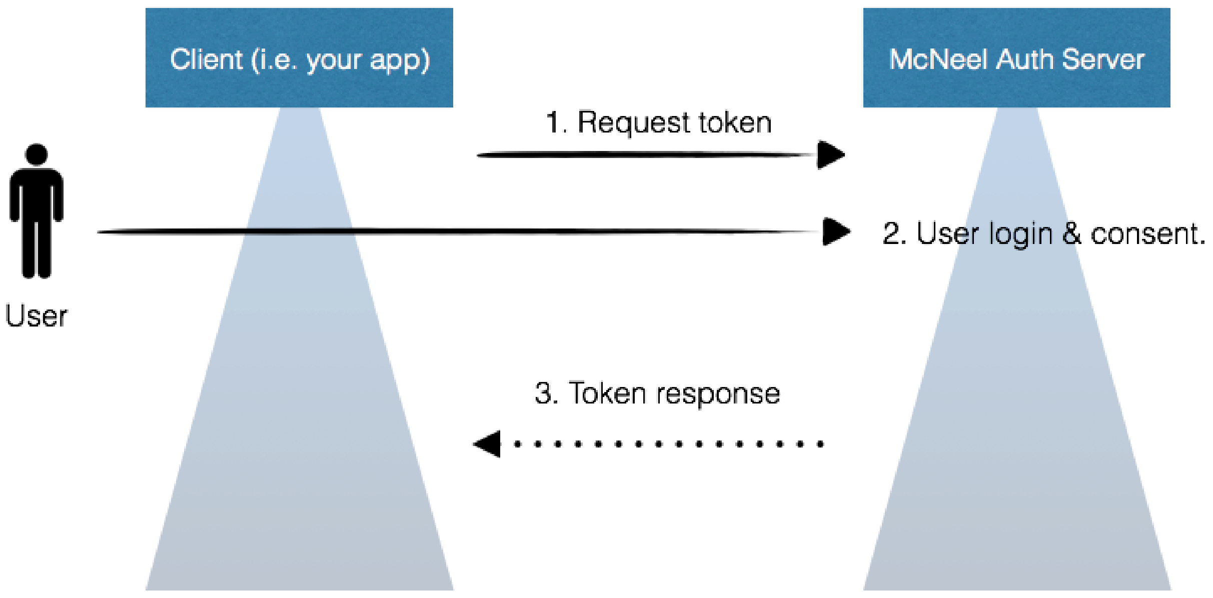
Keep in mind the `access_token` is url safe, and does not have to be url-encoded when sent to a resource server.

Revoking a valid access token

See Revoking an access token programmatically

Using the Implicit flow:

The authorization sequence begins when your application redirects a browser to a Rhino URL; the URL includes query parameters that indicate the type of access being requested. Rhino Accounts handles the user authentication, session selection, and user consent.
The result is an access token that can be used to access the user's protected resources.



Requesting a token

Generate either a `GET` or a `application/x-www-form-urlencoded POST` request to https://accounts.rhino3d.com/oauth2/auth. This endpoint is accessible over HTTPS; plain HTTP connections are refused. To avoid capturing request data in browser caches or logs, you should strive to use a `POST` request over a `GET` request when possible. The request should have the following parameters:

Parameter	Values	Details
`response_type`	`token`	For the Implicit flow, clients should use `token`.
`client_id`	The client ID of your application	Identifies the client that is making the request. The value passed in this parameter must exactly match the value of your application.
`redirect_uri`	One of the `redirect_uri` values listed for your client.	Determines where the response is sent. The value of this parameter must exactly match one of the values listed for your client. (including the http or https scheme, case, and trailing '/').
`scope`	Space-delimited set of permissions that the application requests.	Optional. Identifies the API access that your client is requesting. The values passed in this parameter inform the consent screen that is shown to the user. If no scope is given, `profile` will be assigned.
`state`	Any string under	Optional. Provides any state that might be useful

	1024 chars.	to your application upon receipt of the response. The Rhino Accounts Server roundtrips this parameter, so your application receives the same value it sent. To mitigate against cross-site request forgery (CSRF), it is strongly recommended to include an anti-forgery token in the state, and confirm it in the response.

`prompt`	`none`
	`consent`
	`login`
		Optional. A space delimited list of string values that specifies whether the authorization server should attempt to prompt the user for reauthentication and consent. Note that the request is not guaranteed to be fulfilled. The possible values are:
		`none` The authorization server does not display any authentication or user consent screens; it will return an error if the user is not already authenticated and has not pre-configured consent for the requested scopes. You can use none to check for existing authentication and/or consent.
		`consent` The authorization server prompts the user for consent before returning information to the client.
		`login` The authorization server prompts the user to reauthenticate before returning information to the client.

`max_age`	A positive number denoting seconds.	Specifies the allowable elapsed time in seconds since the last time the user was actively authenticated by Rhino Accounts. If the elapsed time is greater than this value, the behavior is the same as specifying `login` for `prompt`.

Handling the token response

Rhino Accounts returns an access token to your application if the user grants your application the permissions it requested, or an error response based on RFC 6749 if the user denies the request or if the request is invalid. The access token is returned to your application in the fragment as part of the `access_token` parameter. Since a fragment is not returned to the server, client-side code must parse the fragment and extract the value of the `access_token` parameter.

The other parameters included in the response are `expires_in`, `token_type`, `scope`, and `state`. These parameters describe the lifetime of the token in seconds, the kind of token that is being returned, the scope of the access token, and the state parameter with the same value as the value sent in the original request.

An error response:
```
https://demo.appspot.com/oauthcallback#error=access_denied&state=STAT
E
```

A token response:
```
https://demo.appspot.com/oauthcallback#access_token=ACCES_TOKEN_HERE&
token_type=bearer&expires_in=3600&scope=email+profile
```

Revoking a valid access token
See Revoking an access token programmatically

Using Rhino Accounts for Authorization as a Resource Server

Overview
As a resource server, such as a protected file server or a license storage system, you can require requests to your service to include an access token issued by Rhino Accounts.
You can then ask Rhino Accounts to validate the access token to ensure it is valid, and (perhaps) see whether the access token meets the authorization criteria for the request. This spares the resource server from implementing time-consuming and potentially vulnerable authorization mechanisms. The implementation details on how a resource server requests an access token from a client is left up to the designers of such system, but they MUST implement TLS to keep the access tokens confidential at all times.

Validating an access token
To validate an access token, make a `GET` request to `https://accounts.rhino3d.com/oauth2/tokeninfo` with Bearer Authentication. The `Authorization` header must have the access token as shown in the example below.

An example request:
```
GET
Authorization: Bearer ACCESS_TOKEN_HERE
https://accounts.rhino3d.com/oauth2/tokeninfo
```

An example response:
```
{"scope": "profile", "audience": "client_001", "expires_in": 86002}
```

The response will have a status code of 200 if successful, or a status code equal or greater to 400 if there was a problem, along with an error message as defined in RFC 6749.

The resource server is expected to audit the response's values to determine whether the token is sufficient for the request it has been asked to fulfill.

Using Rhino Accounts for Authentication as a Client

Rhino Accounts is OpenID Connect compliant and supports three different authentication workflows depending on the type of your application. Because OpenID Connect is built on top of OAuth2, you can ask Rhino Accounts for both authorization and authentication in a single request. The three flows supported are : *Authorization Code* flow, the *Implicit* flow, and the *Implicit id_token only* flow. In general, applications that are *confidential,* such as a website hosted in a web server, should always use the *Authorization Code* flow for increased security. In contrast, applications that are *not confidential,* such as an application running on a user's device like Rhinoceros, should use the *Implicit* flow, which requires additional security considerations since they are not able to keep secrets confidential to the user or a malicious third party. The *Implicit id_token only* flow is identical to the implicit flow, except that it can only be used for authentication and not authorization. It can be used if you are a non confidential application and only require Rhino Accounts to perform authentication on your behalf. In either case, your application will need to be registered as a client in Rhino Accounts before your application performs any requests. The documentation below explains in detail how a client interacts with Rhino Accounts–keep in mind that a reliable OpenID Connect client library will implement most of the details for you if you choose to use one.

Using the Authorization Code flow:

The flow is identical to the flow used for authorization, except as explained below. All the parameters used for authorization are supported, and you can use them to request authorization and authentication in a single step.

In this flow, you must include `openid` in the set of `scope` values. This will cause Rhino Accounts to issue an OpenID Connect token together with the access token. In addition, the following parameters are supported in addition to all the parameters used for authorization:

`nonce`	Any string.	Recommended. String value used to associate a Client session with an OpenID Connect token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token.

An example response from the token endpoint:

```
{
  "access_token":ACCESS_TOKEN_HERE,
  "id_token":OPENID_CONNECT_TOKEN_HERE,
  "expires_in":3920,
  "scope":”profile”,
  "token_type":”bearer”
}
```

You can view more information about the OpenID Connect token and how you can use it to verify the client’s identity in the appendix.

Using the Implicit flow:

The flow is identical to the flow used for authorization, except as explained below. All the parameters used for authorization are supported, and you can use them to request authorization and authentication in a single step.

In this flow, you must include `openid` in the set of `scope` values. In addition, the `response_type` must be set to `id_token token`. This will cause Rhino Accounts to issue an OpenID Connect token together with the access token. In addition, the following parameters are supported in addition to all the parameters used for authorization:

`nonce`	Any string.	Recommended. String value used to associate a Client session with an OpenID Connect token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token.

An example response from the token endpoint:

```
https://demo.appspot.com/oauthcallback#access_token=ACCES_TOKEN_HERE&
token_type=bearer&expires_in=3600&scope=email+profile&id_token=OPENID
_CONNECT_TOKEN_HERE
```

You can view more information about the OpenID Connect token and how you can use it to verify the client’s identity in the appendix. Note that if you’re trying to verify a client’s identity using an implicit flow, you must take additional precautions that are out the scope for this document.

Using the id_token only flow:
This flow is identical to the *Implicit* flow, except that no access token is returned; only an OpenID Connect token is returned. This flow is useful for applications that only want to perform authentication and not authorization.

In this flow, you must include `openid` in the set of `scope` values. In addition, the `response_type` must be set to `id_token`. This will cause Rhino Accounts to issue an OpenID Connect token but no access token. In addition, the following parameters are supported in addition to all the parameters used for authorization:

`nonce`	Any string.	Recommended. String value used to associate a Client session with an OpenID Connect token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token.

An example response from the token endpoint:
`https://demo.appspot.com/oauthcallback#id_token=OPENID_CONNECT_TOKEN_HERE`

You can view more information about the OpenID Connect token and how you can use it to verify the client's identity in the appendix. Note that if you're trying to verify a client's identity using an implicit flow, you must take additional precautions that are out the scope for this document.

Appendix

Revoking an access token programmatically

OpenID Connect Tokens

RFC6749 security considerations implemented by Rhino Accounts

RFC6819 security considerations implemented by Rhino Accounts

Revoking an access token programmatically
Sometimes it is desirable for an application to invalidate a valid token it holds, perhaps because a user no longer desires using the service it provides. To revoke a token (i.e. make it unusable),

make a `application/x-www-form-urlencoded POST` request `https://accounts.rhino3d.com/oauth2/revoke`.

An example request:
```
POST
https://accounts.rhino3d.com/oauth2/revoke&token=ACCESS_TOKEN_HERE
```

The response will be empty and have a status code of 200 if successful, or a status code equal or greater to 400 if there was a problem, along with an error message as defined in RFC 6749.

OpenID Connect Tokens

In all flows used for authentication, Rhino Accounts issues an OpenID Connect token (ID token). ID tokens are cryptographically signed JSON Web Tokens (JWTs). They are signed by Rhino Accounts. If you received a token using the implicit flow or using the authorization code flow indirectly from the Rhino Accounts server, you MUST verify the signature of the token before reading its contents as described here. TLS server verification may be a substitute for verifying the signature of the token if and only if you received the id token directly from Rhino Accounts in the authorization code flow. ID tokens contain “claims” about a user, such as their unique id, their name, or their locale. The amount of claims included in an ID token depends on the scope specified when the request for authentication was done. At the very least, all ID tokens are guaranteed to include the user’s unique id for verifying their identity. The following is a list of possible claims included in an access token:

Guaranteed claims (they appear in every ID token issued by Rhino Accounts):

- `iat` Time at which the JWT was issued. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.
- `exp` Expiration time on or after which the ID Token MUST NOT be accepted for processing. The processing of this parameter requires that the current date/time MUST be before the expiration date/time listed in the value. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.
- `iss` Issuer Identifier for the Issuer of the response. The iss value is a case sensitive URL using the https scheme that contains scheme, host, and optionally, port number and path components and no query or fragment components. This value will always be *https://accounts.rhino3d.com*
- `aud` The client this ID Token is intended for. It contains the OAuth 2.0 id of the client who requested the token.
- `auth_time` Last time active user authentication occurred. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

- `sub` Subject Identifier. A locally unique and never reassigned identifier within Rhino Accounts for the user, which is intended to be consumed by the Client. This value is always a string.

The following claims may also be present (not comprehensive):

- `nonce` String value used to associate a Client session with an ID Token, and to mitigate replay attacks. The value is passed through unmodified from the Authentication Request to the ID Token. This value is guaranteed to appear when you specify a nonce in the request.
- `at_hash` Access Token hash value. Its value is the base64url encoding of the left-most half of the hash of the octets of the ASCII representation of the access_token value, where the hash algorithm used is the hash algorithm used in the alg Header Parameter of the ID Token's JOSE Header. For instance, if the alg is RS256, hash the access_token value with SHA-256, then take the left-most 128 bits and base64url encode them. This value is guaranteed to appear if an ID token is issued together with an access token.
- `email` The primary email of the user. Present when `email` is part of the scope.
- `email_verified` A boolean specifying whether or not Rhino Accounts has verified the email address. Present when `email` is part of the scope.
- `phone_number` The phone number of the user. Rhino Accounts will validate all phone numbers and return them in E.164 format for international communication. Present when `phone` is part of the scope.
- `phone_number_verified` A boolean specifying whether or not Rhino Accounts has verified the phone number. Present when `phone` is part of the scope.
- `name` The name of the user. Present when `profile` is part of the scope.
- `address` A dictionary representing the address of the user. The granularity of this claim is not guaranteed. Present when `address` is part of the scope.
- `locale` The locale of the user (using ISO standards, i.e. en-US or es-CO). Present when `profile` is part of the scope.
- `picture` The URL of the user's profile picture. Present when `profile` is part of the scope.

JSON Web Key Set

The public key that corresponds to the private key used to encode the ID token is available at https://accounts.rhino3d.com/oauth2/keys. This is also known as the “jwks_uri”.

Using the UserInfo endpoint

Rhino Accounts provides a userinfo endpoint that adheres to OpenID Connect standards (but can be used with a plain access token as well). This endpoint is a protected resource that can be accessed using an OAuth2.0 access token issued by Rhino Accounts. It provides information about an end user in JSON form that would normally be issued as “claims” in an OpenID

Connect token. Note that the information returned depends on the scope of the access token provided.

To access the endpoint, make a `GET` request to `https://accounts.rhino3d.com/oauth2/userinfo` with Bearer Authentication. The `Authorization` header must have the access token as shown in the example below.

An example request:

```
GET
Authorization: Bearer ACCESS_TOKEN_HERE
https://accounts.rhino3d.com/oauth2/userinfo
```

An example response:

```
{"email": "marley_the_dog@mcneel.com",
 "name": "Marley",
 "locale": "en-gb"
}
```

The response will have a status code of 200 if successful, or a status code equal or greater to 400 if there was a problem, along with an error message as defined in RFC 6749.

Using the GroupInfo endpoint

Rhino Accounts provides a groupinfo endpoint that obtains information about a group in the system. This endpoint is a protected resource that can be accessed using an OAuth2.0 access token issued by Rhino Accounts. Note that the access token used MUST have the scope *groups* for a successful response.

To access the endpoint, make a `GET` request to `https://accounts.rhino3d.com/oauth2/groupinfo` with Bearer Authentication. The `Authorization` header must have the access token as shown in the example below.

An example request:

```
GET
Authorization: Bearer ACCESS_TOKEN_HERE
https://accounts.rhino3d.com/oauth2/groupinfo?group_id=GROUP_ID_HERE
```

An example response:

```
{
    "description": "Marley's Favorite Team",
    "creationDate": "2017-06-09T21:00:29.566Z",
    "id": "5677643768791040",
    "name": "McNeel"
}
```

The response will have a status code of 200 if successful, or a status code equal or greater to 400 if there was a problem, along with an error message as defined in RFC 6749.

Using the MemberInfo endpoint

Rhino Accounts provides a memberinfo endpoint that obtains information about a group's members in the system. This endpoint is a protected resource that can be accessed using an OAuth2.0 access token issued by Rhino Accounts. Note that the access token used MUST have the scope *groups* for a successful response.

To access the endpoint, make a `GET` request to `https://accounts.rhino3d.com/oauth2/memberinfo` with Bearer Authentication. The `Authorization` header must have the access token as shown in the example below.

An example request:

```
GET
Authorization: Bearer ACCESS_TOKEN_HERE
https://accounts.rhino3d.com/oauth2/memberinfo?group_id=GROUP_ID_HERE
```

An example response:

```
[
    {
        "picture": "https://urltopic.com/image.png",
        "role": "owner",
        "sub": "6740179150700544",
        "name": "Marley The Dog",
        "email": "marley@mcneel.com"
    },
    {
        "picture": "https://urltopic.com/image.png",
        "role": "member",
        "sub": "6754232283693056",
        "name": "Andrés Jacobo",
        "email": "aj@mcneel.com"
    }
]
```

The response will have a status code of 200 if successful, or a status code equal or greater to 400 if there was a problem, along with an error message as defined in RFC 6749.

Available Grants (Scopes)

The following grants can be requested using the *scope* parameter:

`openid`	Required for any authentication flow. Allows clients to see the user's unique ID.
`email`	Allows clients to see all email addresses for a user.
`profile`	Allows clients to get basic profile information such as the user's name and profile picture.
`phone`	Allows clients to see the user's phone number.
`address`	Allows clients to see the user's location.
`groups`	Allows clients to see the user's groups and their members.
`noexpire`	Issues a token that will never expire (But that can be revoked at any time by a user or the client).

RFC6749 security considerations implemented by Rhino Accounts

The following is a brief description of how Rhino Accounts meets all security criteria considered a `MUST` as described in Section 10 of RFC 6749. Items considered as `SHOULD` or `SHOULD NOT` are sometimes covered in this list when it is of particular interest. Client and user agent requirements are not mentioned on this list. Purely optional features (i.e. features labeled as `OPTIONAL` or `MAY`). for the provider are not generally covered here. Note that the client also plays an equally important role in ensuring the security criteria is met. Even if the user agent and Rhino Accounts comply with every requirement, the client can compromise the security of the protocol through a faulty implementation.

10.1. Client Authentication

- *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.*
  Rhino Accounts distinguishes between confidential clients and non-confidential clients, and does not allow non-confidential clients to hold credentials nor does it allow them to perform confidential-only flows (such as the `Authorization Code` flow).

- *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...*

Rhino Accounts requires that *any* client, be it confidential or not, registers their redirection URIs. For every single type of flow, the redirection URIs are sanitized and checked byte for byte to make sure they match the registered URIs. For convenience, clients are allowed to specify multiple URIs, but partial URIs are never allowed.

10.2. Client Impersonation

- *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*
  Rhino Accounts requires that *any* client, be it confidential or not, registers their redirection URIs. When a client is confidential, Rhino Accounts always requires them to authenticate with their credentials over TLS. The authentication happens through an HTTP header (Basic Authentication), minimizing the chance that any credentials are ever logged on either the client or the server.

10.3. Access Tokens

- *Access token credentials (as well as any confidential access token attributes) MUST be kept confidential in transit and storage…*
  Rhino Accounts never writes the actual token value to a persistent store, but rather keeps a hash of the token that's processed through a secure one-way function. The entropy of the token and the hash guarantee that even if the persistent store is compromised, no known intruder could retrieve the token values within a reasonable amount of time.

- *Access token credentials MUST only be transmitted using TLS*
  Rhino Accounts requires that all communication with the server be through HTTPS. It also requires that all outgoing communication is conducted in HTTPS, with the sole exception of addresses whose top level domain is `localhost`. The reasoning for this is to allow client applications residing in the user's machine to conveniently handle tokens to provide a better user experience. Rhino Accounts is thus assuming that the user agent and the user's device is not compromised, for it if where, many other vectors of attack could be implemented.

- *The authorization server MUST ensure that access tokens cannot be generated, modified, or guessed to produce valid access tokens by unauthorized parties.*
  Rhino Accounts uses an industry standard random number generator to generate credentials with very high entropy.

10.4. Refresh Tokens

Refresh tokens are not currently supported by Rhino Accounts as of August 2015.

10.5. Authorization Codes

- *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.*
  Authorization codes have a very limited lifetime in Rhino Accounts. Currently, Rhino Accounts does not attempt to revoke previously issued access tokens. However, it has thread-safe mechanisms for ensuring that an authorization code is only redeemed at most a single time. Further attempts to redeem an authorization code will result in an error.

- *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.*
  Rhino Accounts forces confidential clients to authenticate when redeeming an authorization code. Rhino Accounts is also aware to whom the code originally issued, and thus enforces that it can only be redeemed by the client who requested it.

10.6. Authorization Code Redirection URI Manipulation

- *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.*
  Rhino Accounts keeps track of the redirection URI and enforces this constraint.

- *The authorization server MUST require public clients and SHOULD require confidential clients to register their redirection URIs.*
  Rhino Accounts requires all clients to register their redirection URIs.

10.7. Resource Owner Password Credentials

Due to security implications, Rhino Accounts does not implement this flow.

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.*
  Rhino Accounts requires that all communications to the server be performed with TLS. All the browsers that Rhino Accounts supports will warn the user of non-verifiable TLS connections. Authorization codes are always sent through HTTPS.

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.*
  Rhino Accounts requires that all communications to the server be performed with TLS.

10.10. Credentials-Guessing Attacks

- *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 probability for guessing any credential generated by Rhino Accounts (with the notable exception of user credentials), is well under the recommended 2^(-160) threshold. Rhino Accounts is designed with a centralized user credentials criteria that can be changed at any time to reduce the probability of guessing a user's password. Several other techniques for protecting user credentials have been taken as a precaution, but they are beyond the scope of this document.

10.11. Phishing Attacks

- *To reduce the risk of phishing attacks, the authorization servers MUST require the use of TLS on every endpoint used for end-user interaction.*
  As stated before, TLS is implemented and enforced system wide. The user portal is only accessible through HTTPS. Most user agents supported by Rhino Accounts have built-in phishing filters; this acts as a first line of defense for Rhino Accounts. Rhino Accounts also implements additional heuristics to inform and prevent the user from falling for a phishing attack.

10.12. Cross-Site Request Forgery

- *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.*
  Sophisticated CSRF protection is implemented by Rhino Accounts to prevent CSRF attacks that enforces every action in Rhino Accounts–whether it is logging in, out, or providing consent–to be performed explicitly by the user. There is one notable exception allowed by Rhino Accounts: Issuing a token to a registered client that already has a valid token whose scope is a superset of the scope they are requesting. This is done to prevent the user from consenting every single time for the same permissions in case the client does not (or cannot) store access tokens. Although there are security implications, I decided the pros outweighed the cons.

10.13. Clickjacking

To prevent this form of attack, native applications SHOULD use external browsers instead of embedding browsers within the application when requesting end-user authorization.
Rhino Accounts employs the use of the `x-frame-options` HTTP header that is understood by all browsers supported. This effectively prevents a browser from allowing a traditional clickjacking attack.

10.14. Code Injection and Input Validation

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.
Rhino Accounts performs basic sanitation for all values passed to it, and performs strict validation of any redirect_uri parameter passed.

10.15. Open Redirectors

Given that Rhino Accounts requires all redirection URIs to be registered, it should never be able to operate as an open redirector.

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

There are no requirements for providers here, although any client should carefully read this section to avoid a *Confused Deputy* attack.

RFC6819 security considerations implemented by Rhino Accounts

The following is a brief description of how Rhino Accounts is designed to mitigate or prevent all threats as described in Section 4 of RFC6819 by implementing the security considerations in section 5 of the same document. Even if the user agent and Rhino Accounts comply with every requirement, the client can compromise the security of the protocol through a faulty implementation.

5.1.1. Ensure Confidentiality of Requests

Rhino Accounts uses end-to-end TLS encryption for *any* network request between itself, the user agent, and registered clients.

5.1.2. Utilize Server Authentication

Rhino Accounts is hosted in a server accepting only secure connections. It only supports user agents that actively verify host certificates to prevent spoofing/illegal proxying/phishing.

5.1.3. Always Keep the Resource Owner Informed

Rhino Accounts will *always* ask the resource owner for consent or reject a request, except when all of these criteria are met:

1. The request already has a record of being granted and has not expired yet
2. The request's `redirect_uri` is not localhost
3. The maximum number of tokens for the particular request has not been exceeded.

5.1.4.1.1. Enforce Standard System Security Means
Rhino Accounts can be easily shut down by an administrator.

5.1.4.1.2. Enforce Standard SQL Injection Countermeasures
Rhino Accounts does not rely on SQL, and never uses concatenated input for queries (i.e. GQL).

5.1.4.1.3. No Cleartext Storage of Credentials
Credentials are stored with the highest available security practices, and the system is designed to be easily upgraded to a new standard should there be a need to.

5.1.4.1.4. Encryption of Credentials
Clients only.

5.1.4.1.5. Use of Asymmetric Cryptography
Asymmetric cryptography is used by Rhino Accounts for its OpenID Connect implementation, as well as for TLS.

5.1.4.2.1. Utilize Secure Password Policy
Rhino Accounts utilizes a basic entropy calculation function to decide whether a user's password meets the minimum requirements. It also aims to educate and encourage users to choose a strong password.

5.1.4.2.2. Use High Entropy for Secrets
Rhino Accounts uses a level of entropy that far exceeds the recommended requirements, and uses an industry-standard pRNG for generating secrets.

5.1.4.2.3. Lock Accounts
Rhino Accounts uses sophisticated calculus-based heuristics to lock accounts when a threat is detected.

5.1.4.2.4. Use Tar Pit
See 5.1.4.2.3**.**

5.1.4.2.5. Use CAPTCHAs
As stated in the document, CAPTCHAs are *not* used in Rhino Accounts because they add accessibility issues, and a determined attacker can learn (or already has learned) how to bypass them completely.

5.1.5.1. Limit Token Scope
Currently Rhino Accounts has no provisions for limiting a token's scope.

5.1.5.2. Determine Expiration Time
Rhino Accounts uses a short token expiration time by default to mitigate the effects of token leakage. However, for convenience, Rhino Accounts allows users to choose a specific token duration to prevent frequently having to give consent.

5.1.5.3. Use Short Expiration Time
See 5.1.5.2

5.1.5.4. Limit Number of Usages or One-Time Usage
Rhino Accounts *strongly* enforces that Authorization codes be redeemed only once. After the first attempt, any subsequent attempt will fail. Currently Rhino Accounts does not invalidate the first token that was successfully issued.

5.1.5.5. Bind Tokens to a Particular Resource Server (Audience)
Given the relatively small scope of Rhino Accounts, no such implementations are done, since the implementation wouldn't yield a substantial improvement in security but it would add considerable complexity.

5.1.5.6. Use Endpoint Address as Token Audience
Since `redirect_uri`s *must* be registered, this measure is not currently implemented by Rhino Accounts.

5.1.5.7. Use Explicitly Defined Scopes for Audience and Tokens
Rhino Accounts includes a provision for namespacing token scopes that can be used by authorization servers to mitigate attacks against multiple servers with a single token.

5.1.5.8. Bind Token to Client id
Tokens are issued to a specific client, and Rhino Accounts remembers this binding through the lifetime of the token.

5.1.5.9. Sign Self-Contained Tokens
#INCOMPLETE

5.1.5.10. Encrypt Token Content
#INCOMPLETE

5.1.5.11. Adopt a Standard Assertion Format
JWT is used for Rhino Accounts's OpenID Connect implementation.

5.1.6. Access Tokens

Rhino Accounts mandates that all access tokens sent to or from itself are encrypted using TLS.

5.2.1.1. Automatic Revocation of Derived Tokens If Abuse Is Detected
This is currently not implemented due to complexity.

5.2.2. Refresh Tokens
Refresh tokens are not currently supported by Rhino Accounts.

5.2.3.1. Don't Issue Secrets to Clients with Inappropriate Security Policy
Rhino Accounts does not prevent non-confidential clients from using a secret. The responsibility of keeping the secret safe is delegated to the client.

5.2.3.2. Require User Consent for Public Clients without Secret
If a client is impersonated, Rhino Accounts relies on TLS verification by the user agent to alert the user of fraud. There are two exceptions where this safeguard does not work: when the redirect uri is http://localhost, and when the attacker has managed to also steal or forge the TLS certificate of the client. To mitigate this risk, Rhino Accounts keeps a limit of tokens issued to a particular client for a particular user. When this limit is exceeded, the oldest token issued is silently revoked without warning.

5.2.3.3. Issue a "client_id" Only in Combination with "redirect_uri"
Rhino Accounts requires clients–public or confidential–to pre-register their redirect-uris.

5.2.3.4. Issue Installation-Specific Client Secrets
Rhino Accounts does not provide such provisions. However, an application could theoretically register multiple clients in Rhino Accounts with different secrets and achieve a similar result.

5.2.3.5. Validate Pre-Registered "redirect_uri"
Rhino Accounts mandates that redirect_uris specified match *exactly*, character for character.

5.2.3.6. Revoke Client Secrets
Rhino Accounts allows a client's secret to be modified at any time. Rhino Accounts also makes it possible to find and revoke all access tokens that were granted to a specific client.

5.2.3.7. Use Strong Client Authentication (e.g., client_assertion/client_token)
Rhino Accounts does not implement this measure currently due to complexity.

5.2.4.1. Automatic Processing of Repeated Authorizations Requires Client Validation
See 5.1.3

5.2.4.2. Informed Decisions Based on Transparency
Rhino Accounts aims to provide localized descriptions of the scope of a request to the user.

5.2.4.3. Validation of Client Properties by End User
Rhino Accounts includes the application’s name in the consent screen.

5.2.4.4. Binding of Authorization "code" to "client_id"
Rhino Accounts remembers which client was issued which authorization code through the lifetime of the code.

5.2.4.5. Binding of Authorization "code" to "redirect_uri"
Rhino Accounts remembers which redirect_uri was specified for an authorization code through the lifetime of the code.

5.3. Client App Security
There are no requirements for providers here, although any client should carefully read this section.

5.4. Resource Servers
There are no requirements for providers here, although resource servers carefully read this section.

5.5. A Word on User Interaction and User-Installed Apps
Rhino Accounts is built with the best intentions; But as this section proves, there will always be a way for a determined attacker to cause theft or damage; Rhino Accounts aims to mitigate these by implementing all reasonable security considerations.