Google Code

Account Authentication API

Authentication for Web Applications

Web applications that need to access Google services protected by a user's Google account can do so using the Authentication Proxy service. To maintain a high level of security, the proxy interface, called AuthSub, enables the web application to get access without ever handling their users' account login information. Before using, verify that the Google service to be accessed supports the Authentication service.

Note: If your application uses the JavaScript client library for the Google data API, you'll need to use the library's methods to access the AuthSub for JavaScript interface, as described here.

Developers can opt to handle authentication with either secure tokens or non-secure tokens. To use secure tokens, the web application must be registered with Google and have a certificate on file; if registered, the web application can secure all requests with a digital signature.

This page describes how to incorporate use of the Authentication service into your web-based application. See also the Google Accounts API Group for discussion on using the Accounts API.

Contents

Audience

This document is aimed at programmers who are developing web applications that access Google services and who want to implement a programmatic way to log into Google accounts. It assumes you've read up on the service(s) being accessed and are aware of any access/authentication issues involved. You need to know some service-specific details when incorporating use of the Authentication proxy service.

Example Scenario

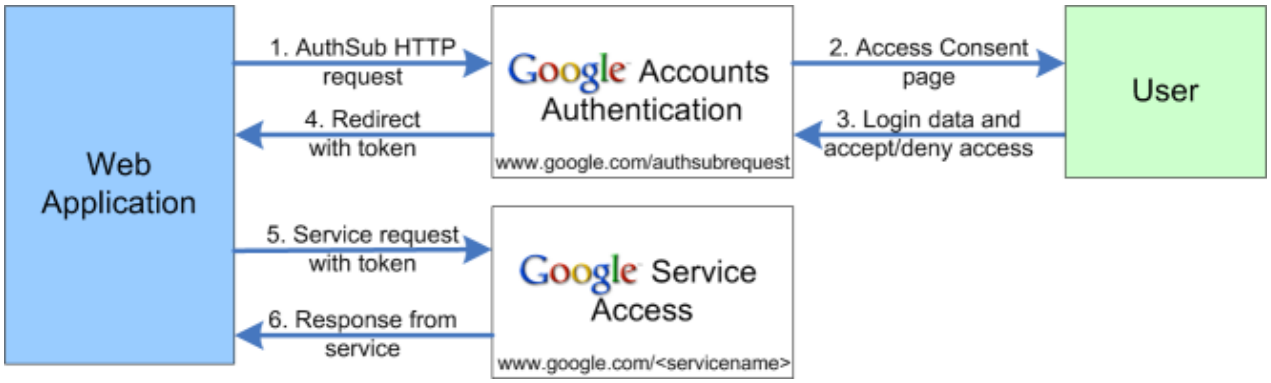
Users of Google Calendar manage their schedules, add, update, or delete events, and share calendar information. In this scenario, you want your application to access your users' Google Calendar data and manipulate that data on their behalf.

To do this, your application needs to get access to your user's Calendar account. Before accessing the Calendar service, the application must request authorization from Google. It does so by calling the Google

Authentication URL and supplying some required information, such as the name of the service to be accessed. A Google page is displayed that prompts the user to log in and either consent or refuse to provide access to their Google Calendar account. If they consent, Google redirects the user back to your web application and supplies an authentication token for the requested service. Your web application can then make requests directly to the Google service, referencing the token in each request.

The Authentication Process

Authentication for web applications involves a sequence of interactions between three entities: the web application, Google services, and the user. This diagram illustrates the sequence:



1. When the web application needs to access a user's Google service, it makes an AuthSub call to Google's Authentication Proxy service.
2. The Authentication service responds by serving up an "Access Consent" page. This Google-managed page prompts the user to log into their Google account and grant/deny access to their Google service.
3. The user logs into their Google account and decides whether to grant or deny access to the web application. If the user denies access, they are directed to a Google page rather than back to the web application.
4. If the user successfully logs in and grants access, the Authentication service redirects the user back to the web application. The redirect contains an authentication token good for one use; it can be exchanged for a long-lived token.
5. The web application contacts the Google service with a request, using the authentication token to act as an agent for the user.
6. If the Google service recognizes the token, it supplies the requested data.

Tokens and Token Management

Each authentication token is specific to one user and one service. If a web application needs access to more than one Google service for a user, it must request a separate token for each service. Each token describes the scope of the access allowed, which is determined by the Google service; some services choose to limit access to certain types of data or activity, such as read-only access.

You can opt to use one-time-use tokens or session tokens, depending how you decide to manage token usage in your application. A one-time-use token allows the application to make a single call to the Google service, while a session token lets the application make unlimited calls to the Google service. Session tokens do not expire. When using session tokens, your application should store the session token for each user rather than requesting a new one each time it needs access to a Google service. For other token management options, see Working With AuthSub.

Registered web applications can request a secure token in their AuthSub request. When calling a Google service with a secure token, the application must digitally sign the request, including both the token and the signature in the header of a request. The request can include a timestamp to enhance security, protecting against replay attacks. For more information about using secure tokens, see Signing Requests. Other than

these variations, the AuthSub request process is the same when using secure and non-secure tokens.

Some Google services may restrict access to web applications that are registered and using secure tokens.

Registration

If you're using AuthSub, you can opt to register your web application with Google. There are three levels of registration:

- Unregistered: Application is not recognized by Google. The Google Access Consent page, which prompts your users to either grant or deny access for your application, displays this caution highlighted in yellow: "Non-registered, not secure. (Warning Icon) This website has not registered with Google. We recommend that you continue the process only if you trust this destination."
- Registered: Application is recognized by Google. The Google Access Consent page, displays this caution: "Registered, not secure. (Warning Icon) This website is registered with Google to make authorization requests, but has not been configured to send requests securely. We recommend you continue if you trust this destination."
- Registered with enhanced security: Registered applications with a security certificate on file can use secure tokens. The Google Access Consent page displays this message: "Registered, secure. This website is registered with Google to make authorization requests."

For more information on the registration process, see Registration for Web Applications.

The AuthSub Interface

The AuthSub interface provides several methods for acquiring and managing authentication tokens. Once a web application has received a token, it can request access to a Google service. For information on forming the access request to a Google service, see the documentation for the service.

The following methods make up the AuthSub interface:

AuthSubRequest. A call to this method sends the user to a Google Accounts web page, where the user is given the opportunity to log in and grant Google account access to the web application. If successful, Google provides a single-use authentication token, which the web application can use to access the user's Google service data.

AuthSubSessionToken. A call to this method allows the web application to exchange a single-use token for a session token.

AuthSubRevokeToken. A call to this method revokes a session token. Once a token is revoked it is no longer valid.

AuthSubTokenInfo. A call to this method verifies whether a specified session token is valid and returns data associated with the token.

AuthSubRequest

Call AuthSubRequest to acquire single-use authentication. AuthSubRequest is called as a URL; make a request to `https://www.google.com/accounts/AuthSubRequest` with the following query parameters:

Parameter	Description
`next`	(required) URL the user should be redirected to after a successful login. This value should be a page on the web application site, and can include query parameters.
`scope`	(required) URL identifying the service to be accessed. The resulting token enables access to the specified service only. Some services may limit scope further, such as read-only access.

Sample Requests

The first example requests a single-use token for the Google Calendar service. The second example requests a secure token for the Google Calendar service that can be exchanged for a session token.

```
https://www.google.com/accounts/AuthSubRequest?next=http%3A%2F%2Fwww.yourwebapp
.com%2Fshowcalendar.html&scope=http%3A%2F%2Fwww.google.com%2Fcalendar%2Ffeeds
%2F&session=0&secure=0

~~~~~~~~~~~~~~~~

https://www.google.com/accounts/AuthSubRequest?next=http%3A%2F%2Fwww.yourwebapp
.com%2Fshowcalendar.html&scope=http%3A%2F%2Fwww.google.com%2Fcalendar%2Ffeeds
%2F&session=1&secure=1
```

AuthSubRequest Response

There are two potential responses to an AuthSub request:

- The request is rejected by Google. This response may occur if the request is malformed or if Google has reason to believe the requestor is not acting in good faith. It could also be rejected if the request's scope is not sufficiently narrow. If the request is rejected, the user is not redirected back to the requestor's web page; instead, a Google Accounts page is displayed indicating that the request failed.
- The request is accepted. In this case a Google Accounts "access request" page is displayed (see example). This page:
  - Prompts the user for their login information.
  - Lets the user know that a third party wants access to their Google service (identifies both the third party and the Google service)
  - Prompts the user to confirm that its OK to give the third party access to their Google service.

Note: The example shown illustrates the response to an AuthSub request from a non-registered web site. The response to a request from a registered application omits the yellow warning statement.

If the request is accepted but the user does not consent to the access request, they are redirected to a Google page (see example) instead of directly back to the web application page.

If the user successfully logs in and consents, Google redirects the user to the URL specified in the AuthSub request. The redirect URL contains the authentication token. All requests for access to a Google service must include a valid authentication token, specific to both the service and the user.

Sample Responses

This example illustrates the format of a Google redirect back to the web application with an authentication token. The redirect URL is the value of the AuthSub request *next* parameter. The token is a text string, up to 256 bytes.

```
http://www.yourwebapp.com/showcalendar.html?token=CKF50YzIHxCT85KMAg
```

The token returned is for a single use only; it lets you make only one request to the referenced Google service for that user. This limitation exists to secure authentication tokens specified in a URL. If you expect to make multiple calls to the Google service, you will want request a token that can be exchanged for a session token (in AuthSubRequest, set *session* to "true"). Then use the method AuthSubSessionToken.

AuthSubSessionToken

Call AuthSubSessionToken to exchange a single-use token for a long-lived session token. The single-use token is acquired by calling AuthSubRequest.

AuthSubSessionToken is a programmatic handler. Make an HTTP GET to the following URL: `https://www.google.com/accounts/AuthSubSessionToken`. Use an `Authorization` header with the following form:

```
Authorization: AuthSub token="token"
```

If the token is secure, it must be accompanied by a digital signature. See Signing Requests for instructions and examples.

Parameter	Description
`token`	(required) The authentication token received from Google in response to an AuthSubRequest call.

Sample Request

This example shows a request for a non-secure session token.

```
GET /accounts/AuthSubSessionToken HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: AuthSub token="GD32CMCL25aZ-v____8B"
User-Agent: Java/1.5.0_06
Host: https://www.google.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

AuthSubSessionToken Response

If the request for a session token is successful, Google responds with an HTTP 200 message with a set of key-value pairs in a "key=value" format. These values contain a session token and an expiration date. You can ignore the expiration date, which is not currently used; session tokens effectively do not expire.

Sample Responses

This example illustrates an AuthSub token returned in the response header.

```
Token=DQAA...7DCTN
Expiration=20061004T123456Z
```

AuthSubRevokeToken

Call AuthSubRevokeToken to revoke a valid session token. Session tokens have no expiration date and remain valid unless revoked.

AuthSubRevokeToken is a programmatic handler. To revoke a session token, make an HTTP GET to the following URL: `https://www.google.com/accounts/AuthSubRevokeToken`. Use an `Authorization` header with the following form:

```
Authorization: AuthSub token="token"
```

If the token is secure, it must be accompanied by a digital signature. See Signing Requests for instructions and examples.

Parameter	Description
token	(required) The session token, received in response to an AuthSubSessionToken request, to be revoked.

Sample Request

This example shows a revocation request for a non-secure session token.

```
GET /accounts/AuthSubRevokeToken HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: AuthSub token="GD32CMCL25aZ-v____8B"
User-Agent: Java/1.5.0_06
Host: www.google.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

AuthSubRevokeToken Response

If the request for session token revocation is successful, Google responds with an HTTP 200 message.

AuthSubTokenInfo

Call AuthSubTokenInfo to test whether a given session token is valid. This method validates the token in the same way that a Google service would; application developers can use this method to verify that their application is getting valid tokens and handling them appropriately without involving a call to the Google service. It can also be used to get information about the token, including next URL, scope, and secure status, as specified in the original token request.

This method can be used for both single-use and session tokens. Keep in mind, however, that if it is called with a single-use token, the call is treated as a valid use. Consequently, the AuthSubTokenInfo response indicates the token is valid, but the token is rendered invalid from that point on.

AuthSubTokenInfo is a programmatic handler. Make an HTTP GET to the following URL: `https://www.google.com/accounts/AuthSubTokenInfo`. Use an `Authorization` header with the following form:

```
Authorization: AuthSub token="token"
```

If the token is secure, it must be accompanied by a digital signature. See Signing Requests for instructions and examples.

Parameter	Description
token	(required) The authentication token received from Google in response to an AuthSub request.

Sample Request

This example shows a request for information on a non-secure token.

```
GET /accounts/AuthSubTokenInfo HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: AuthSub token="GD32CMCL25aZ-v____8B"
User-Agent: Java/1.5.0_06
Host: https://www.google.com
```

```
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

AuthSubTokenInfo Response

If the request is successful, Google responds with an HTTP 200 message with a set of key-value pairs in a "key=value" format. These values identify the target URL, scope, and secure status values, which were specified in the original token request. The target URL is simply the hostname gleaned from the next URL value.

Sample Responses

This example illustrates a response containing a session authentication token.

```
Target=http://www.yourwebapp.com
Scope=http://www.google.com/calendar/feeds/
Secure=true
```

Working With AuthSub

Incorporating AuthSub into your web application requires these tasks:

1. **Decide whether or not to register your web application.**

   Registered web applications have the advantage of being trusted by Google; the standard caveat displayed to users on the Google login page is omitted. In addition, registered web applications are identified with a descriptive name rather than merely the calling URL. Keep in mind that some Google services limit access to registered web applications only.

   If you choose to register, you need to generate a certificate and keys and go through the registration process. Registered web applications can acquire secure tokens for use when requesting data from a Google service, but can also use non-secure tokens if necessary.

2. **Decide what type of token to use and how to manage them.**

   An authentication token received from Google is intended to be used for all future interactions with the specified Google service for that user. The type of token you choose to use--single-use or session--depends on the type of interactions your web application will have with a Google service. For example, a single-use token may be sufficient if the interaction is a one-time or rare event.

   If you choose to get session tokens and use them regularly to access the Google service, your web application will need to manage token storage, including tracking the user and Google service the token is valid for. Google Accounts is not set up to manage large numbers of tokens, and in fact does not allow more than ten valid tokens (per user, per web application) to be outstanding at any one time. This limitation allows a web application to get multiple tokens to cover different services, if necessary; it does not support getting a new token each time the web application needs to access a Google service. If you decide to store session tokens, the tokens should be treated as securely as any other sensitive information stored on the server.

   Alternatively, the web application can get fresh tokens on a regular basis as long as a mechanism is set up to revoke a valid token before requesting a new one. Your application would need to revoke the existing token before requesting another. In this scenario, the user must log in and grant access each time a new token is requested.

3. **Determine the scope required by the Google service to be accessed.**

   Each Google service defines the scope covered by a token. The scope may be a simple URL identifying the service, or it may specify more restricted access. Some services severely limit access, such as allowing read-only access only. Refer to the documentation for the Google service you want to access to get the appropriate scope. You should request the most tightly scoped token possible. For example, if attempting to access Gmail's Atom feed feature, use the scope "http://www.google.com/calendar/feeds/",

not "http://www.google.com/calendar/". Google services are much more restrictive of access with large-scope requests.

4. **Set up a mechanism to request and receive an authentication token.**

   The mechanism must generate a well-formed request, including specifying the appropriate next URL and scope URL. It should also be equipped to parse the redirect received from Google, which contains the single-use token, and take action with it.

   The next URL can include query parameters helpful for processing the token. For example, when supporting multiple languages, include a query parameter that identifies the version of the web application the user is viewing. A *next* value of `http://www.yoursite.com/Retrievetoken?Lang=de` would result in the redirect `http://www.yoursite.com/Retrievetoken?Lang=de&token=DQAADKEDE`. Parsing the token and the Lang parameter ensures that the user is redirected back to the correct version of the site.

   Because authentication tokens are specific to a user, your application must be able to associate a token with its user. The preferred option is to issue a cookie to the user before making the token request. Then, when Google redirects the user back to your site with an appended token, your application can read the cookie and associate the token with the correct user identification.

5. **Set up mechanisms to request session tokens and store or revoke them, if relevant.**

   Depending on what decisions you made in step 2, you need to create mechanisms to form the HTTP requests. You can test both session and revocation mechanisms using AuthSubTokenInfo, which indicates whether a given token is valid or not. If storing tokens, the application must track both the user ID and the service (scope) covered by the token.

6. **Set up a mechanism to request access to a Google service.**

   Refer to documentation for the Google service for information on the proper request format. All requests to a Google service must include a valid authentication token. In general, a request to a Google service is in the form of an HTTP GET (or POST if writing new data), with the token referenced in the request header.

   The request header should take the following form:

```
Authorization: AuthSub token="token"
```

   where *token* is the authentication token, single-use or session, received from Google in response to an AuthSub request. If the token is secure, it must be accompanied by a digital signature. See "Signing Requests" for instructions and examples.

   The example below illustrates a request header for a call to the Google Calendar feed service. This request contains a non-secure token:

```
GET /calendar/feeds/default/private/full HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Authorization: AuthSub token="GD32CMCL25aZ-v____8B"
User-Agent: Java/1.5.0_06
Host: www.google.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Special Topics

This section covers some special topics related to Google Accounts authentication.

Signing Requests

All requests to Google services using a secure authentication token must be signed. This includes calls to AuthSubSessionToken, AuthSubRevokeToken, AuthSubTokenInfo, and all requests made to Google services. This section describes how to include a signature in a request.

As mentioned, secure tokens are issued only to web sites or applications that have registered with Google. Part of the registration process includes providing a certificate to Google. To sign requests, the web application generates a signature from the private key corresponding to the certificate. Google supports the RSA signature algorithm, and may support additional signature algorithms in the future. When a signed request is received, the Google service verifies the signature before granting access. Requests using a secure token are denied if they are not signed.

Note: Some Google services respond to a request with a redirect. In this case, the redirected request must also be signed.

A signature is added to the "Authorization" header of the request, along with the authentication token. The header should take the form:

```
Authorization: AuthSub token="token" sigalg="sigalg" data="data" sig="sig"
```

Parameter	Description
`token`	(required) The authentication token received from Google in response to an AuthSubRequest call.
`sigalg`	(required if token is secure) Signature algorithm. The only legal value for this parameter is "rsa-sha1", referring to SHA-1 with RSA using PKCS#1 padding. In the future, more signature algorithms may be supported.
`data`	(required if token is secure) The request metadata to be signed. The value of this parameter is a string with the following format: `data = http-method SP http-request-URL SP timestamp SP nonce` *http-method*: the HTTP method being invoked *SP*: a single ASCII space character *http-request-URL*: the full HTTP URL being requested *timestamp*: an integer representing the time the request was sent, expressed in number of seconds after January 1, 1970 00:00:00 GMT *nonce*: a random 64-bit, unsigned number encoded as an ASCII string in decimal A timestamp/nonce combination should never be used more than once.
`sig`	(required if token is secure) The signature for the secure token. This value must be a signature made by the private key corresponding to the certificate provided during registration. It must be encoded in BASE64, and must use the algorithm specified in the *sigalg* parameter.

Sample Request:

```
GET /accounts/AuthSubSessionToken HTTP/1.1
  Content-Type: application/x-www-form-urlencoded
  Authorization: AuthSub token="GD32CMCL25aZ-v____8B"
    data="GET https://www.google.com/accounts/AuthSubSessionToken
    1148503696 15948652339726849410" sig="MCwCFrV93K4agg==" sigalg="rsa-sha1"
  User-Agent: Java/1.5.0_06
```

```
Host: https://www.google.com
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Note: The Authorization line must be on a single line.

Single Sign-On

If your web application supports users with multiple Google services accessed through Google Accounts, you may be looking to get a single authentication token good for all of the user's Google services. This is called 'single sign-on''. Currently, Google AuthSub does not support a single-sign-on feature for third-party web applications. You need to get a separate token for each service, then store and manage all tokens. However, we know that our development community wants such a feature. We are evaluating options over the next few months to add this support.