Using "AuthSub" Authentication with the JavaScript Client Library

This document describes how to use the Google data API ("GData") JavaScript client library to connect to Google's "AuthSub for JavaScript" authentication system.

Note: If you're already familiar with AuthSub, Google's account authentication service for web-based applications, you'll see that AuthSub for JavaScript is conceptually very similar. The underlying implementation is different, but the differences aren't important to you as a client application developer. In some of the documentation, in contexts where the distinction is irrelevant, we refer to AuthSub for JavaScript as "AuthSub" for short.

The AuthSub for JavaScript interface allows a web-based application to access a Google service on behalf of a user. To maintain a high level of security, the interface enables the application to get an authentication token without ever handling the user's account login information.

The JavaScript client library provides methods to help you use AuthSub for JavaScript in your web application. In particular, there are methods for acquiring and revoking authentication tokens.

See also the Google Accounts API Group for discussion on using all the Authentication service APIs.

Contents

Audience

This document is aimed at programmers who are developing web applications, such as JavaScript mashups, that access Google services using the Google data APIs JavaScript client library.

This document assumes that you understand the general ideas behind the Google data APIs protocol. It also assumes that you know how to program in JavaScript. It does not assume that you're familiar with the AuthSub interface.

Supported environments

AuthSub for JavaScript is currently supported in Firefox 1.5 and higher and Internet Explorer 6.0 and higher.

Parts of this document refer to hypothetical situations in which your client connects to multiple services at once, but the only service that currently supports AuthSub for JavaScript is Google Calendar.

How AuthSub for JavaScript works

Here's a quick summary of how communication works between a web application, the Google Authentication service, and a Google data service:

1. To access a Google data service on a user's behalf, the web application must have a valid authentication token. To acquire a token, the web application makes an AuthSub for JavaScript login call to the Authentication service, specifying the service to be accessed.
2. On receiving the request from the web application, the Authentication service redirects the user to an "Access Request" page. This page prompts the user to log into their Google Account and to grant or deny access to their Google service.
3. The user 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 URL that made the original call. The redirect delivers an authentication token for the specified service. The client library stores the token as a cookie in the user's browser, under the web application's domain. The token is valid until revoked.
5. The web application contacts the Google data service. The client library sends the authentication token along with each request sent to the service.
6. If the Google data service recognizes the token, it supplies the requested data.

Using the AuthSub for JavaScript interface

The AuthSub for JavaScript library is included in the main GData JavaScript client library, so you don't need to do anything special to acquire the AuthSub for JavaScript library.

The following steps walk through the process of getting an authentication token and using it to access a Google service.

1. **Request an authentication token.**

   To request an authentication token, call the AuthSub for JavaScript method `google.accounts.user.login(scope)`. In most cases, this single step is all that's needed to acquire a token. This function automatically checks to see if a token is already stored in the browser; if one is found, it is returned. If no token is found, a request is sent to the authentication service.

   ```
   function doLogin(){
     scope = "http://www.google.com/calendar/feeds";
     var token = google.accounts.user.login(scope);
   }
   ```

   This function requires a scope value. Each Google service defines the scope of the access it allows, and you need to reference that scope in the token request. To determine what scope value to use, check the documentation for the Google service you want to access. The scope looks like a URL; it may be a simple URL identifying the service, or it may specify more restricted access, such as

limiting access to read-only. When the service offers a choice of scopes, request the most tightly scoped token possible. For example, to access Google Calendar's data feeds, use the scope "http://www.google.com/calendar/feeds", not "http://www.google.com/calendar".

Tips:

- We strongly recommend that you provide a login button or other user input mechanism to prompt the user to start the login process manually. If, instead, you call `google.accounts.user.login()` immediately after loading, without waiting for user interaction, then the first thing the user sees on arrival at your page is a Google login page. If the user decides not to log in, then Google does not direct them back to your page; so from the user's point of view, they tried to visit your page but were sent away and never sent back. This scenario may be confusing and frustrating to users.
- Applications that need to access more than one Google service for a user must request a new token for each new service (because each service has a different scope).

2. **Access the Google service normally.**

   The JavaScript client library automatically handles the tokens for you, attaching the relevant token to each request it sends to the service.

   On rare occasions, the Google service may reject the token. This occurs if the token becomes corrupt, or if a token stored in the client browser is otherwise invalid. For example, if the user visits their Google Accounts Authorized Websites page and revokes access, then the token is removed on the Google server but not from the cookies. To handle this exception, call `logout()`, which revokes the current authentication token, and then call `login(scope)` again to acquire a new, valid token.

3. **Revoke the authentication token.**

   The authentication token remains valid until you explicitly revoke it by calling `logout()`.

```
function doLogout(){
  google.accounts.user.logout();
}
```

Google recommends providing a manual logout feature, such as a logout button or a clickable link. That approach gives users the option to log out when they choose, or to stay logged in and keep their data feeds conveniently available for the next time they access your application.

AuthSub for JavaScript interface reference

The following methods make up the AuthSub for JavaScript interface, providing functionality to acquire and manage authentication tokens. All methods are called on the `google.accounts.user` object, which is defined in the JavaScript client library.

The `logout()` and `getInfo()` methods both rely implicitly on the "current token," which is the token associated with the scope that you specified at the last call to `login()` or `checkLogin()`. If your client is using multiple scopes, it can switch to a new scope by calling `checkLogin()` .

login(scope)

Call this method to get a valid authentication token. If an existing token is not found, a new token is requested from the authentication service, which causes the user's browser to redirect to a Google login page.

logout()

Call this method to revoke the current token and remove any related authentication cookies on the client browser.

checkLogin(scope)

Call this method to check whether or not an authentication token is stored in a cookie on the browser. This method does not redirect to a Google login page; it simply tells your client application whether there's a token available for that scope or not. You can use this method to indicate which scope to apply a subsequent `logout()` or `getInfo()` call to.

getInfo(callback)

Call this method to get detailed information associated with the current token. For debugging use.

login(scope)

Retrieves a valid authentication token. This method first checks whether or not a cookie for the specified scope is already stored in the browser. If one is found, the token value is returned. If no cookie is found, a request is sent to the authentication service, which shows the user a Google Accounts Request Access page. If the user successfully logs in and grants access, then cookies containing scope and token are stored in the client browser, and the token value is returned to the calling function.

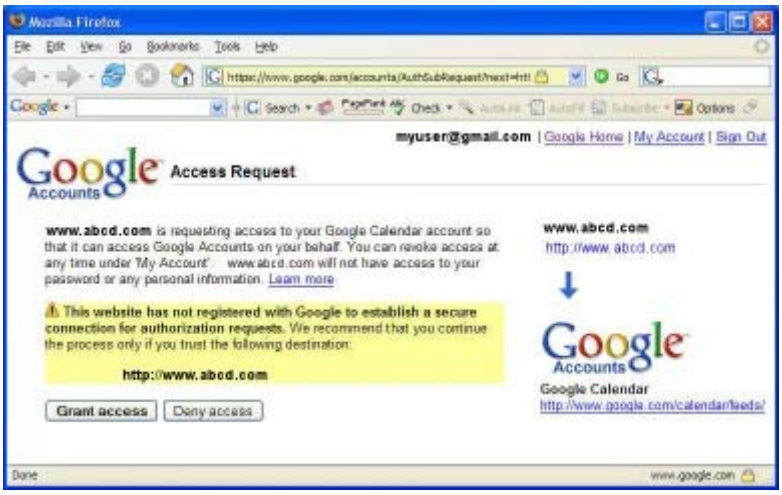
Parameter	Description
`scope`	(required) URL identifying the service to be accessed. The resulting token enables access to the specified service only. Depending on the service, some scopes are broad, while others are very restrictive. In some cases, the scope may specify read-only access.

Sample code:

```
function doLogin(){
  scope = "http://www.google.com/calendar/feeds";
  var token = google.accounts.user.login(scope);
}
```

Returns:

A token, keyed to a specific combination of user, client application domain, and Google service scope, is returned as a string.



If no valid token exists in the client browser, then the user is redirected to a Google page and invited to log into their Google account and authorize access by the web application. In this scenario, there are four possible outcomes:

- Google accepts the request. In this case a Google Accounts "access request" page is displayed (see screen shot #1). 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 it's okay to give the third party access to their Google data.

  If these steps are completed, a new token is issued and returned as a string and stored in the form of a cookie.

- Google rejects the request. This may occur if the request is malformed, or if Google has reason to believe the requestor is not acting in good faith. It may also occur if the request's scope is not sufficiently narrow. The user is redirected back to the application page that made the original call.
- If the request is accepted but the user does not have a Google account or does not consent to the access request, they are redirected to a Google page (see screen shot #2) instead of directly back to the web application page.
- Only a limited number of valid authentication tokens can exist for a user at any one time. If the user has too many active tokens, then the authentication system redirects them to an error page with a link to their Google Accounts Authorized Websites page. This page lists all the applications that are currently authorized; to make room for the new application's token, the user needs to manually revoke access for at least one currently authorized application.

logout()

Revokes the current authentication token and deletes all authentication cookies for that token stored by the browser. The Google service refuses subsequent requests for access until your client acquires a new token.

If your client application uses multiple scopes, then it should call `checkLogin()` to set the appropriate scope before calling `logout()`.

Sample code:

```
function doLogout() {
  scope = "http://www.google.com/calendar/feeds";
  if (google.accounts.user.checkLogin(scope)) {
    google.accounts.user.logout();
  }
}
```

Returns:

None.

checkLogin(scope)

Checks whether a cookie containing a token for the specified scope exists. Also, sets the current scope to the specified scope, so that subsequent calls to `logout()` and `getInfo()` use the new current scope.

Parameter	Description
`scope`	(required) URL identifying the service to be accessed.

Sample code:

```
function doCheck(){
  scope = "http://www.google.com/calendar/feeds";
  var token = google.accounts.user.check(scope);
}
```

Returns:

A token, if one exists. If no token exists for the specified scope, this method returns an empty string.

getInfo(callback)

Gets detailed information related to the current token, including target (the domain that originally requested the token) and scope. This method is primarily intended for debugging purposes, because this information is not needed to verify authentication. Unlike other AuthSub for JavaScript methods, this method uses a callback function to asynchronously receive the requested data.

If your client application uses multiple scopes, then it should call `checkLogin()` to set the appropriate scope before calling `getInfo()`.

Parameter	Description
`callback`	(required) Callback function to return information.

Sample code:

```
function handleInfo(data) {
  alert(data.scope);
}

function doGetInfo() {
  scope = "http://www.google.com/calendar/feeds";
  if (google.accounts.user.checkLogin(scope)) {
    google.accounts.user.getInfo(handleInfo);
  }
}
```

Returns:

Token details relating to the original token request, including:

- target: a URL value identifying the domain of the web page that made the original request
- scope: a URL value identifying the scope of the token
- secure: a Boolean value indicating whether the token is valid only for signed requests from SSL certificates registered with Google. Currently, this returned value is always false for tokens acquired using AuthSub for JavaScript.

About cookies and tokens

This section describes the cookies and tokens used by AuthSub for JavaScript. In most contexts, you won't need to know this information; the JavaScript client library handles all the details behind the scenes.

There are two cookies stored when an AuthSub for JavaScript login request is successful, each named with the prefix "g314":

- *g314<alphanumeric string>*: value is an encoded version of the token.
- *g314scope*: value is an alphanumeric string identifying the scope covered by the token. The URL is encoded using the default URL encoding specified in RFC 1738.

Each authentication token is specific to the following data:

- Google service scope
- user's Google account
- client application

The token data ensures that only the specified third-party application can request data and that the request is limited to data from the specified scope and user account.

Only one token for this combination of scope, user, and client can be valid at any one time. A web application must request a new token each time it needs access to a new Google service for a given user. The scope of access covered by the token depends on the Google service, which may choose to limit access to certain types of data or activity, such as read-only access.

The token returned by the AuthSub for JavaScript interface can be used as many times as is needed until it is revoked. It's up to your application to manage the life of the token, balancing security with convenience. Google recommends requesting a new token each time a new session is initiated.

Some Google services may allow access only by web applications that are registered and using secure tokens. AuthSub for JavaScript is not supported for such services. To use secure tokens, you'll need to use the standard AuthSub interface (and you thus can't use the JavaScript client library for this purpose), and your organization must register an SSL certificate with Google and sign all requests for those data feeds.