ColdFusion Adwords API Library

Documentation- 2012-03-10

The coldfusion adwords api library consists of a set of coldfusion components. Using the components you can access the data behind the google adwords api.

I assume that you are familiar with the basic concepts of google adwords and the adwords api. If not, I suggest reading http://code.google.com/apis/adwords/index.html and a few of the docs at http://code.google.com/apis/adwords/docs/ .

Feedback & Contact: m.orth@cfsolutions.de

Facebook Page:
http://www.facebook.com/pages/ColdFusion-Adwords-API-Client-Library/405372629479704

AdWords API web services

The AdWords API web services can be grouped by purpose into four categories:

- **Campaign Data Management**
- **Optimization**
- **Account Management**
- **Utility**

Each of the following web services provides a consistent set of operations: get retrieves entities based on a selector you pass in, and mutate lets you specify a series of operations (add, set, or remove) to perform. Web services that are designed only to provide data do not have a mutate operation, while some web services provide additional special-purpose calls.

Campaign Data Management

Use the campaign data management services to work with AdWords campaigns and their associated entities. Each campaign data management service corresponds to an entity in the campaign data hierarchy (documented in the Campaign Data Overview).

CampaignService (campaignservice.cfc)

Create, update, and delete campaigns. A campaign organizes one or more ad groups together and has its own budget, bidding strategy, serving date range, and targeting settings.

Coldfusion Examples: getAllCampaigns.cfm addCampaign.cfm

CampaignTargetService (campaigntargetservice.cfc)

Set targeting options (geographic regions, mobile carriers, search networks, etc.) to choose where and when a campaign's ads will appear.

AdGroupService (adgroupservice.cfc)

Create, update, and delete ad groups. An ad group organizes a set of ads and criteria together, and also provides the default bid for its criteria.

ColdFusion Examples: getAllAdGroups.cfm addAdGroup.cfm

AdGroupAdService (adgroupadservice.cfc)

Create, update, and delete ads.

ColdFusion Examples: getAllAdGruopAds.cfm addAdGroupAd.cfm

AdParamService (adparamservice.cfc)

Set ad parameters to quickly updates snippets of parameterized text in your text ads.

CampaignAdExtensionService (campaignadextensionservice.cfc)
and AdExtensionOverrideService (adextensionoverrideservice.cfc)

Create, update, and delete ad extensions. Campaign ad extensions enrich standard text ads by adding location information, additional links, or a phone number to all ads within a campaign. You can use ad extension overrides to specify a unique location on a per-ad basis.

CampaignCriterionService (campaigncriterionservice.cfc)
and AdGroupCriterionService (adgroupcriterionservice.cfc)

Create, update, and delete criteria. A criterion describes the conditions that determine whether an ad should appear.

ColdFusion Examples: getAllKeywords.cfm addKeyword.cfm

ConversionTrackerService (conversiontrackerservice.cfc)

Measure the effectiveness of your ads and keywords by learning what happens after a user clicks on your ad.

DataService (dataservice.cfc)

Retrieve ads campaign management data, based on specified criteria.

UserListService (userlistservice.cfc)

Create, update, and delete user lists. Use user lists and user list criteria to display ads to people who have previously triggered a conversion event on your website.

Optimization

Use the optimization services to retrieve performance statistics and discover ideas for new criteria.

BulkOpportunityService (bulkopportunityservice.cfc)

Download keyword, bid, and budget ideas that appear on the Opportunities Tab.

ReportDefinitionService (reportdefinitionservice.cfc)

Download a variety of performance reports.

TargetingIdeaService (targetingideaservice.cfc)

Generate new keyword and placement ideas based on parameters you specify.

ColdFusion Examples: getTargetingIdeas.cfm

TrafficEstimatorService (trafficestimatorservice.cfc)

Get traffic estimates for proposed campaigns, ad groups and keywords.

ColdFusion Examples: getTrafficEstimates.cfm

Account Management

Use the account management services to track your API unit usage and account activity.

AlertService (alertservice.cfc)

Retrieve all MCC alerts for a given account.

CustomerSyncService (customersyncservice.cfc)

Retrieve a record of campaign data changes within a specified date range.

ColdFusion Examples: getAllAccountChanges.cfm

InfoService (infoservice.cfc)

Get API usage information, such as the API units spent over a given date range.

ColdFusion Examples: getOperationCount.cfm

ServicedAccountService (servicedaccountservice.cfc)

Retrieve the hierarchy of accounts managed by the calling account.

ColdFusion Examples: getAccounts.cfm

Utility

Use these utility services as necessary to perform other tasks with the AdWords API.

BulkMutateJobService (bulkmutatejobservice.cfc)

Process a large batch of campaign data operations asynchronously. Bulk mutate jobs take longer to complete than synchronous calls to the standard web services, but the processed operations are half the cost.

GeoLocationService (geolocationservice.cfc)

Retrieve coordinates and the canonical address for a specified address.

ColdFusion Examples: getGeoLocationInfo.cfm

MediaService (mediaservice.cfc)

Upload and retrieve IDs for media you use in media-based ads (such as image or video ads).

Working with the library

So let’s start coding. We will work with the campaignservice to receive campaign data from the google adwords api. I start with a new instance of campaignservice.cfc.

```
<!--- create instance of the class campaignservice  --->
<cfset oCampaignService=createObject("component","campaignservice")>
```

Authentfication and Login

When sending a request to any of the AdWords API web services, you must provide the following data with your request:

authToken
The authToken dentifies either an MCC manager acting on behalf of a client, or an advertiser directly managing their own account. You retrieve this token by posting your account login and password to the Google ClientLogin API. An authentication token is valid for about a week after you acquire it; when it expires, use ClientLogin to retrieve a new one.

We will need a valid authToken for each request send to the adwords api. Use the getAuthTokenFromGoogle method from the component clientloginservice.cfc to login and getting back the authToken.

```
<!--- create instance of the class clientloginservice --->
<cfset oClientLoginService=createObject("component","clientloginservice")>

<cfset oClientLoginService.setEmail(adwords_api_email_account)>
<cfset oClientLoginService.setPasswd(adwords_api_password)>
<cfset authToken=oClientLoginService.getAuthTokenFromGoogle()>
```

The class adwordsuser.cfc stores all those data.

```
<!--- create instance of the class adwordsuser --->
<cfset oAdwordsUser=createObject("component","adwordsuser")>

<!--- setting authToken inside oAdwordsuser --->
<cfset oAdwordsUser.setAuthToken(authToken)>
```

There are also a few other settings that have to be send to the api for each request.

developerToken
A 22-character string that uniquely identifies an AdWords API developer. An example developer token string is ABcdeFGH93KL-NOPQ_STUv.

```
<cfset oAdwordsUser.setDeveloperToken(developer_token)>
```

userAgent
An arbitrary string that defines the sender and purpose of the request.

```
<cfset oAdwordsUser.setUseragent(adwords_api_useragent)>
```

clientCustomerId
When acting on behalf of a client, this header specifies that client's customer ID. Customer IDs are typically of the form 123-456-7890. Cannot be used if clientEmail is set.

```
<cfset oAdwordsUser.setClientCustomerId(customerId)>
```

Email address is no longer to be used as a client identifier; only Customer ID (available through InfoService or ServicedAccountService) is supported.

Sandbox

The AdWords API sandbox is an environment where requests can be made to the API without affecting ad serving, which can be useful when developing against new services or features. Additionally, no developer token is required to use the sandbox and no unit costs are incurred. Be aware that some services may behave differently in the sandbox than in production.

To access the sandbox, all you need is a Google account (if you don't have one, create a new account). You will use this account's email address and password to set the sandbox-specific request headers.

To create a sandbox account, send a get request to the sandbox version of CampaignService, using the WSDL location and sandbox headers as described below. This initial call to the sandbox creates an MCC sandbox account, along with five client accounts, for the email address you specified. Your sandbox account and its client accounts start out empty.

Sandbox WSDL Locations

To access the sandbox instead of the production environment, connect to the sandbox versions of the AdWords API web services. The sandbox WSDL URLs are identical to their production counterparts, except their domain is adwords-sandbox.google.com instead of adwords.google.com:

Production URL	https://adwords.google.com/api/adwords/cm/v201008/CampaignService
Sandbox URL	https://adwords-sandbox.google.com/api/adwords/cm/v201008/CampaignService

You don't have to know the URLs. **The coldfusion adwords api library will handle them automatically.**

Use the Sandbox

Simply set the value of method setUseSandbox() to true.

```
<!--- switch to sandbox mode --->
<cfset oCampaignService.setUseSandbox(true)>
```

Sandbox Authentification and Login

In addition to the automatic URL changes, you have to provide a different set of request header values inside of oAdwordsUser to work with the sandbox.

authToken (same as production environment)
Authenticates and authorizes the request to the sandbox.

developerToken
Identifies the sandbox account and its associated currency code. A string of the form login++currency_code, where login is your Google account's email address and currency_code identifies the currency code associated with this sandbox account. Your initial call to the sandbox sets the currency code for your sandbox account. Once you set the currency code, you cannot change it.

userAgent
An arbitrary string that identifies the intent of the request.

clientEmail (optional)
Specifies the client account. Since your sandbox account is an MCC account, you must specify a client account if you want to experiment with campaign data. Each client email address is of the form client_n+login, where n is one of {1, 2, 3, 4, 5} and login is your Google account's email address.

Logging

Call method setUseDefaultLogging(true) of your prefered service class eg. Campaignservice.cfc to turn on defaultLogging.

```
<!--- use defaultLogging and log request and response data --->
<cfset oCampaignService.setUseDefaultLogging(true)>
```

This will create a logs directory and a daily logfile inside the directory where the calling page is located. Eg. examples/logs/2011-05-19.txt

You can also pass in a custom logging config instead of using method setUseDefaultLogging(true):

```
<cfset stCustomLogSettings=structNew()>
<cfset stCustomLogSettings['bEnableLogging']=true>
<cfset stCustomLogSettings['bLogSOAPRequests']=true>
<cfset stCustomLogSettings['bLogSOAPResponse']=false>
<cfset stCustomLogSettings['folder']="./customLogDirectory">
<cfset stCustomLogSettings['extension']="log">

<!--- get a reference to the service object inside
 oCampaignService and call setLoggingSettings() --->
<cfset oService=oCampaignService.getService()>
<cfset oService.setLoggingSettings(stCustomLogSettings)>
```

Selector Object

With a selector you retrieve data using the get() method of most services. Eg. Selector to get a single campaign.

```
<!--- create instance of the class selector --->
<cfset oServiceSelector=createObject("component"," selector")>

<!--- setting the return fields --->
<cfset oServiceSelector.setFields("Id,Name,Status,StartDate,EndDate")>

<!--- create instance of the class predicate --->
<cfset oPredicate=createObject("component","predicate")>
<!--- setting field by which to filter the returned data --->
<cfset oPredicate.setField("Id")>
<!--- setting the operator to use for filtering the data returned --->
<cfset oPredicate.setOperator("EQUALS")>
<!--- setting the campaignID --->
<cfset oPredicate.setValues(val(form.campaignID))>

<!--- add predicate to the serviceSelector --->
<cfset oServiceSelector.setPredicates(oPredicate)>

<!--- get the campaignPage object through the google adwords api --->
<cfset oCampaignPage=oCampaignService.get(oServiceSelector)>
```

Understanding selector properties

The main properties of the selector are:

Fields
When using generic selectors to retrieve an object, the server will not return every field of the object by default; it will return only the requested set of fields. Other fields in the object will be set as undefined/null depending on the programming language you use. You can use the Fields property to specify the list of fields to be returned. You can use any Selectable field from the entity here. For example, when using CampaignService, you can use any Selectable field from Campaign. Campaign.name is marked as a Selectable field, and its selector field name is specified as “Name”, so you can use that as an entry for Fields.

Note that when an object has a field that is a reference to another object, then that property may not be marked as Selectable. In such cases, its sub fields will be marked as Selectable. For example, Campaign.budget is a property of type Budget, so Campaign.budget is not a Selectable field. To retrieve a campaign’s budget fields, you have to request Selectable fields from the Budget object.

Predicates
You can use predicates to filter the returned data by field values. You can use any Filterable field from the entity in a predicate. You can also use various operators to specify the filtering condition. Also, when using multiple predicates in a single selector, the predicates are ANDed to evaluate the effective filter condition.

Each service provides a set of service-specific fields and generic selector fields that can be used with Predicates. E.g. you can use CampaignId as a predicate field with AdGroupAdService, even though CampaignId is not a member of AdGroupAd. For the list of all the generic selector fields, you can refer to the selector migration guide.

DateRange
You can provide an optional DateRange to control the date range for which the data is retrieved. Specifying a date range only affects the stats returned by the server (if applicable), not the actual entities being returned by the server. The date format for min and max fields have the format yyyymmdd. The date range should be in the range [19700101, 20380101].

Ordering
You can use the ordering field to specify the fields on which you want to sort the data (ascending or descending). The order in which you specify the OrderBy objects in the ordering field is significant; the first element specifies the primary sort order, the second element specifies the secondary sort order and so on.

Paging
You can use the paging field to specify the position from which to start returning results and the number of results to return per page.

Proxy Configuration

Use method getService() of the adwords api webservice to get a reference to the service object (service.cfc). Create a structure with your proxy configuration and pass it to method setCFHTTPAttributes() of the service object reference.

```
<!--- all services use cfhttp to send data to the google adwords api --->
<!--- you can add proxy configuration settings to the service object --->
<!--- do not overwrite url and method attributes --->

<!--- create an attributesCollection for cfhttp --->
<cfset stCFHTTPAttributes=structNew()>
<!--- example proxy configuration --->
<cfset stCFHTTPAttributes.proxyServer="127.0.0.1">
<cfset stCFHTTPAttributes.proxyPort="8080">
<cfset stCFHTTPAttributes.proxyUser="username">
<cfset stCFHTTPAttributes.proxyPassword="password">

<!--- get a reference to the service object inside
 oCampaignService and call setCFHTTPAttributes() --->
<cfset oService=oCampaignService.getService()>

<cfset oService.setCFHTTPAttributes(stCFHTTPAttributes)>
```

Utils.cfc

Every class of the library extends utils.cfc with two nice methods.

method getObjectData()
Working with the library means working with objects and data. The method getObjectData() returns all data inside an object as a coldfusion structure or an array of structures. getObjectData(true) returns also the data of nested objects and arrays of objects.

```
<!--- get the returned campaign data as an array of structures --->
<cfset aEntries=oCampaignPage.getObjectData(true).entries>
```

method setObjectData()
The method setObjectData() dynamicaly sets all properties inside an object. Also nestings objects and arrays of objects will dynamicaly created for you.

Working with setObjectData():

```
<cfset stUser=structNew()>
<cfset stUser.authToken=authToken>
<cfset stUser.useragent=adwords_api_useragent>
<cfset stUser.developerToken=developer_token>
<cfset stUser.clientEmail=client_email>

<cfset oAdwordsUser=createObject("component","adwordsuser")>
<cfset oAdwordsUser.setObjectData(stUser)>
```

The normal way using setter methods:

```
<!--- create instance of the class adwordsuser and setting values --->
<cfset oAdwordsUser=createObject("component","adwordsuser")>

<cfset oAdwordsUser.setAuthToken(authToken)>
<cfset oAdwordsUser.setUseragent(adwords_api_useragent)>
<cfset oAdwordsUser.setDeveloperToken(developer_token)>
<cfset oAdwordsUser.setClientEmail(client_email)>
```

API Errors

As with any programming system, error handling is a critical part of the AdWords API. The apis error handling system gives you the type of error (e.g. PolicyViolationError), the cause of error (e.g. the editorial policy that was violated), the field that triggered the error and so forth. When an error occurs the api raised an exception of the type apierror. The last error message is also saved inside the service object. Use cftry witch cfcatch to handle those errors.

```
<cftry>

<!--- get the campaignPage object through the google adwords api --->
<cfset oCampaignPage=oCampaignService.get(oServiceSelector)>

<!--- Is it an error raised through the api usage? --->
<cfcatch type="adwordsapi">

<!--- getError() returns the error message as a coldfusion structure --->
<cfdump var="#oCampaignService.getError()#" label="API Error Message">

</cfcatch>

<!--- Okay, it's a general error. --->
<cfcatch type="any">

<cfdump var="#cfcatch#" label="general cfcatch">

</cfcatch>

</cftry>
```