

Présentation de l'API Google Analytics en PHP

par Jean-François Lépine ([Halleck](#))

Date de publication : Avril 2009

Dernière mise à jour :

Il existe désormais un moyen officiel pour accéder à toutes vos données Google Analytics. Je vous propose ici de vous présenter brièvement comment fonctionne cet outil : l'API Google Analytics

I - Présentation.....	3
II - Réception des données : le principe.....	3
II-A - Principe général.....	3
II-B - Authentification.....	3
II-C - Réception de données : cURL.....	4
II-D - Réception de données : Stream.....	5
II-E - Exemple : lister tous les profils du compte.....	5
III - Pour aller plus loin.....	6
III-A - Un jeton à usage multiple.....	6
III-B - Un flux XML c'est bien, des données organisées c'est mieux.....	6
III-C - Exemple : lister tous les profils du compte.....	7
IV - Plus simple et plus facile avec le Zend Framework.....	8
IV-A - Présentation.....	8
IV-B - Authentification.....	8
IV-C - Réception de données.....	8
IV-D - Exemple : lister tous les profils du compte.....	8
V - Pour ceux qui ne peuvent pas utiliser le Zend Framework : alternative POO.....	9
V-A - Présentation.....	9
V-B - Exemple : lister tous les profils du compte.....	12
V-C - Afficher le nombre de pages vues.....	12
VI - Conclusion.....	13
VI-A - Conclusion.....	13
VI-B - Ressources et fichiers.....	13

I - Présentation

Il existe désormais (depuis Avril 2009) un moyen officiel pour accéder à toutes vos données Google Analytics : l'**API Google Analytics**.

Cette API vous permet d'accéder à l'ensemble de vos statistiques, pour peu que vous ayez les droits d'accès sur les informations demandées. Cet accès est sécurisé et rapide. Seule limite : vous n'aurez droit "qu'à" 10 000 requêtes par jour, 100 par dizaine de seconde, et la pagination des résultats est "limitée" à 1000. Bref, c'est largement suffisant dans la plupart des cas...

II - Réception des données : le principe

II-A - Principe général

L'API Google Analytics est relativement simple : vous demandez une information en GET (en accédant à une URL), vous recevez un flux XML en réponse.

II-B - Authentification

L'échange des données se fait après une authentification préalable par Google. En réponse de l'authentification, vous recevez un jeton d'identification "token", **valable pour une seule requête** (nous verrons comment le rendre permanent) :

Il existe deux types d'authentification :

Deux types d'authentification

- **Authentification client** :
L'utilisateur est redirigé vers Google Account et doit saisir son mot de passe. le compte utilisé est celui de l'utilisateur.
- **Authentification serveur** :
L'identifiant et le mot de passe sont enregistrés dans l'application, et transmis de façon transparente pour l'utilisateur. Le compte utilisé est celui du propriétaire de l'application.

Authentification client	Authentification serveur
Redirection vers Google Account	Identification directe par compte unique et obtention d'un jeton
L'utilisateur accepte (ou non) que l'application ait accès à ses données	Utilisation par l'application de ce jeton pour effectuer une requête
Redirection vers l'application et livraison d'un jeton	
Utilisation par l'application de ce jeton pour effectuer une requête	

Nous allons dans un premier temps n'examiner que le premier cas d'authentification (Authentification client). Pour un exemple d'Authentification serveur, vous pouvez vous rendre dans la section IV de ce tutoriel (**Zend Framework**).

Voyons maintenant ce qui se passe lors d'une Authentification client. Concrètement, il suffit d'appeler la page <https://www.google.com/accounts/AuthSubRequest>, en précisant deux paramètres :

Authentification client - Paramètres

- scope : le service (ici pour Google Analytics : <https://www.google.com/analytics/feeds/>)
- next : la page de redirection une fois l'authentification effectuée

```

/**
 * appeler la page de connexion
 * l'utilisateur sera redirigé vers le site Google, puis redirigé vers la page indiquée
 * @param string url de retour
 */
function google_login($urlRetour) {
    $url = 'https://www.google.com/accounts/AuthSubRequest?'
        . '&next='.$urlRetour
        . '&scope=https://www.google.com/analytics/feeds/'
        . '&secure=0'
        . '&session=1';

    header("Location: $url");
    exit;
}
    
```

II-C - Réception de données : cURL

Une fois authentifié, Google redirige l'utilisateur vers votre page (url passée dans "next"), et ajoute un argument ? token=xxxxxxx. C'est votre jeton d'identification, valable une fois.

```

/**
 * récupérer le token de l'url en GET
 * @return string token
 */
function google_get_token() {
    if( !isset($_GET['token']) ) return false;
    if( empty($_GET['token']) ) return false;
    return strip_tags( $_GET['token'] );
}
    
```

Vous aurez besoin de ce jeton à chaque requête envoyée à Google. Ce jeton s'envoie en en-tête de votre requête :

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

C'est ici que ça se corse un peu. Pour envoyer cet en-tête, vous aurez besoin de cURL :

```

/**
 * effectuer une requete cUrl chez google
 * @param string requete
 * @return xml / génère une erreur
 */
function google_call($url, $token) {

    //
    // entête à envoyer
    $headers = array( 'Authorization: AuthSub token="'.$token.'" );

    //
    // Requete cUrl
    $ch = curl_init($url);
    $options = array(
        CURLOPT_RETURNTRANSFER => true, // renvoie une page web
        CURLOPT_HEADER => false, // ne renvoie pas d'en-tête
        CURLOPT_HTTPHEADER => $headers, // en-têtes à envoyer
        CURLOPT_FOLLOWLOCATION => true, // suivre les redirections
        CURLOPT_ENCODING => "", // tout encodage possible
        CURLOPT_USERAGENT => "spider", // nom de l'agent
        CURLOPT_AUTOREFERER => true, // set referer on redirect
        CURLOPT_CONNECTTIMEOUT => 120, // timeout on connect
        CURLOPT_TIMEOUT => 120, // timeout on response
        CURLOPT_MAXREDIRS => 10, // stop after 10 redirects
    );
    
```

```

CURLOPT_SSL_VERIFYHOST => 0, // ne pas vérifier le SSL
CURLOPT_SSL_VERIFYPEER => false, // ne pas vérifier le SSL
CURLOPT_VERBOSE => 1
);
curl_setopt_array($ch,$options);

//
// Reponse cUrl
$content = curl_exec($ch);
$error = curl_errno($ch);
$errormsg = curl_error($ch);
$response = curl_getinfo($ch);
curl_close($ch);

//
// Erreur : le code 200 n'est pas envoyé
if($response['http_code'] != '200') {
    echo "<strong>Impossible d'effectuer la requete : $content</strong><p><pre>";
    print_r($response);
    echo '</pre></p>';
    exit;
}

//
// renvoyer le contenu
return $content;
}
    
```

Si tout est bon, Google renvoie le code "200".

II-D - Réception de données : Stream

Vous pouvez tout autant passer par des fonctions de flux (stream), et éviter ainsi l'utilisation de cURL :

```

/**
 * effectuer une requete chez google en utilisant les fonctions de flux
 * @param string requete
 * @return xml
 */
function google_call($url, $token) {

    //
    // entête à envoyer
    $headers = array( 'Authorization: AuthSub token="$token" ' );
    $options = array(
        'http' => array(
            'method' => 'POST',
            'header' => $this->headers)
    );

    //
    // flux
    $context = stream_context_create($options);
    $content = file_get_contents($url, false, $context);

    return $content;
}
    
```

II-E - Exemple : lister tous les profils du compte

Maintenant, utilisons cette fonction pour effectuer la requête qui va nous fournir la liste de nos comptes :

```

//
// effectuer une requete : on va lister tous les comptes du profil
    
```

```
$accountxml = google_call("https://www.google.com/analytics/feeds/accounts/default");
echo $accountxml;
```

Vous voilà avec un flux XML contenant la liste de vos comptes.

III - Pour aller plus loin

III-A - Un jeton à usage multiple

Le jeton que vous avez obtenu n'est valable qu'une fois, c'est-à-dire qu'à chaque requête, Google va vous demander de vous reconnecter (si vous essayez sans, vous recevrez à la place du flux une erreur 301 "Invalid Token" dans le corps de la réponse).

Il va falloir rendre ce jeton à usage unique permanent (nous parlerons de jeton à usage multiple). Pour cela, effectuer une requête à l'adresse "http://www.google.com/accounts/AuthSubSessionToken", en passant notre jeton actuel en en-tête. Nous recevons une réponse du type : Token=xxxxxxx

où xxxxxx est le jeton à usage multiple que vous devrez désormais utiliser :

```
/**
 * transformer un jeton à usage unique en un jeton à usage multiple
 * @param string jeton à usage unique
 * @return string jet à usage multiple
 */
function google_get_permanent_token($token) {

    $url = 'http://www.google.com/accounts/AuthSubSessionToken';
    $new_token = google_call($url, $token);

    //
    // extraire le token du résultat
    $new_token = str_ireplace('Token=', '', $new_token);
    return $new_token;
}
```

III-B - Un flux XML c'est bien, des données organisées c'est mieux

Vous me direz : j'ai eu un flux XML, super, mais qu'est-ce que j'en fait. Je n'arrive même pas à le parser correctement avec SimpleXML ?

Pas de souci, je vous propose une petite fonction bien pratique, que j'ai trouvée sur [Alex Curelea's Dev Log](#)

```
/**
 * parser les comptes utilisateur
 * trouvé sur Alex Curelea's Dev Log
 * http://www.alexc.me/using-the-google-analytics-api-getting-total-number-
 * of-page-views/74/
 * @param string xml
 * @return array
 */
function google_parse_account_list($xml)
{
    $doc = new DOMDocument();
    $doc->loadXML($xml);
    $entries = $doc->getElementsByTagName('entry');
    $i = 0;
    $profiles = array();
}
```

```
foreach($entries as $entry)
{
    $profiles[$i] = array();

    $title = $entry->getElementsByTagName('title');
    $profiles[$i]["title"] = $title->item(0)->nodeValue;

    $entryid = $entry->getElementsByTagName('id');
    $profiles[$i]["entryid"] = $entryid->item(0)->nodeValue;

    $properties = $entry->getElementsByTagName('property');
    foreach($properties as $property)
    {
        if (strcmp($property->getAttribute('name'), 'ga:accountId') == 0)
            $profiles[$i]["accountId"] = $property->getAttribute('value');

        if (strcmp($property->getAttribute('name'), 'ga:accountName') == 0)
            $profiles[$i]["accountName"] = $property->getAttribute('value');

        if (strcmp($property->getAttribute('name'), 'ga:profileId') == 0)
            $profiles[$i]["profileId"] = $property->getAttribute('value');

        if (strcmp($property->getAttribute('name'), 'ga:webPropertyId') == 0)
            $profiles[$i]["webPropertyId"] = $property->getAttribute('value');
    }

    $tableId = $entry->getElementsByTagName('tableId');
    $profiles[$i]["tableId"] = $tableId->item(0)->nodeValue;

    $i++;
}
return $profiles;
}
```

III-C - Exemple : lister tous les profils du compte

Nous avons donc désormais une page complète, qui va nous permettre de lister les comptes liés à notre profil.

```
//
// appel du fichier qui contient toutes les fonctions que nous avons créées.
// Vous pouvez télécharger ce fichier dans la partie "Ressources & Fichiers" de ce tutoriel
require_once 'inc/google_functions.php';

//
// configuration
$profil_id = '123456789';
$urlRetour = 'www.monsite.fr/
mapage.php'; // url de la page vers laquelle sera redirigé l'utilisateur une fois connecté
//
// Récupérer le token
$single_use_token = google_get_token();

//
// Etablir la connexion à Google Analytics si pas encore de token
if(!$single_use_token) {
    google_login( $urlRetour );
}

//
// obtenir un token à usage multiple
$token = google_get_permanent_token( $single_use_token );

//
// effectuer une requete : on va lister tous les comptes du profil
$accountxml = google_call("https://www.google.com/analytics/feeds/accounts/default");

//
```

```
// parser cette requete et afficher les résultats
$tProfiles = google_parse_account_list($accountxml);
echo '<pre>';
print_r( $tProfiles );
```

IV - Plus simple et plus facile avec le Zend Framework

IV-A - Présentation

Je vous propose maintenant de découvrir une solution plus simple d'utilisation, mais qui nécessite l'utilisation du Zend Framework. Cette solution repose sur le **package GData**, que vous pourrez télécharger [ici](#).

L'avantage de cette solution est qu'elle est stable et compatible avec tous les Web Services (API) de Google. L'inconvénient (ca peut en être un sur de petits projets qui n'en n'ont pas un réel besoin) c'est qu'elle requiert un nombre important de fichiers sur le serveur et une configuration qui supporte le ZF.

Pour la suite de cette section, je présumerai que vous avez installé sur votre serveur le package GData du Zend Framework.

IV-B - Authentification

L'authentification est on ne peut plus simple :

```
//
// Authentification
$client = Zend_Gdata_ClientLogin::getHttpClient('mon.email@email.fr', 'mon.mot.de.passe', 'analytics');
$token = $client->getClientLoginToken();
```

Vous voilà déjà avec un jeton à usage multiple !

Vous remarquerez qu'ici on a passé directement les informations du compte du propriétaire de l'application. Il s'agit désormais d'une Authentification serveur (rappelez-vous que, jusqu'ici, nous n'utilisons que des Authentification client : les données étaient issues du compte du visiteur et non pas d'un cmpte unique commun à tous les visiteurs).

IV-C - Réception de données

Comme d'habitude, on récupère les données en appelant l'URL concernée, en envoyant en en-tête le jeton, qui nous renvoie un flux XML :

```
//
// requete vers le web service
$client = new Zend_Http_Client($url);
$client->setHeaders( "Authorization: GoogleLogin auth=".self::get('token') );
$r = $client->request(Zend_Http_Client::GET);
//
// on reçoit en réponse un flux XML : $xmlBody :
$xmlBody = $r->getBody();
//
// nettoyer les balises du flux
$xmlBody = str_replace('dxp:', '', $xmlBody);
```

IV-D - Exemple : lister tous les profils du compte

Pour lister les profils, nous utiliserons SimpleXML (vous devez donc l'avoir activé sur votre serveur) :

```

$url = "https://www.google.com/analytics/feeds/accounts/default";
$client = new Zend_Http_Client($url);
$client->setHeaders( "Authorization: GoogleLogin auth=".self::get('token') );
$r = $client->request(Zend_Http_Client::GET);
$xmlBody = $r->getBody();
$xmlBody = str_replace('dxp:', '', $xmlBody);

//
// utilisation de SimpleXML
$xml = simplexml_load_string($xmlBody);
$profiles = array(); // tableau qui va recevoir tous les profils

//
// pour chaque résultat
foreach($xml->entry as $entry) {

    //
    // tableau qui va contenir temporairement les infos d'un seul profil
    $val = array();

    //
    // récupérer les infos principales
    $val['title'] = strval( $entry->title );
    $val['id'] = strval( $entry->id );
    $val['tableId'] = strval( str_replace('ga:', '', $entry->tableId) );

    //
    // récupérer les propriétés, qui sont stockées sous forme d'attribut, dans des noeuds "property"
    foreach($entry->property as $r) {
        $name = strval($r['name']);
        $name = trim( str_replace('ga:', '', $name) );
        $val[$name] = strval($r['value']);
    }

    //
    // ajouter au tableau final
    $profiles[] = $val;

}

//
// afficher
print_r($profiles);
    
```

V - Pour ceux qui ne peuvent pas utiliser le Zend Framework : alternative POO

V-A - Présentation

Cette partie est entièrement facultative pour la compréhension de l'API. Si vous avez la possibilité d'utiliser Zend Framework, vous pouvez directement passer à la conclusion :-)

Pour ceux qui sont intéressés, je vous propose de une petite classe, qui nous permettra de gérer l'API Google Analytics sans le Zend Framework, ce de manière plus efficace qu'en utilisant de simples fonctions. Nous appellerons cette classe Google_Analytics, et elle sera située dans le fichier class/Google_Analytics.php

Pour plus de simplicité, j'ai utilisé des raccourcis, notamment l'utilisation d'un tableau pour stocker l'ensemble des informations de la classe, ce qui n'est pas forcément à faire en production.

Cette classe reprend l'ensemble des fonctions que nous avons vu plus haut, je vous la présente donc globalement sans m'arrêter dans le détail.

L'authentification est gérée par la méthode authenticate(), qui regarde si un jeton permanent existe, et effectue les opérations d'identification si besoin (appel des méthodes login() et setPermanentToken()).

La fonction `google_call` est remplacée par la méthode `call()`.

Enfin, pour éviter d'avoir à redemander à chaque nouvelle page un nouveau token, ce token est conservé en session.

```

abstract class Google_Analytics {

    private static $_var = array(); // espace de stockage pour les variables.

    /**
     * authentifier l'utilisateur
     * la méthode va chercher si il y a un token défini
     * sinon, elle va tenter d'en déterminer un :
     * soit il y a un token dans la session
     * soit il n'y a aucun token => redirection vers la page de login de Google
     * soit il y a déjà un token à usage unique => transformation du token en token à usage multiple
     */
    public static function authentificate() {

        if( !self::get('token')) {

            //
            // rechercher dans session
            if( isset($_SESSION['token']) ) {
                self::set('token', $_SESSION['token']);
                return;
            }

            //
            // premier affichage :
            if( !self::get('single_use_token') ) {

                //
                // pas de token en GET, le demander
                if( !isset($_GET['token']) ) {
                    self::login( self::get('base_url') );
                } else {
                    //
                    // token en GET, le rendre permanent
                    self::set( 'single_use_token', strip_tags($_GET['token']) );
                    self::setPermanentToken();
                }
            }
        }

        /**
         * appeler la page de connexion
         * l'utilisateur sera redirigé vers le site Google, puis redirigé vers la page indiquée
         */
        public static function login() {
            $url = 'https://www.google.com/accounts/AuthSubRequest?'
                . '&next='.$_urlRetour
                . '&scope=https://www.google.com/analytics/feeds/'
                . '&secure=0'
                . '&session=1';

            header("Location: $url");
            exit;
        }

        /**
         * transformer un jeton à usage unique en un jeton à usage multiple
         */
        public static function setPermanentToken() {

            $url = 'http://www.google.com/accounts/AuthSubSessionToken';
        }
    }
}

```

```

$token = self::call($url);

//
// extraire le token
$token = str_ireplace('Token=', '', $token);
self::set('token', $token);

//
// stocker en session
if(isset($_SESSION)) {
    $_SESSION['token'] = $token;
}

}

/**
 * effectuer une requete cUrl chez google
 * @param string requete
 * @return xml / génère une erreur
 */
public static function call($url) {

    //
    // choix du token à utiliser
    $token = self::get('token');
    if(!$token) $token = self::get('single_use_token');

    //
    // entête à envoyer
    $headers = array(
        sprintf("Authorization: AuthSub token=\"%s\"/n", $token ),
        //sprintf("X-Google-Key: key=\"%s\"/n", self::get('key') ) // au cas où, utilisez ici votre developper key Google
    );

    //
    // Requete cUrl
    $ch = curl_init($url);
    $options = array(
        CURLOPT_RETURNTRANSFER => true, // renvoie une page web
        CURLOPT_HEADER => false, // ne renvoie pas d'en-tête
        CURLOPT_HTTPHEADER => $headers, // en-têtes à envoyer
        CURLOPT_FOLLOWLOCATION => true, // suivre les redirections
        CURLOPT_ENCODING => "", // tout encodage possible
        CURLOPT_USERAGENT => "demo-dvp", // nom de l'agent
        CURLOPT_AUTOREFERER => true, // set referer on redirect
        CURLOPT_CONNECTTIMEOUT => 120, // timeout on connect
        CURLOPT_TIMEOUT => 120, // timeout on response
        CURLOPT_MAXREDIRS => 10, // stop after 10 redirects
        CURLOPT_SSL_VERIFYHOST => 0, // ne pas vérifier le SSL
        CURLOPT_SSL_VERIFYPEER => false, // ne pas vérifier le SSL
        CURLOPT_VERBOSE => 1
    );
    curl_setopt_array($ch,$options);

    //
    // Reponse cUrl
    $content = curl_exec($ch);
    $err = curl_errno($ch);
    $errmsg = curl_error($ch);
    $resp = curl_getinfo($ch);
    curl_close($ch);

    //
    // Erreur : le code 200 n'est pas envoyé
    if($resp['http_code'] != '200') {
        echo '<strong>Impossible d\'effectuer la requete : '.$content.'</strong><p><pre>';
        print_r($resp);
        print_r($headers);
        echo '</pre></p>';
        exit;
    }
}

```

```

    }

    //
    // renvoyer le contenu
    return $content;
}

/**
 * getters and setters
 */
public function set($var, $value) { self::$_var[$var] = $value; }
public function get($var) { if(!isset(self::$_var[$var])) return false; return self::$_var[$var]; }
}
    
```

Ce code est simple, je vous laisse le consulter en vous référant aux remarques précédentes.

Cette classe est loin d'être complète, mais j'espère qu'elle sera vous aider à mieux comprendre comment utiliser l'API Google Analytics.

V-B - Exemple : lister tous les profils du compte

Nous pouvons maintenant terminer notre application, qui sera cette fois totalement orientée Objet :

```

session_start();

require_once 'class/Google_Analytics.php';
require_once 'inc/google_functions.php';

//
// configuration
Google_Analytics::set('profil_id' , '123456' );
Google_Analytics::set('base_url' , 'http://monsite.fr/mapage.php' );

//
// Etablir la connexion à Google Analytics
Google_Analytics::authenticate();

//
// lister les comptes
$accountxml = Google_Analytics::call("https://www.google.com/analytics/feeds/accounts/default");
$tProfiles = parse_account_list($accountxml);
echo '<pre>';
print_r($tProfiles);
    
```

V-C - Afficher le nombre de pages vues

Il est possible d'accéder à une large gamme de données (**dont voici la liste complète**).

Il suffit d'effectuer une requête à l'adresse <https://www.google.com/analytics/feeds/data>, en précisant :

- ids : l'id du profil
- metrics : les données demandées

Il y a **d'autres paramètres**, mais ceux-là suffisent à notre exemple.

```

//
// profil id du premier compte trouvé:
$profilId = $tProfiles[0]["tableId"];
    
```

```
//  
// requete : nombre de pages vues (ga:pageviews) du 01 avril 2009 à aujourd'hui  
$requrl = sprintf("https://www.google.com/analytics/feeds/data?ids=%s&metrics=ga:pageviews&start-date=2009-04-01&end-date=".date('Y-m-d'), );  
$pagecountxml = Google_Analytics::call($requrl);  
  
//  
// analyse du résultat  
$doc = new DOMDocument();  
$doc->loadXML($pagecountxml);  
$metrics = $doc->getElementsByTagName("metric");  
$views = $metrics->item(0)->getAttribute('value');  
  
echo 'il y a '.$views.' pages vues pour la période';
```

VI - Conclusion

VI-A - Conclusion

L'API a été longtemps attendue et répond à nos attentes : grâce au package GData du Zend Framework, il est très facile de l'utiliser, et il est possible d'accéder à une **très large gamme de données**.

VI-B - Ressources et fichiers

Voici une liste des sites utiles :

Liens utiles

- [Google Analytics API](#)
- [Google Analytics Data API - Reference](#)
- [Google Authentication for Web Applications](#)
- [Package GData du Zend Framework](#)
- [Documentation de GData](#)

Et les fichiers utilisés dans ce tutoriel :

Fichiers utilisés dans ce tutoriel

- [Exemple simple : lister les profil du compte](#)
- [Exemple POO : nombre de page vues pour chaque compte](#)

je vous remercie d'avoir lu ce tutoriel jusqu'au bout, et espère qu'il saura vous être utile ;-)