

OAuth Application Configuration

For a client application to be authenticated in ShopSite using OAuth, it must first be registered in the ShopSite Back Office by going to Utilities -> Applications and clicking the Add button. Enter a name for the application and check the boxes for the access permissions that the application should be granted. Then click the Generate Key button.

The Access Grant screen displays the client credentials that are necessary for the application to be authenticated when it requests access to protected data in ShopSite. Each of the items on this screen needs to be copied and pasted into the client application configuration or saved in a securely protected file for access at run time.

CAUTION: The secret key must always be safeguarded to prevent unauthorized use! It should never be transmitted over the internet, including in e-mails, using an insecure connection. If the secret key is believed to have been compromised, the current client credentials should be deleted immediately and new credentials generated, which will include a new secret key.

RECOMMENDATION: For added security, the secret key in your application configuration should be in encrypted form instead of clear text. Include code in your application to decrypt the key when it comes time for the key to be used. Immediately after use, the application should erase the secret key and free the memory.

The Edit button in Utilities -> Applications allows you to change the application name or the access grant permissions. The client credentials are displayed as read-only; they cannot be changed.

Although you cannot change the Authorization URL directly, it will reflect any changes made to the shopping cart SSL configuration. If the Authorization URL has changed, remember to update it in the application configuration.

Requesting an Access Token

Each time a client application requires access to protected data in ShopSite, it must first obtain an access token from the authorization server. The Authorization URL is used to make the request along with the following query parameters in the HTTP request body using the "application/x-www-form-urlencoded" format:

grant_type	Value must be set to "authorization_code".
code	The authorization code obtained when the application was registered.
client_credentials	The client id and a nonce* combined in the string format: "client id:nonce". This string is base64 encoded and saved as encoded credentials.
signature	The result of an HMAC computed hash of the encoded credentials, signed with the secret key, and then base64 encoded. The hash must be computed using the SHA1 hashing algorithm.

*A nonce as used here is a randomly generated string of characters used to provide variability when combined with other data to compute a hash or digest for authentication.

Example:

client id	B9D3299B
authorization code	MTQxMDZ8ZGVlYmllfDJ8
secret key	CCAB-DFB5-C73A-5228
nonce	80026bc7
client credentials	B9D3299B: 80026bc7
encoded credentials	QjIEMzI5OUl6ODAwMjZiYzc=
signature	r fw0k21I9YdXlOr240Uw+3aa9A4=

POST /authorize.cgi HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded

```
grant_type=authorization_code&code=MTQxMDZ8ZGVlYmllfDJ8&client_credentials=
QjIEMzI5OUl6ODAwMjZiYzc=&signature=r fw0k21I9YdXlOr240Uw+3aa9A4=
```

When the authorization server receives the above request from the client, it uses the authorization code to look up the client information in the applications database. It then uses the nonce passed in the client credentials along with the client id and secret key in the database to compute its own signature. The computed signature is compared to the one passed by the client, and if they match, the server downloads authorization information to the client, which is in JavaScript Object Notation (JSON) format. Here is an example:

```
{
  "access_token" : "MTMwMjA5NjAyOHxkZWJiaWV8MnxCYWlsZXI8MjU2fA==",
  "token_type" : "MAC",
  "expires_in" : 30,
  "download_url" : "https://server.example.com/path/db_xml.cgi"
  "upload1_url" : "https://server.example.com/path/dbupload.cgi"
  "upload2_url" : "https://server.example.com/path/dbmake.cgi"
  "publish_url" : "https://server.example.com/path/generate.cgi"
}
```

The access token is opaque to the client and has meaning only to the resource server. It contains the permissions and other information necessary to make protected data in ShopSite available to the client.

The token type identifies the message hashing scheme used for the token. "MAC" stands for *Message Authentication Code*, or keyed hash.

The token expiration time is given in number of seconds. It is followed by the URLs that are to be used for downloading, uploading, and publishing.

A routine named *oauth_resource_access()* is provided in the sample code to parse the JSON object and to put each of the data elements in a RESOURCE data structure.

Requesting Data Access

After receiving an access token from the authorization server, the client application next prepares and makes a request to the resource server to perform an upload, download, or publish operation.

Since the access token is a MAC token, it is necessary to assemble the query with the information needed to create a signed message digest for MAC authentication. This process is described in the following working document:

Hammer-Lahav, E., "*HTTP Authentication: MAC Authentication*", draft-hammer-oauth-v2-mac-token-02
<http://tools.ietf.org/html/draft-hammer-oauth-v2-mac-token-02>

Begin the query with the usual *XML Programmatic Interface* parameters. To download orders, for example, start the query as:

```
"clientApp=1&dbname=orders&version=10.2"
```

A new nonce is calculated along with a timestamp. You then add these to the query string along with the access token.

Example:

```
access token  MTMwMjA5NjAyOHxkZWJiaWV8MnxCYWlsZXI8MjU2fA==  
nonce        503bb763  
timestamp    1302101084
```

```
"clientApp=1&dbname=orders&version=10.2&token=MTMwMjA5NjAyOHxkZWJiaWV8MnxCYWlsZXI8MjU2fA==&timestamp=1302101084&nonce=503bb763"
```

If you are downloading orders and you need to have the payment information included in the download, you must add the "pay=x" parameter to the query, where the value "x" is either "no_cvv" to **not** include the CVV2 number or any other value to include it. CAUTION: The CVV2 number can be downloaded only once, after which it is purged and is no longer accessible. The same rule applies when it is viewed in the back office.

A secure SSL connection is required for payment information to be downloaded. Additionally, the client application must have the proper access permissions granted.

If payment information has been encrypted with a merchant key, then this key must be included in the query in order for the information to be decrypted when it is downloaded. The merchant key is stored somewhere in a file on the local system. The content of this file is read into memory, base64 encoded, and then added to the query using the “dkey” parameter name. In the sample code there is a routine named *addMerchantKeyToQuery()* that shows how this is done.

After all of the required parameters have been added to the query, call the *oauth_message_digest()* routine in the sample code to create a signed message hash. The input parameters are the request URL (for uploading, downloading, or publishing), the query string, and the secret key. A signature is returned in the output parameter. Add the signature to the query string and then submit the request to the resource server.

Example:

```
POST /db_xm.cgi HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
```

```
clientApp=1&dbname=orders&version=10.2&token=MTMwMjA5NjAyOHxkZWJiaWV8MnxCYWlsZXI8MjU2fA==&timestamp=1302101084&nonce=503bb763&signature=LXIUwI9RSLqzd19GpOJnbO0hTQg=
```

If successful, the results of the operation are downloaded to the client. If the client then wishes to perform another operation, it must go back to the authorization server and request another access token.

Sample Code

Sample “C” code is provided to illustrate in more detail how OAuth authentication is done and how the *XML Programmatic Interface* is used within OAuth. Included in the sample code are the following “C” functions to perform the required data manipulation operations used in the examples:

```
oauth_encode();           // base64 encode a string
oauth_nonce();           // compute a nonce
oauth_signature();       // compute a HMAC hash and signature
oauth_message_digest()  // create a MAC message digest from a given request URL
oauth_resource_access()  // parse authorization information to access protected data
```

References:

OAuth Specifications: <http://oauth.net/documentation/spec/>

OAuth 2.0 Working Draft: <http://tools.ietf.org/html/draft-ietf-oauth-v2-15>

MAC Authentication: <http://tools.ietf.org/html/draft-hammer-oauth-v2-mac-token-02>