Conexim DNS API Documentation

Last Updated August 2013

Contents

Introduction

DNS is at the foundation of your web sites, email, applications and cloud services, so it's important that it's secure, always available and delivers high performance. Conexim is trusted by thousands of companies to deliver professional, managed DNS services that meet the strictest requirements for performance, reliability and security.

Conexim's Managed DNS platform has been designed from the ground up with years of Internet engineering experience in delivering always available DNS services.

The DNS platform is managed through Conexim's management portal which provides a familiar interface for managing all Conexim managed services. It provides powerful DNS functionality beyond basic DNS zone management in a way that is both accessible and helps avoid common mistakes that may compromise availability or performance of DNS.

Feedback

Conexim encourages and welcomes feedback to ensure we're delivering the best services possible. If you have ideas for a new feature, or if there's an aspect of Conexim DNS you feel we could improve, please let us know – we'd love to hear from you.

Please send feedback to feedback@conexim.com.au.

1. Authentication and Authorisation

1.1 Authentication

The Conexim DNS RESTful API makes use of SHA-256 HMAC (keyed-hash message authentication code) to generate a hashed signature of all parameters passed to the API servers. A shared secret exists between the Client and the Server, but the secret key is never sent with the request, thereby maintaining security. The method in its simplest form is described in RFC 2104.

All requests are sent over an SSL channel using secure ciphers ensuring secrecy of all parameters sent as part of the transaction.

If an SSL session is decrypted, it's unlikely an API call can be replayed. This is because the current GMT UNIX Time is sent as a separate header (**Conexim-Time)**.

The maximum clock skew between the API server and the client is 5 minutes, so it's important to ensure that the client's clock is accurate. A 401 response is returned with the message "Client clock skew is greater than maximum allowed." if the date sent is not within the bounds of the allowed period.

There are two parts to Conexim DNS Authentication which are both sent as the **Authorization** HTTP header. A typical HTTP request Authorization header appears as follows:

Authorization: CONEXIM <key>:<sig>

In the above example, <key> represents the ID associated with the key and <sig> represents the dynamically generated signature based on the request and its parameters.

The signature is formulated as follows:

sigString = **KeyId** + "\n" +
UnixTime + "\n" +
Verb + "\n" +
Action + "\n" +
URLencodedParameters

URLencodedParameters are simply any parameters that are sent for a PUT/POST request encoded in a URL-encoded format (e.g. paramA=valA&paramB=valB). Parameters must be pre-sorted to ensure that they're reconstructed in the same order on the server for verification.

The signature digest is generated by applying a SHA-256 HMAC function against the signature and secret key. The result is converted to Base64 for adding to the header.

1.2 Authorisation

The Conexim DNS API makes use of commonly used RESTful verbs over HTTP. A given API Key can have these parameters enabled or disabled at any time from within the DNS Management Console.

Verb	Action
GET	Read records
PUT	Update records
POST	Create new records
DELETE	Delete records

1.3 Managing API Keys and Permissions

API access is managed through the My Conexim console. Please refer to the **Conexim DNS Administrator's Guide** for details on using Conexim DNS and managing user access.

Once logged in to Conexim DNS, the first step is to create an API key and assign the desired permissions to it. Keys and their associated IDs are generated on creation.

2. Responses

Conexim DNS returns JSON response data.

In addition to call-specific responses, HTTP response codes are used to signify status. The following response codes have been implemented:

HTTP Response	Description
200	Successful API call.
401	Request is not properly authenticated.
404	Returned if the object is not found.
500	Malformed request – check the parameters passed.
501	The API call is not recognised.

When developing your application, it can be assumed that response code **200** (successful API call) will return valid JSON as the response. Response codes other than 200 will represent a HTML error response.

3. Performing API Calls

The following pseudo code represents how a typical request is formulated made:

```
//The Key Identifier
$keyId = "51f881d6d158f"

//The Key Shared Secret
$keyData = "19d6a39ee3edf428ae15684202624 1bc7c20f5c793946d9e9..."

$baseURL = "https://api.conexim.com.au"

//API call to be made
$action = "/api/dns/v1/domains/testzone.tld"

//Verb (GET/POST/PUT/DELETE)
$verb = "GET"

//Encode all parameters as a key/value JSON request
$jsonRequest = { "param1": "val1",
                 "param2": "val2" }

//Generate the signed data. Parameters must be pre-sorted
$sigString = $keyId+"\n"+
             $time + "\n" +
             $verb + "\n" +
             $action + "\n" +
             "param1=val1&param2=val2"

//Generate a signature for the request
$signedString = base64_encode(hash_hmac("SHA256", $sigString,
$keyData))

$httpObject = new HTTPRequestObject()

//Set the Content-type header to application/json
$httpObject.Headers['Content-type'] = 'application/json'
$httpObject.Headers['Conexim-Time'] = $time

//Set the request method as either GET, POST, DELETE or PUT
$httpObject.Method = $verb

//Set the post fields as a JSON request
$httpObject.PostData = $jsonRequest

//Execute the request and retrieve the response as JSON
$resultSet = $httpObject.execute()
```

Data returned by a successful API call (HTTP response code 200) is returned as JSON data. Responses other than 200 are basic HTML errors messages.

3.1 Example in PHP

The following example code can be used as a framework to build an interface to the Conexim API in PHP.

```php
<?php

$secretkey =
'19d6a39ee3eccd2df428ae156842026241bc7b3118f4c20f5c793946d9e9668b6a7c7eb3ee9bfe658d9
82aadd0716dc15d3ae38fccf7e26a0004877b2c6b78c4';
$keyid = '51f881d6d158f';

$server = 'https://api.conexim.com.au';
$action = '/api/dns/v1/domains';
$verb = 'GET';

$params = array();
ksort($params);

$utimeGMT = (int)gmdate('U');

$sigString = $keyid . "\n" .
                $utimeGMT . "\n" .
                $verb . "\n" .
                $action . "\n" .
                http_build_query($params);

$sig = hash_hmac("sha256", $sigString, $secretkey, true);
$ch = curl_init($server . $action);

$headers[] = 'Content-type: application/json';
$headers[] = 'Authorization: CONEXIM ' . $keyid . ':' . $sig;
$headers[] = 'Conexim-Time: ' . $utimeGMT;

$options = array(
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_HTTPHEADER => $headers,
        CURLOPT_CUSTOMREQUEST => $verb,
        CURLOPT_POSTFIELDS => json_encode($params),
        CURLOPT_RETURNTRANSFER => true,
);

$responseCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);

echo "Response Code: " . $responseCode . "\n";
echo "Response Data:\n";
print_r(json_decode($result));

?>
```

3.2 Example in Python

The following example code can be used as a framework to build an interface to the Conexim API in Python.

```
import urllib2
import urllib
import hmac
import hashlib
import base64
import json
import pprint
import calendar
import time
import collections

secretkey =
'19d6a39ee3eccd2df428ae156842026241bc7b3118f4c20f5c793946d9e9668b6a7c7eb3ee9bfe658d9
82aadd0716dc15d3ae38fccf7e26a0004877b2c6b78c4'
keyid = '51f881d6d158f'

server = 'https://api.conexim.com.au'
action = '/api/dns/v1/domains/testzonefromapi.net/records'
verb = 'POST'

params = { 'type': 'A',
           'name': 'www',
           'value': '123.123.123.123' }

params = collections.OrderedDict(sorted(params.items()))

utimeGMT = "%d" % calendar.timegm(time.gmtime())

sigString = keyid + '\n' + utimeGMT + '\n' + verb + '\n' + action + '\n'  + urllib.urlencode(params)
sig = base64.b64encode(hmac.new(key=secretkey, msg=sigString,
digestmod=hashlib.sha256).digest())

request = urllib2.Request(server + action, json.dumps(params))
request.add_header('Authorization', 'CONEXIM ' + keyid + ':' + sig)
request.add_header('Content-type', 'application/json')
request.add_header('Conexim-Time', utimeGMT)
request.get_method = lambda: verb

try:
    opener = urllib2.urlopen(request)
    retdata = json.loads(opener.read())

    print "Response Code: " + str(opener.getcode())
    print "Contents:"
    pprint.pprint(retdata)
except urllib2.HTTPError, err:
    print err.read()
```

3.3 Example in Ruby

The following code can be used with Ruby without the Rails framework.

```ruby
require 'net/https'
require 'openssl'
require 'base64'
require 'json'
require 'time'

secretkey =
'19d6a39ee3eccd2df428ae156842026241bc7b3118f4c20f5c793946d9e9668b6a7c7eb3ee9bfe658d9
82aadd0716dc15d3ae38fccf7e26a0004877b2c6b78c4'
keyid = '51f881d6d158f'

server = URI('https://api.conexim.com.au')
action = '/api/dns/v1/domains'
verb = 'GET'

params = Hash[]

queryString = ''
paramSep = ''
params.sort.map do |key, value|
  queryString << paramSep << key << "=" << value
  if paramSep == ''
    paramSep = '&'
  end
end

utimeGMT = Time.now.getutc.to_i()

sigString = "#{keyid}\n#{utimeGMT}\n#{verb}\n#{action}\n#{queryString}"
digest = OpenSSL::Digest::Digest.new("sha256")
hmac = OpenSSL::HMAC.digest(digest, secretkey, sigString)
sig = Base64.strict_encode64(hmac)

Net::HTTP.start(server.host, server.port,
  :use_ssl => server.scheme == 'https') do |http|
    case verb
      when 'GET'
        req = Net::HTTP::Get.new(action)
      when 'POST'
        req = Net::HTTP::Post.new(action)
      when 'PUT'
        req = Net::HTTP::Put.new(action)
      when 'DELETE'
         req = Net::HTTP::Delete.new(action)
    end

    req.body = params.to_json

    req['Authorization'] = "CONEXIM #{keyid}:#{sig}"
    req['Conexim-Time'] = utimeGMT.to_s()

    response = http.request(req)

    puts JSON.pretty_generate(JSON.parse(response.body))
end
```

4. Managing DNS Records from the API

4.1 Retrieving DNS Zones and Records

4.1.1 Retrieving DNS Zones

Retrieving a list of all DNS zones and the associated SOA data is performed with the following API call.

Verb: GET

Action: /api/dns/v1/domains

Parameters: (none)

Returns: JSON response (as documented below).

Retrieving a specific zone can be performed either based on its domain name or its numeric ID.

Verb: GET

Action: /api/dns/v1/domains/<domain name | ID>

Parameters: (none)

Returns: JSON response (as documented below).

JSON Response

A typical JSON response appears as below.

```
{
    "<ID>": {
        "domain": "testdomain1.tld",
        "last_updated": null,
        "master_server": null,
        "soa_admin": "hostmaster.conexim.com.au.",
        "soa_expiry": "86400",
        "soa_minimum": "86400",
        "soa_ns": "ns.cdns1.net",
        "soa_refresh": "28800",
        "soa_retry": "7200",
        "soa_serial": "2009021002",
        "template_id": "0",
        "type": "native"
    }...,
}
```

The attributes of the returned JSON request are as follows:

Attribute	Description
<ID>	The unique identifier for the zone/domain. This can be used as an alternative to specifying the domain name.
domain	The domain name associated with the zone.
master_server	When running in master mode, this represents the IPv4/IPv6 address of the master server.
soa_admin	Email address of the person in charge of the domain name.
soa_expiry	Slaves stop responding to queries for the zone when this time (in seconds) has expired and no contact has been made with the master.
soa_minimum	The default TTL (time-to-live) for resource records, how long data will remain in other nameserver's cache.
soa_ns	Primary master nameserver. Any name server that will respond authoritatively for the domain.
soa_refresh	Interval for the slave to try to refresh the zone from the master.
soa_retry	Interval between retries if the slave (secondary) fails to contact the master when refresh has expired.
soa_serial	Incremental serial number – automatically updated on record change.
template_id	If this zone is associated with a template, this attribute refers to the parent template ID.
type	Indicates if the zone is Native (fully managed by Conexim DNS), Slave or Master.

4.1.2 Retrieving Records for a Specific Zone

Retrieving a list of all records and their attributes for a specific zone is performed as follows:

Verb: GET

Action: /api/dns/v1/domains/<domain name | ID>/records

Parameters: (none)

Returns: JSON response (as documented below).

Retrieving a specific record can be performed by specifying its ID.

Verb: GET

Action: /api/dns/v1/domains/<domain name | ID>/records/<record ID>

Parameters: (none)

Returns: JSON response (as documented below).

JSON Response

A typical JSON response appears as below.

```
{
    "<ID>": {
        "domain_id": "1181",
        "name": "",
        "prio": "0",
        "template_id": "0",
        "template_record_id": "0",
        "ttl": "86400",
        "type": "A",
        "value": "123.123.123.43"
},
```

Attribute	Description
<ID>	The unique identifier for the resource record.
domain_id	The unique identifier of the domain (zone) to which this record belongs.
prio	Record Priority – typically only used with MX records.
template_id	If the zone belongs to a template, indicate the template ID.
template_record_id	If the zone belongs to a template, indicate the template's corresponding record ID.
ttl	The default TTL (time-to-live) for the record - how long data will remain in other nameserver's cache.
type	The RR type (MX, CNAME, A, AAAA etc).
value	Value associated with the record.

4.2 Creating and Updating DNS Zones and Records

Updating records is similar to querying them. The main differences are:

1. Attributes that are to be updated are sent in JSON format.
2. A PUT HTTP request for updating records and a POST request is used for creating new records.

4.2.1 Creating and Updating DNS Zone Information

The table below indicates what attributes may be updated and those that can be specified on zone creation.

Attribute	Update (PUT)	Create (POST)	Description
<ID>	No	No	System generated unique Identifier
domain	No	Yes*	Domain name
type	Yes	Yes	Native, Master, Slave mode
master_server	Yes	Yes	Change the master server (slave mode only)
template_id	No	No	Template ID – set when a zone is created from a template.
soa_admin	Yes	Yes	Email address of the person in charge of the zone.
soa_expiry	Yes	Yes	Slaves stop responding to queries for the zone when this time (in seconds) has expired and no contact has been made with the master.
soa_minimum	Yes	Yes	The default TTL (time-to-live) for resource records, how long data will remain in other nameserver's cache.
soa_ns	Yes	Yes	Primary master nameserver. Any name server that will respond authoritatively for the domain.
soa_refresh	Yes	Yes	Interval for the slave to try to refresh the zone from the master.
soa_retry	Yes	Yes	Interval between retries if the slave (secondary) fails to contact the master when refresh has expired.
soa_serial	Yes	Yes	Incremental serial number – automatically updated on record change.

*Mandatory when creating a new zone.

To create a new zone, the POST method is used:

Verb: POST

Action: /api/dns/v1/domains

Parameters: JSON-formatted set of attributes to be modified.

Returns: JSON response (as documented below).

To update an existing zone, the PUT method is used in a similar way:

Verb: PUT

Action: /api/dns/v1/domains/<domain name | ID>

Parameters: JSON-formatted set of attributes to be modified.

Returns: JSON response (as documented below).

JSON Response

The following JSON response is returned on execution of the API call. The response will provide both a success/fail response and provide details about any failed validation. The response also returns the newly generated ID if creating a zone.

```
{
    "id": "<ID>",
    "message": "Created <domain name> OK.",
    "result": "true"
}
```

The result is similar for a PUT (update) request appears as:

```
{
    "message": "Update OK",
    "result": "true"
}
```

The attributes returned are as follows:

Attribute	Description
<ID>	The unique identifier for the resource record created. Only sent on successful creation.
message	Response message – includes any validation information for invalid attributes.
result	True/false response to indicate successful creation.

4.2.2 Creating and Updating DNS Records

Creating an updating Resource Records (RRs) is managed in a similar fashion to managing zones.

Attributes that are sent as parameters are as follows:

Attribute	Update (PUT)	Create (POST)	Description
<ID>	No	No	System generated unique Identifier.
name	Yes	Yes*	Name of the record (e.g. "www" – no not specify the FQDN).
type	Yes	Yes*	The RR type (MX, CNAME, A, AAAA etc).
value	Yes	Yes*	Value associated with the record.
ttl	Yes	Yes	The default TTL (time-to-live) for the record - how long data will remain in other nameserver's cache.
prio	Yes	Yes	Record Priority – typically only used with MX records.
template_id	No	No	If the zone belongs to a template, indicate the template ID.
template_record_id	No	No	If the zone belongs to a template, indicate the template's corresponding record ID.

*Mandatory when creating a new resource record.

To create a new Resource Record:

Verb: POST

Action: /api/dns/v1/domains/<domain name | ID>/records

Parameters: JSON-formatted set of attributes to be modified.

Returns: JSON response (as documented below).

To update an existing Resource Record:

Verb: PUT

Action: /api/dns/v1/domains/<domain name | ID>/records/<record ID>

Parameters: JSON-formatted set of attributes to be modified.

Returns: JSON response (as documented below).

JSON Response

The following JSON response is returned on execution of the API call. The response will provide both a success/fail response and provide details about any failed validation. The response also returns the newly generated ID if creating a new record.

```
{
    "id": "<ID>",
    "message": "Created record OK",
    "result": "true"
}
```

Setting a record with the client's IP address

If the value of an A or AAAA record is set to "self", the value of the record is set to the client's IP address. Care should be taken to ensure that the client is connecting over IPv4 when updating/creating an A record or IPv6 when updating/creating an IPv6 record.

4.3 Deleting DNS Zones and Records

The deletion of DNS Zones and Records is very similar to reading zone and record information. The only difference is that the DELETE HTTP method is used instead of GET.

Deleting a Zone

Verb: DELETE

Action: /api/dns/v1/domains/<domain name | ID>

Parameters: (none)

Returns: JSON response (as documented below).

Deleting a Record

Verb: DELETE

Action: /api/dns/v1/domains/<domain name | ID>/<record ID>

Parameters: (none)

Returns: JSON response (as documented below).

JSON Response

The JSON response simply indicates a true/false result for the zone or record deletion.

```
{
    "result": "true"
}
```

5 Contact

Conexim Australia Pty. Limited

W: www.conexim.net

E: info@conexim.net

Australia: 1300 133 900

New Zealand: 0800 450 236

USA: 0800 450 236