Overview

The documentation pages in this section describe the RESTful APIs included with Cascade Profiler and Cascade Express products. It is assumed that the reader has practical knowledge of RESTful APIs, so the documentation does not go into detail about what REST is and how to use it. Instead the documentation focuses on what data can be accessed and how to access it.

The primary focus of the current version of the API is on providing access to common data. The following information can be accessed via the API:

- System information (serial number, model, etc.)
- Information and resources for authenticating (login, logout, oauth 2.0, etc.)

Details about REST resources can be found in the Resources section.

SSL

All communication to the profiler is SSL encrypted on Port 443. There is no support for access to the profiler on the standard HTTP port 80.

Ciphers

The ciphers supported by the Profiler may change, depending on security setting (e.g., FIPS mode). Any client when initiating a request must include one or more ciphers available in the Profiler's configured list. Otherwise, the client will receive an SSL error indicating that there is no cipher overlap, and will be unable to connect.

Certificate

The profiler by default uses a self-signed certificate for SSL communication. The client should be able to handle this, by permitting self-signed certificates.

Examples

Using the 'curl' command line client to request the services resource on a non-FIPS box. The -k switch is used to allow the self-signed certificate, and a cipher suite (SSL v3) is provided.

```
curl -k -3 https://hostname:443/api/common/1.0/services
```

Using the 'curl' command line client to request the services resource on a FIPS box. An explicit cipher is selected.

```
curl --ciphers rsa_aes_256_sha -k https://hostname:443/api/common/1.0/services
```

Known Issues

Some clients, such as Curl (both as a library and a command line executable), do not support both an explicit cipher list, and a cipher suite. The following command will fail on a FIPS Profiler:

```
curl --ciphers rsa_aes_256_sha -3 -k https://hostname:443/api/common/1.0/services
```

This is because the cipher suite (-3) overrides the --ciphers argument. Clients with this issue will receive a 'no cipher overlap' error, even if they have explicitly provided a cipher that is known to be FIPS compliant.

BASIC Authentication

For BASIC authentication the request header "Authorization" must be set to a base64-encoded string of username:password.

If the "Authorization" header is not provided, the "WWW-Authenticate" response header is returned. Basic authentication has a built-in support in various tools. Refer to the coding examples.

Example of client request to protected resource using Basic Authentication:

```
POST /api/profiler/1.0/ping
Host: 127.0.0.1
Accept: application/json
Authorization: Basic YWRtaW46YWRtaW4=
```

Server response:

```
HTTP/1.1 204 OK
```
Sample PHP script for BASIC authentication

Use the Ping resource to demonstrate BASIC authentication.

```php
<?php

define('HOST', '127.0.0.1'); // IP address of Profiler
define('BASIC_AUTH', 'admin:admin');

// HTTP GET
function do_GET($url, &$info) {
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_HTTPAUTH, CURLAUTH_BASIC );
    curl_setopt($curl, CURLOPT_USERPWD, BASIC_AUTH);
    curl_setopt($curl, CURLOPT_SSLVERSION,3);
    curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, 2);
    curl_setopt($curl, CURLOPT_HEADER, true);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_HTTPHEADER, array('Accept: application/json'));
    $output = curl_exec($curl);
    $info   = curl_getinfo($curl);
    curl_close($curl);

    $headers = substr($output, 0, $info['header_size']);
    $headers = explode("n", $headers);
    $info['headers'] = $headers;
    $body = substr($output, $info['header_size']);
    return $body;
}

// Ping to test basic authentication
$url = 'https://' . HOST . '/api/profiler/1.0/ping';
echo "GET $url\n";

$info = array();
$output = do_GET($url, $info);
if ($info['http_code'] == 204) {
    echo "Ping is successful\n";
} else {
    echo "Ping failed\n";
    echo $output;
}
?>
```

Sample Python script for BASIC authentication

Use the Ping resource to demonstrate BASIC authentication.

```python
Sample Python script for BASIC authentication

Use the Ping resource to demonstrate BASIC authentication.
```
Sample Perl script for BASIC authentication

Use the Ping resource to demonstrate BASIC authentication.
use strict;
use warnings;
use LWP::UserAgent;
use HTTP::Request;
use List::MoreUtils qw(firstidx);
use JSON qw( encode_json decode_json );

use constant HOST => '127.0.0.1';
use constant LOGIN => 'admin';
use constant PASSWORD => 'admin';

our $ua = LWP::UserAgent->new;
$ua->agent("ProfilerScript/0.1");

our $API_BASE = "https://127.0.0.1";

sub _request($) {
    my $req = shift;
    $req->header('Accept' => 'application/json');
    $req->authorization_basic(LOGIN, PASSWORD);
    my $res = $ua->request($req);
    return {
        code => $res->code,
        status => $res->status_line,
        headers => $res->headers(),
        data => $res->content
    };
}

sub GET($) {
    my $req = HTTP::Request->new(GET => $API_BASE . shift);
    return _request($req);
}

# Ping to test basic authentication

print "GET /api/profiler/1.0/ping\n";
my $response = GET('/api/profiler/1.0/ping');

if ($response->{code} == 204) {
    print "Ping is successful!\n";
} else {
    print "Ping failed!\n";
    print $response->{data};
}

Sample .Net/C# script for BASIC authentication

using System;
using System.Collections.Generic;
using System.Net;
using System.Text;
using System.IO;
using System.Net.Security;

namespace CascadeRestClient
{
    class Program
    {
        static string BASIC_AUTH = "admin:admin";

        // callback used to validate the self-gen certificate in an SSL conversation
        private static bool ValidateRemoteCertificate(object sender, X509Certificate cert, X509Chain chain, SslPolicyErrors policyErrors)
        {
            return true;
            /*
             * X509Certificate2 certv2 = new X509Certificate2(cert);
             * if (certv2.GetNameInfo(X509NameType.SimpleName,true) == "www.riverbed.com")
             *     return true;
             * return false;
             */
        }
    }
}
private static string Base64Encode(string toEncode)
{
    byte[] toEncodeAsBytes
    = System.Text.ASCIIEncoding.ASCII.GetBytes(toEncode);
    return System.Convert.ToBase64String(toEncodeAsBytes);
}

static void Main(string[] args)
{
    if (args.Length == 0 || string.IsNullOrWhiteSpace(args[0]))
    {
        Console.WriteLine("Usage: CascadeRestClient hostname");
        return;
    }
    try
    {
        // Code to allow run with self-signed certificates
        // validate cert by calling a function
        ServicePointManager.ServerCertificateValidationCallback
            += new RemoteCertificateValidationCallback(ValidateRemoteCertificate);

        // Starting to run rest
        string rootUrl = "https:" + args[0];
        string requestUrl = rootUrl + "/api/profiler/1.0/ping";

        // Ping to test basic authentication
        Console.WriteLine("GET " + requestUrl);
        // POST to run the report
        HttpWebRequest request = WebRequest.Create(requestUrl) as HttpWebRequest;
        request.Headers.Add("Authorization: Basic " + Base64Encode(BASIC_AUTH));
        request.ContentType = "application/json";
        request.ContentLength = 0;
        using (var response = request.GetResponse() as HttpWebResponse)
        {
            if (response.StatusCode == HttpStatusCode.NoContent)
            {
                Console.WriteLine("Ping is successful!");
            }
            else
            {
                Console.WriteLine("Ping failed!");
                using (Stream stream = response.GetResponseStream())
                {
                    using (StreamReader reader = new StreamReader(stream, Encoding.UTF8))
                    {
                        String responseString = reader.ReadToEnd();
                        Console.WriteLine(responseString);
                    }
                }
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

### Sample CURL command (BASIC authentication)

Use the Ping resource to demonstrate BASIC authentication.

```
% curl --user username:password https://{host}/api/profiler/1.0/ping -k
```

### Sample WGET command (BASIC authentication)

Use the Ping resource to demonstrate BASIC authentication.

```
% wget --http-user username --http-password password https://{host}/api/profiler/1.0/ping --no-check-certificate
```

### SESSION (Cookie) authentication

In order to use the SESSION (Cookie) authentication, a session ID must be generated. The session ID can then be used to access protected resources. To generate a session ID the client must send a POST request with username, password and optionally purpose. The API supports three different methods of input: x-www-form-urlencoded, JSON and XML.
Client request using x-www-form-urlencoded input:

POST /api/common/1.0/login
Host: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Accept: application/json

username=username&password=password&purpose=script XYZ

Server response:

HTTP/1.1 200 OK
Set-Cookie: SESSID=bfe3c2fd7b53053eeccd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

{  
  "session_key": "SESSID",
  "session_id": "bfe3c2fd7b53053eeccd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7"
}

Client request using JSON input:

POST /api/common/1.0/login
Host: 127.0.0.1
Content-Type: application/json
Accept: application/json

{  
  "username" : "username",
  "password" : "password",
  "purpose"  : "script XYZ"
}

Server response:

HTTP/1.1 200 OK
Set-Cookie: SESSID=bfe3c2fd7b53053eeccd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

{  
  "session_key": "SESSID",
  "session_id": "bfe3c2fd7b53053eeccd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7"
}

Client request using XML input:

POST /api/common/1.0/login
Host: 127.0.0.1
Content-Type: text/xml
Accept: text/xml

<login username="user" password="pass" purpose="UI login"/>

Server response:

HTTP/1.1 200 OK
Set-Cookie: SESSID=bfe3c2fd7b53053eeccd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

<login username="user" password="pass" purpose="UI login"/>

The client must include the Cookie header when accessing authenticated resources. The session (cookie) expiration rules are the same as the ones used in the GUI of the product. The rules can be changed from the Log-in Settings page.

Client request to protected resource using the session ID:

POST /api/profiler/1.0/ping
Host: 127.0.0.1
Accept: application/json
Cookie: SESSID=bfe3c2fd7b53053eeccd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

Server response:

HTTP/1.1 204 OK

Client request to protected resource using expired/invalid session ID:

POST /api/profiler/1.0/ping
Host: 127.0.0.1
Accept: application/json
Cookie: SESSID=bfe3c2fd7b53053eeccd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

Server response:
HTTP/1.1 401 AUTH_INVALID_SESSION
Content-Type: application/json

{
    "error_id": "AUTH_INVALID_SESSION",
    "error_text": "Session ID is invalid"
}

To end a previously started session, the client sends a GET request to /logout including a Cookie header with the session ID.

Client request to end a previously started session:

GET /api/common/1.0/logout
Host: 127.0.0.1
Accept: application/json
Cookie: SESSID=bfe3c2fd7b53053eeccdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

Server response:

HTTP/1.1 204

Sample PHP script for SESSION (Cookie) authentication

Use the Ping resource to demonstrate SESSION (Cookie) authentication.

```php
<?php

define('HOST', '127.0.0.1'); // IP address of Profiler

// HTTP POST
function do_POST($url, $string, &$info) {
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_SSLVERSION,3);
    curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, 2);
    curl_setopt($curl, CURLOPT_HEADER, true);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type: application/json','Accept: application/json'));
    curl_setopt($curl, CURLOPT_POST, 1);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $string);
    $output = curl_exec($curl);
    $info = curl_getinfo($curl);
    $headers = substr($output, 0, $info['header_size']);
    $headers = explode("\n", $headers);
    $info['headers'] = $headers;
    $body = substr($output, $info['header_size']);
    return $body;
}

// HTTP GET
function do_GET($url, $session_key, $session_id, &$info) {
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_SSLVERSION,3);
    curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, 2);
    curl_setopt($curl, CURLOPT_HEADER, true);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_HTTPHEADER, array('Accept: application/json','Cookie: '.$session_key.'='.$session_id));
    curl_setopt($curl, CURLOPT_HTTPGET, true);
    $output = curl_exec($curl);
    $info = curl_getinfo($curl);
    $headers = substr($output, 0, $info['header_size']);
    $headers = explode("\n", $headers);
    $info['headers'] = $headers;
    $body = substr($output, $info['header_size']);
    return $body;
}

// Post to create session id
LOGIN_data = array('username' => 'admin','password' => 'admin', 'purpose' => 'demonstrate SESSION authentication');
$url = 'https://' . HOST . '/api/common/1.0/login';
$output = do_POST($url, json_encode($LOGIN_data), $info);
```
Sample Python script for SESSION (Cookie) authentication

Use the Ping resource to demonstrate SESSION (Cookie) authentication.
from urlparse import urlparse
import base64
import logging
import httplib
import json
import time
import sys

HOST = '127.0.0.1'

# Lib functions

def do_POST(url, string):
    '''HTTP POST'''
    conn = httplib.HTTPSConnection(HOST, 443)
    headers = {
        "Content-Length" : str(len(string)),
        "Content-Type" : "application/json",
        "Accept" : "application/json"
    }
    conn.request('POST', url, body=string, headers=headers)
    res = conn.getresponse()
    info = {
        "status" : res.status,
        "headers" : res.getheaders()
    }
    data = res.read()
    conn.close()
    return data, info

def do_GET(url, session_key, session_id):
    '''HTTP GET'''
    conn = httplib.HTTPSConnection(HOST, 443)
    headers = {
        "Content-Length" : 0,
        "Content-Type" : "application/json",
        "Accept" : "application/json",
        "Cookie" : "%s=%s" % (session_key, session_id)
    }
    conn.request('GET', url, body='', headers=headers)
    res = conn.getresponse()
    info = {
        "status" : res.status,
        "headers" : res.getheaders()
    }
    data = res.read()
    conn.close()
    return data, info

# Post to create session id
login_data = {
    "username" : "admin",
    "password" : "admin",
    "purpose" : "demonstrate SESSION authentication"
}
url = "https://%s/api/common/1.0/login" % HOST
output, info = do_POST(url, json.dumps(login_data))
if info['status'] != 200:
    print "Login Failed!"
    print output
    sys.exit(1)
data = json.loads(output)
session_key = data['session_key']
session_id  = data['session_id']
print "Login successful, %s=%s" % (session_key, session_id)

url = "https://%s/api/profiler/1.0/ping" % HOST
# Ping to test session authentication
output, info = do_GET(url, session_key, session_id)
if info['status'] != 204:
    print "Ping is successful!"
else:
    print "Ping failed!"
print output
Sample Perl script for SESSION (Cookie) authentication

Use the Ping resource to demonstrate SESSION (Cookie) authentication.

```perl
#!/usr/bin/perl
use strict;
use warnings;
use LWP::UserAgent;
use HTTP::Request;
use List::MoreUtils qw(firstidx);
use JSON qw( encode_json decode_json );

our $ua = LWP::UserAgent->new;
$ua->agent("ProfilerScript/0.1");
our $API_BASE = "https://127.0.0.1";

sub GET($$$)
{
    my $req = HTTP::Request->new(GET => $API_BASE . shift);
    $req->header('Accept' => 'application/json');
    my $session_key = shift;
    my $session_id = shift;
    $req->header('Cookie' => "$session_key=$session_id");
    my $res = $ua->request($req);
    return {
        code    => $res->code,
        status  => $res->status_line,
        headers => $res->headers(),
        data    => $res->content
    };
}

sub POST($$)
{
    my $req = HTTP::Request->new(POST => $API_BASE . shift);
    $req->content_type('application/json');
    $req->content(encode_json(shift));
    $req->header('Accept' => 'application/json');
    $res = $ua->request($req);
    return {
        code    => $res->code,
        status  => $res->status_line,
        headers => $res->headers(),
        data    => $res->content
    };
}

# Post to create session id
my $login_data = {
    username => 'admin',
    password => 'admin',
    purpose  => 'demonstrate SESSION authentication'};
my $response = POST('/api/common/1.0/login', $login_data);
die "Login Failed.\n$response->{data}" unless $response->{code} == 200;

my $data = decode_json($response->{data});
my $session_key = $data->{session_key};
my $session_id = $data->{session_id};
print "Login successful, $session_key=$session_id\n";

# Ping to test session authentication
$response = GET('/api/profiler/1.0/ping', $session_key, $session_id);
if ($response->{code} == 204) {
    print "Ping is successful!\n";
} else {
    print "Ping failed!\n";
    print $response->{data};
}
```
Use the Ping resource to demonstrate SESSION (Cookie) authentication.

```csharp
using System;
using System.Collections.Generic;
using System.Net;
using System.Text;
using System.IO;
using System.Net.Security;

namespace CascadeRestClient
{
    public class AuthResult
    {
        public string session_key { get; set; }
        public string session_id { get; set; }
    }

    class Program
    {
        // callback used to validate the self-gen certificate in an SSL conversation
        private static bool ValidateRemoteCertificate(object sender, X509Certificate cert, X509Chain chain, SslPolicyErrors policyErrors)
        {
            return true;
        }

        static void Main(string[] args)
        {
            if (args.Length == 0 || string.IsNullOrWhiteSpace(args[0]))
            {
                Console.WriteLine("Usage: CascadeRestClient hostname");
                return;
            }
            try
            {
                //Code to allow run with self-signed certificates
                ServicePointManager.ServerCertificateValidationCallback += new RemoteCertificateValidationCallback(ValidateRemoteCertificate);
                //Starting to run rest
                string rootUrl = "https://" + args[0];
                string requestUrl = rootUrl + "/api/common/1.0/login.json";
                var jsondata = new
                {
                    username = "admin",
                    password = "admin",
                    purpose = "demonstrate SESSION authentication"
                };
                //Serialize anonymous type to json
                JavaScriptSerializer serializer = new JavaScriptSerializer();
                string postData = serializer.Serialize(jsondata);
                // Login
                AuthResult r;
                using (var response = MakeRequest(requestUrl, WebRequestMethods.Http.Post, null, postData))
                {
                    if (response.StatusCode != HttpStatusCode.OK)
                    {
                        Console.WriteLine("Login Failed!");
                        LogResponse(response);
                        return;
                    }
                    r = ReadResponse<AuthResult>(response);
                }
                Console.WriteLine(string.Format("Login successful, {0}={1}", r.session_key, r.session_id));
                // Ping to test session authentication
                requestUrl = rootUrl + "/api/profiler/1.0/ping";
                Console.WriteLine("GET " + requestUrl);
                using (var response = MakeRequest(requestUrl, WebRequestMethods.Http.Get, string.Format("Cookie: {0}={1}", r.session_key, r.session_id)))
                {
                    if (response.StatusCode == HttpStatusCode.NoContent)
                    {
                        Console.WriteLine("Ping is successful!");
                    }
                    else
                    {
                        Console.WriteLine("Ping failed!");
                        LogResponse(response);
                    }
                }
            }
        }
    }
}
```
OAuth 2.0 authentication

In order to use the OAuth 2.0 authentication an access code needs to be generated. To generate the code:

- Go to the Configuration => Account management => OAuth Access page.
- Click the "Generate new" button.
- Enter a description for the access code. The description is used for auditing purposes.
- The system generates an access code. Use this in your script.

All access to protected resources requires a valid access token. To get an access token, the client must send a POST request with the access code. The server will issue an access token that is valid for the next 1 hour and return it in the body of the POST. If the client script runs for more than 1 hour, then it will need to generate another access token when the one that it has expires. An expired token results into an error with HTTP code 401 and error_id AUTH_EXPIRED_TOKEN.

Client request to generate an OAuth 2.0 access token:
POST /api/common/1.0/oauth/token
Host: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Accept: application/json

grant_type=access_code&assertion={access code here}

Server response:

HTTP/1.1 200 OK
Content-Type: application/json

{  "access_token": "ewoJIm5vbmNlIjogImY0MmJhZmliLAojImF1ZCI6ICJod..."  
"token_type": "bearer",  
"expires_in": 3600  
}

Client request to protected resource using the OAuth 2.0 access token:

POST /api/profiler/1.0/ping
Host: 127.0.0.1
Accept: application/json
Authorization: Bearer ewoJIm5vbmNlIjogImY0MmJhZmliLAojImF1ZCI6ICJod...

Server response:

HTTP/1.1 204 OK

Client request to protected resource using expired OAuth 2.0 access token:

POST /api/profiler/1.0/ping
Host: 127.0.0.1
Accept: application/json
Authorization: Bearer ewoJIm5vbmNlIjogImY0MmJhZmliLAojImF1ZCI6ICJod...

Server response:

HTTP/1.1 401 AUTH_EXPIRED_TOKEN
Content-Type: application/json

{  "error_id": "AUTH_EXPIRED_TOKEN",  
"error_text": "OAuth access token is expired"  
}

Sample PHP script for OAuth 2.0 authentication

In order to use the OAuth 2.0 authentication an access code needs to be generated. To generate the code:

- Go to the Configuration => Account management => OAuth Access page.
- Click the "Generate new" button.
- Enter a description for the access code. The description is used for auditing purposes.
- The system generates an access code. Use this in your script.

All access to protected resources requires a valid access token. To get an access token, the client must send a POST request with the access code. The server will issue an access token that is valid for the next 1 hour and return it in the body of the POST. If the client script runs for more than 1 hour, then it will need to generate another access token when the one that it has expires. An expired token results into an error with HTTP code 401 and error_id AUTH_EXPIRED_TOKEN.

```php
<?php
define('OAUTH_CODE', 'ewoJIm5vbmNlIjogImFmNTBlOTkxIiwKCSJhdWQiOiAiaHR0cHM6Ly9kZXNvLTEvYXBpL2NvbW1vbi8xLjAvb2F1dGgvdGvdG9rZW4iLAEjImlzcyI6ICJodHRwczovL2Rlc28tMSIsCgkicHJuIjogImFkbWluIiwKCSJqdGkiOiAiMSIsCgkiZXhwIjogIjEzNTY1NTM5NDkiLAoJImlhdCI6ICIxMzUzOTYxOTQ5Igp9);

define('HOST', '127.0.0.1'); // IP address of Profiler

// HTTP POST
function do_POST($url, $string, &$info) {
    $curl = curl_init();
    curl_setopt($curl, CURLOPT_URL, $url);
    curl_setopt($curl, CURLOPT_SSLVERSION,3);
    curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, 2);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_HEADER, true);
    curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type: application/x-www-form-urlencoded', 'Accept: application/json'));
    curl_setopt($curl, CURLOPT_POST, 1);
}
```
In order to use the OAuth 2.0 authentication an access code needs to be generated. To generate the code:

- Go to the Configuration => Account management => OAuth Access page.
- Click the "Generate new" button.
- Enter a description for the access code. The description is used for auditing purposes.
- The system generates an access code. Use this in your script.

All access to protected resources requires a valid access token. To get an access token, the client must send a POST request with the access code. The server will issue an access token that is valid for the next hour and return it in the body of the POST. If the client script runs for more than 1 hour, then it will need to generate another access token when the one that it has expires. An expired token results into an error with HTTP code 401 and error_id AUTH_EXPIRED_TOKEN.
from urlparse import urlparse
import base64
import logging
import httplib
import json
import time
import sys

OAUTH_CODE =
    'ewoJIm5vbmNlIjogImFmNTBlOTkxIiwKCSJhdWQiOiAiaHR0cHM6Ly9kZXNvLTEvYXBpL2NvbW1vbi8lJAvb2F1dGgvdG9rZW4iLAoJImlzcyI6ICJodHRwczovL2Rlc28tMSIsCgkiZXhwIjogIjEzNTY1NTM5NDkiLAoJImlhdCI6ICIxMzUzOTYxOTQ5Igp9'

HOST = '127.0.0.1'

# Lib functions

def do_POST(url, string):
    '''HTTP POST'''
    conn = httplib.HTTPSConnection(HOST, 443)
    headers = {
        "Content-Length" : str(len(string)),
        "Content-Type"   : "application/x-www-form-urlencoded",
        "Accept"         : "application/json",
    }
    conn.request('POST', url, body=string, headers=headers)
    res = conn.getresponse()
    info = {
        "status" : res.status,
        "headers" : res.getheaders(),
    }
    data = res.read()
    conn.close()
    return data, info

def do_GET(url, access_token):
    '''HTTP GET'''
    conn = httplib.HTTPSConnection(HOST, 443)
    headers = {
        "Content-Length" : 0,
        "Content-Type"   : "application/json",
        "Accept"         : "application/json",
        "Authorization"  : "Bearer %s % access_token"
    }
    conn.request('GET', url, body='', headers=headers)
    res = conn.getresponse()
    info = {
        "status" : res.status,
        "headers" : res.getheaders(),
    }
    data = res.read()
    conn.close()
    return data, info

# Post to get access token based on the access code
url = "https://%s/api/common/1.0/oauth/token" % HOST
output, info = do_POST(url, "grant_type=access_code&assertion=%s" % OAUTH_CODE)
if info['status'] is not 200:
    print "Post to get access token failed!"
    print output
    sys.exit(1)
data = json.loads(output)
access_token = data['access_token']
expires_in = data['expires_in']

print "Post to get token id is successful"
print "Token: %s % access_token"
print "The token will expire in %s seconds" % expires_in

# Ping to test OAuth 2.0 authentication
url = "https://%s/api/profiler/1.0/ping" % HOST
output, info = do_GET(url, access_token)
if info['status'] is 204:
    print "OAuth 2.0 authentication is successful!"
else:
    print "OAuth 2.0 authentication failed!"
    print output
Sample Perl script for OAuth 2.0 authentication

In order to use the OAuth 2.0 authentication an access code needs to be generated. To generate the code:

- Go to the Configuration => Account management => OAuth Access page.
- Click the "Generate new" button.
- Enter a description for the access code. The description is used for auditing purposes.
- The system generates an access code. Use this in your script.

All access to protected resources requires a valid access token. To get an access token, the client must send a POST request with the access code. The server will issue an access token that is valid for the next 1 hour and return it in the body of the POST. If the client script runs for more than 1 hour, then it will need to generate another access token when the one that it has expires. An expired token results into an error with HTTP code 401 and error_id AUTH_EXPIRED_TOKEN.
#!/usr/bin/perl
use strict;
use warnings;
use LWP::UserAgent;
use HTTP::Request;
use List::MoreUtils qw(firstidx);
use JSON qw( encode_json decode_json );
our $ua = LWP::UserAgent->new;
$ua->agent("ProfilerScript/0.1");

our $OAUTH_CODE =  "ewoJIm5vbmNlIjogImFmNTBlOTkxIiwKCSJhdWQiOiAiaHR0cHM6Ly9kZXNvLTEvYXBpL2NvbW1vbi8xLjAvb2F1dGgvdG9rZW4iLAoJImlzcyI6ICJodHRwczovL2Rlc28tMSIsCgkiZXhwIjogIjEzNTY1NTM5NDkiLAoJImlhdCI6ICIxMzUzOTYxOTQ5Igp9"
;
our $API_BASE = "https://127.0.0.1";

sub GET($$) {
    my $req = HTTP::Request->new(GET => $API_BASE . shift);
    $req->header('Accept' => 'application/json');
    my $access_token = shift;
    $req->header('Authorization' => "Bearer $access_token");
    my $res = $ua->request($req);
    return {
        code    => $res->code,
        status  => $res->status_line,
        headers => $res->headers(),
        data    => $res->content
    };
}

sub POST($$) {
    my $req = HTTP::Request->new(POST => $API_BASE . shift);
    $req->content_type('application/json');
    $req->content(shift);
    $req->header('Accept' => 'application/json');
    $req->header('Content-Type' => 'application/x-www-form-urlencoded');
    my $res = $ua->request($req);
    return {
        code    => $res->code,
        status  => $res->status_line,
        headers => $res->headers(),
        data    => $res->content
    };
}

# Post to get access token based on the access code
my $url = '/api/common/1.0/oauth/token';
my $response = POST($url, "grant_type=access_code&assertion=$OAUTH_CODE");
die "Post to get access token failed\n$response->{data}" unless $response->{code} == 200;
my $data = decode_json($response->{data});
my $access_token = $data->{access_token};
my $expires_in = $data->{expires_in};
print "Post to get token id is successful\nToken: $access_token\nThe token will expire in $expires_in seconds"
;

# Ping to test OAuth 2.0 authentication
$response = GET("/api/profiler/1.0/ping", $access_token);
if ($response->{code} == 204) {
    print "OAuth 2.0 authentication is successful\n";
} else {
    print "OAuth 2.0 authentication failed\n";
    print $response->{data};
}

Sample .Net/C# script for OAuth 2.0 authentication
In order to use the OAuth 2.0 authentication an access code needs to be generated. To generate the code:
Go to the Configuration => Account management => OAuth Access page.

Click the "Generate new" button.

Enter a description for the access code. The description is used for auditing purposes.

The system generates an access code. Use this in your script.

All access to protected resources requires a valid access token. To get an access token, the client must send a POST request with the access code. The server will issue an access token that is valid for the next 1 hour and return it in the body of the POST. If the client script runs for more than 1 hour, then it will need to generate another access token when the one that it has expires. An expired token results into an error with HTTP code 401 and error_id AUTH_EXPIRED_TOKEN.

```csharp
using System;
using System.Collections.Generic;
using System.Net;
using System.Text;
using System.IO;
using System.Net.Security;
namespace CascadeRestClient
{
    public class OAuthResult
    {
        public string access_token { get; set; }
        public string expires_in { get; set; }
    }
    class Program
    {
        // callback used to validate the self-gen certificate in an SSL conversation
        private static bool ValidateRemoteCertificate(object sender, X509Certificate cert, X509Chain chain, SslPolicyErrors policyErrors)
        {
            return true;
        }
        static void Main(string[] args)
        {
            if (args.Length == 0 || string.IsNullOrWhiteSpace(args[0]))
            {
                Console.WriteLine("Usage: CascadeRestClient hostname");
                return;
            }
            try
            {
                // Code to allow run with self-signed certificates
                // validate cert by calling a function
                ServicePointManager.ServerCertificateValidationCallback += new RemoteCertificateValidationCallback(ValidateRemoteCertificate);
                // Starting to run rest
                string rootUrl = "https://" + args[0];
                string OAUTH_CODE = "ewoJIm5vbmNlIjogIjUyZGFhMzZjIiwKCSJhdWQiOiAiaHR0cHM6Ly9jc2MtcGVyY29tbW9uLmdvbXBlcy5jb20ifQ==";
                string requestUrl = rootUrl + "/api/common/1.0/oauth/token";
                string postData = "grant_type=access_code&assertion=" + OAUTH_CODE;
                OAuthResult r;
                // Login
                {
                    if (response.StatusCode != HttpStatusCode.OK)
                    {
                        Console.WriteLine("Login Failed!");
                        LogResponse(response);
                        Console.Read();
                        return;
                    }
                    r = ReadResponse<OAuthResult>(response);
                }
                Console.WriteLine("Post to get token id is successful
Token:" + r.access_token);
                Console.WriteLine(string.Format("The token will expire in {0} seconds", r.expires_in));
                // Ping to test session authentication
                requestUrl = rootUrl + "/api/profiler/1.0/ping";
                Console.WriteLine("GET " + requestUrl);
                {
                    if (response.StatusCode == HttpStatusCode.NoContent)
                    {
                        Console.WriteLine("OAuth 2.0 authentication is successful!");
                    }
                }
            }
        }
    }
}
Console.WriteLine("OAuth 2.0 authentication is successful!");

catch (Exception e)
{
    Console.WriteLine(e.Message);
}

Console.ReadKey();

private static void LogResponse(HttpWebResponse response)
{
    using (Stream stream = response.GetResponseStream())
    {
        using (StreamReader reader = new StreamReader(stream, Encoding.UTF8))
        {
            string responseString = reader.ReadToEnd();
            Console.WriteLine(responseString);
        }
    }
}

private static T ReadResponse<T>(HttpWebResponse response) where T : class
{
    DataContractJsonSerializer jsonSerializer = new DataContractJsonSerializer(typeof(T));
    object objResponse = jsonSerializer.ReadObject(response.GetResponseStream());
    return objResponse as T;
}

/// <summary>
/// Make request
/// </summary>
/// <param name="requestUrl">url for request</param>
/// <param name="action">Http Verb, Get, Post etc</param>
/// <param name="requestData">Data posted</param>
/// <returns></returns>
private static HttpWebResponse MakeRequest(string requestUrl, string action, string contenttype, string header, string requestData = null)
{
    HttpWebRequest request = WebRequest.Create(requestUrl) as HttpWebRequest;
    try
    {
        if (!string.IsNullOrWhiteSpace(header))
            request.Headers.Add(header);
        request.ContentType = contenttype;
        request.Accept = "application/json";
        request.Method = action;
        if (requestData == null)
        {
            request.ContentLength = 0;
        }
        else
        {
            ASCIIEncoding encoding = new ASCIIEncoding();
            byte[] byte1 = encoding.GetBytes(requestData);
            request.ContentLength = byte1.Length;
            using (Stream newStream = request.GetRequestStream())
            {
                newStream.Write(byte1, 0, byte1.Length);
            }
        }
    }
    var response = request.GetResponse() as HttpWebResponse;

    try
    {
        if (!string.IsNullOrWhiteSpace(header))
            request.Headers.Add(header);
        request.ContentType = contenttype;
        request.Accept = "application/json";
        request.Method = action;
        if (requestData == null)
        {
            request.ContentLength = 0;
        }
        else
        {
            ASCIIEncoding encoding = new ASCIIEncoding();
            byte[] byte1 = encoding.GetBytes(requestData);
            request.ContentLength = byte1.Length;
            using (Stream newStream = request.GetRequestStream())
            {
                newStream.Write(byte1, 0, byte1.Length);
            }
        }
    }
    catch (Exception)
    {
        request.Abort();
        throw;
    }
}

Resources
Services: List services
Get information for supported API namespaces.

GET https://{device}/api/common/1.0/services

Authorization
This request requires authorization.

Response Body
On success, the server returns a response body with the following structure:

```json
{
  "services": [
    {
      "id": "common",
      "versions": [
        "1.0"
      ]
    },
    {
      "id": "profiler",
      "versions": [
        "1.0"
      ]
    }
  ]
}
```

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>services</td>
<td>&lt;array of &lt;object&gt;&gt;</td>
<td>List of namespaces and versions supported by the system.</td>
<td></td>
</tr>
<tr>
<td>services[service]</td>
<td>&lt;object&gt;</td>
<td>Object representing an API service.</td>
<td></td>
</tr>
<tr>
<td>services[service].id</td>
<td>&lt;string&gt;</td>
<td>ID of the service, such as profiler.</td>
<td></td>
</tr>
<tr>
<td>services[service].versions</td>
<td>&lt;array of &lt;string&gt;&gt;</td>
<td>List of versions for a given service.</td>
<td></td>
</tr>
<tr>
<td>services[service].versions[version]</td>
<td>&lt;string&gt;</td>
<td>Version of the service.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Auth_info: Get authentication info
Get information for supported authentication methods.

GET https://{device}/api/common/1.0/auth_info

Authorization
This request requires authorization.

Response Body
On success, the server returns a response body with the following structure:

```json
...
```
```
{
  "login_banner": string,
  "specify_purpose": boolean,
  "supported_methods": [
    string
  ]
}
```

Example:
```
{
  "supported_methods": [
    "BASIC",
    "COOKIE",
    "OAUTH_2_0"
  ],
  "specify_purpose": true,
  "login_banner": "A free-form text string that should be displayed to the user prior to logging in"
}
```

```
{
  "password": string,
  "purpose": string,
  "username": string
}
```

Example:
```
{
  "username": "username",
  "password": "password",
  "purpose": "Purpose for logging in"
}
```

### Login: Login
Start cookie based authentication session.

**POST** https://{device}/api/common/1.0/login?username={string}&password={string}&purpose={string}

### Authorization
This request requires authorization.

### Parameters

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>&lt;string&gt;</td>
<td>User name that identifies the user to the system.</td>
<td>Optional</td>
</tr>
<tr>
<td>password</td>
<td>&lt;string&gt;</td>
<td>User account password.</td>
<td>Optional</td>
</tr>
<tr>
<td>purpose</td>
<td>&lt;string&gt;</td>
<td>Login purpose describing why the user logs in to the system.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Request Body
Provide a request body with the following structure:

**JSON**
```
{
  "password": string,
  "purpose": string,
  "username": string
}
```

Example:
```
{
  "username": "username",
  "password": "password",
  "purpose": "Purpose for logging in"
}
```

### Response Body
On success, the server returns a response body with the following structure:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>&lt;object&gt;</td>
<td>Specification for request to log in.</td>
<td></td>
</tr>
<tr>
<td>login.password</td>
<td>&lt;string&gt;</td>
<td>Password.</td>
<td></td>
</tr>
<tr>
<td>login.purpose</td>
<td>&lt;string&gt;</td>
<td>Login purpose describing why the user logs in to the system.</td>
<td>Optional</td>
</tr>
<tr>
<td>login.username</td>
<td>&lt;string&gt;</td>
<td>User name that identifies the user to the system.</td>
<td></td>
</tr>
</tbody>
</table>
```json
{
    "session_id": string,
    "session_key": string
}
```

Example:
```json
{
    "session_key": "SESSID",
    "session_id": "164c38b379c7cff47fa8503a28743e5c8648d97dcbd0154f597673d9031a1c63"
}
```

---

**Oauth: Get OAuth token**

Get a OAuth token based on OAuth code.

**POST** `https://{device}/api/common/1.0/oauth/token?grant_type={string}&assertion={string}&state={string}`

**Authorization**

This request requires authorization.

**Parameters**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>grant_type</td>
<td>&lt;string&gt;</td>
<td>The type of authorization method used to grant this token.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value must be <code>access_code</code>.</td>
<td></td>
</tr>
<tr>
<td>assertion</td>
<td>&lt;string&gt;</td>
<td>The access code generated by the system on the OAuth Access page.</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>&lt;string&gt;</td>
<td>Optional client-provided value that will be echoed back in the response.</td>
<td></td>
</tr>
</tbody>
</table>

**Request Body**

Do not provide a request body.

**Response Body**

On success, the server returns a response body with the following structure:

```json
{
    "access_token": string,
    "allowed_signature_types": [
        string
    ],
    "expires_in": number,
    "state": string,
    "token_type": string
}
```

Example:
```json
{
    "access_token": "eyJhbGciOiJub25lIn0K.ew0KICAiaXNzIjogImh0dHBzOi8vY29tIi5zcGVjaW50b3JvdW50by8yMDIyYjYxNzEwMDI0MDIyMDA4MzkwNjUyMDMxMzYyNzE5NjM4NzQ5NjY4NzY4IiwNCiAgImF1dGlvbnMiOltdDQp9DQo=",
    "token_type": "bearer",
    "allowed_signature_types": ["none"],
    "expires_in": 3600,
    "state": "a34rfFas"
}
```

---

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>token_response</td>
<td>&lt;object&gt;</td>
<td>TokenResponse object</td>
<td></td>
</tr>
<tr>
<td>token_response.access_token</td>
<td>&lt;string&gt;</td>
<td>The generated access token that can be used to access protected resources.</td>
<td></td>
</tr>
<tr>
<td>token_response.allowed_signature_types</td>
<td>&lt;array of &lt;string&gt;&gt;</td>
<td>Array of allowed signature methods.</td>
<td></td>
</tr>
</tbody>
</table>
Oauth: Get Oauth code/implicit token
Get Oauth code/implicit token for the current user.

Authorization
This request requires authorization.

Parameters

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>&lt;string&gt;</td>
<td>Client identifier.</td>
<td>Optional</td>
</tr>
<tr>
<td>response_type</td>
<td>&lt;string&gt;</td>
<td>The value must be 'code' for requesting an access code and 'token' for an access token.</td>
<td></td>
</tr>
<tr>
<td>desc</td>
<td>&lt;string&gt;</td>
<td>Description of the use of this code. Used in audit trail logs.</td>
<td>Optional</td>
</tr>
<tr>
<td>state</td>
<td>&lt;string&gt;</td>
<td>Included if the state parameter was passed in the token request.</td>
<td>Optional</td>
</tr>
<tr>
<td>redirect_uri</td>
<td>&lt;string&gt;</td>
<td>URI that will be used for redirect.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Response Body
On success, the server does not provide any body in the responses.

Ping: Ping
Simple test of service availability.

Authorization
This request requires authorization.

Response Body
On success, the server does not provide any body in the responses.

Logout: Logout
End cookie based authentication session.

Authorization
This request requires authorization.

Parameters

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>banner_disagree</td>
<td>&lt;string&gt;</td>
<td>Used when the session is being ended due to the user not agreeing to the login banner conditions.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Request Body
Do not provide a request body.

Response Body
On success, the server does not provide any body in the responses.

Info: Get info
Get appliance info.
Authorization
This request requires authorization.

Response Body
On success, the server returns a response body with the following structure:

```json
{
  "device_name": string,
  "hw_version": string,
  "mgmt_addresses": [
    string
  ],
  "model": string,
  "serial": string,
  "sw_version": string
}
```

Example:

```json
{
  "sw_version": "10.0 (release 20121106_2000)",
  "device_name": "cam-redfin2",
  "mgmt_addresses": [
    "10.38.9.106"
  ],
  "serial": "FB8RS00035E98",
  "model": "02260"
}
```

GET https://{device}/api/common/1.0/info

Authorization
This request requires authorization.

Response Body
On success, the server returns a response body with the following structure:

```json
{
  "device_name": string,
  "hw_version": string,
  "mgmt_addresses": [
    string
  ],
  "model": string,
  "serial": string,
  "sw_version": string
}
```

Example:

```json
{
  "sw_version": "10.0 (release 20121106_2000)",
  "device_name": "cam-redfin2",
  "mgmt_addresses": [
    "10.38.9.106"
  ],
  "serial": "FB8RS00035E98",
  "model": "02260"
}
```

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>&lt;object&gt;</td>
<td>Information about the system.</td>
<td></td>
</tr>
<tr>
<td>info.device_name</td>
<td>&lt;string&gt;</td>
<td>Name of the device that the API is running on.</td>
<td>Optional</td>
</tr>
<tr>
<td>info.hw_version</td>
<td>&lt;string&gt;</td>
<td>Unsupported.</td>
<td>Optional</td>
</tr>
<tr>
<td>info.mgmt_addresses</td>
<td>&lt;array of &lt;string&gt;&gt;</td>
<td>List of IP addresses.</td>
<td></td>
</tr>
<tr>
<td>info.mgmt_addresses[ip]</td>
<td>&lt;string&gt;</td>
<td>IP address.</td>
<td>Optional</td>
</tr>
<tr>
<td>info.model</td>
<td>&lt;string&gt;</td>
<td>Model of the device.</td>
<td>Optional</td>
</tr>
<tr>
<td>info.serial</td>
<td>&lt;string&gt;</td>
<td>Serial number of the device.</td>
<td>Optional</td>
</tr>
<tr>
<td>info.sw_version</td>
<td>&lt;string&gt;</td>
<td>Version of the software that is running on the device.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Error Codes

In the event that an error occurs while processing a request, the server will respond with appropriate HTTP status code and additional information in the response body:

```json
{
  "error_id": "{error identifier}"
  "error_text": "{error description}"
  "error_info": {error specific data structure, optional}
}
```

The table below lists the possible errors and the associated HTTP status codes that may returned.

<table>
<thead>
<tr>
<th>Error ID</th>
<th>HTTP Status</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERNAL_ERROR</td>
<td>500</td>
<td>Internal server error.</td>
</tr>
<tr>
<td>AUTH_REQUIRED</td>
<td>401</td>
<td>The requested resource requires authentication.</td>
</tr>
<tr>
<td>AUTH_INVALID_CREDENTIALS</td>
<td>401</td>
<td>Invalid username and/or password.</td>
</tr>
<tr>
<td>AUTH_INVALID_SESSION</td>
<td>401</td>
<td>Session ID is invalid.</td>
</tr>
<tr>
<td>AUTH_EXPIRED_PASSWORD</td>
<td>403</td>
<td>The password must be changed. Access only to password change resources.</td>
</tr>
<tr>
<td>AUTH_DISABLED_ACCOUNT</td>
<td>403</td>
<td>Account is either temporarily or permanently disabled.</td>
</tr>
<tr>
<td>AUTH_FORBIDDEN</td>
<td>403</td>
<td>User is not authorized to access the requested resource.</td>
</tr>
<tr>
<td>AUTH_INVALID_TOKEN</td>
<td>401</td>
<td>OAuth access token is invalid.</td>
</tr>
<tr>
<td>AUTH_EXPIRED_TOKEN</td>
<td>401</td>
<td>OAuth access token is expired.</td>
</tr>
<tr>
<td>AUTH_INVALID_CODE</td>
<td>401</td>
<td>OAuth access code is invalid.</td>
</tr>
<tr>
<td>AUTH_EXPIRED_CODE</td>
<td>401</td>
<td>OAuth access code is expired.</td>
</tr>
<tr>
<td>RESOURCE_NOT_FOUND</td>
<td>404</td>
<td>Requested resource was not found.</td>
</tr>
<tr>
<td>HTTP_INVALID_METHOD</td>
<td>405</td>
<td>Requested method is not available for this resource.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Status Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>HTTP_INVALID_HEADER</td>
<td>400</td>
<td>An HTTP header was malformed.</td>
</tr>
<tr>
<td>REQUEST_INVALID_INPUT</td>
<td>400</td>
<td>Malformed input structure.</td>
</tr>
<tr>
<td>URI_INVALID_PARAMETER</td>
<td>400</td>
<td>URI parameter is not supported or malformed.</td>
</tr>
<tr>
<td>URI_MISSING_PARAMETER</td>
<td>400</td>
<td>Missing required parameter.</td>
</tr>
</tbody>
</table>