Common REST API. v1.1

Copyright © Riverbed Technology Inc. 2024

Created Jan 16, 2024 at 02:01 PM

Overview

Overview

The documentation pages in this section describe the RESTful APIs included with NetProfiler 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

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, 1);
  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($curl, CURLOPT_HTTPHEADER, array('Accept: application/json'));
  curl_setopt($curl, CURLOPT_HTTPGET, true);
  $output = curl_exec($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 Perl script for BASIC authentication

Use the Ping resource to demonstrate BASIC authentication.

```
#!/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 );

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

Use the Ping resource to demonstrate BASIC authentication.

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Text;
using System.IO;
using System.Net.Security;
using System.Security.Cryptography.X509Certificates;

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 beaic 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.Method = WebRequestMethods.Http.Get;
                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=bfe3c2fd7b53053eecdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

{
  "session_key": "SESSID",
  "session_id": "bfe3c2fd7b53053eecdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7"
}
```

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=bfe3c2fd7b53053eecdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

{
  "session_key": "SESSID",
  "session_id": "bfe3c2fd7b53053eecdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7"
}
```

Client request using XML input:

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

Server response:

```
HTTP/1.1 200 OK
Set-Cookie: SESSID=bfe3c2fd7b53053eecdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7

<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=bfe3c2fd7b53053eecdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7
```

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=bfe3c2fd7b53053eecdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7
```

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=bfe3c2fd7b53053eecdd54b08c01d6a8d447aa6c15ed8f7523032c5814221ee7
```

Server response:

```
HTTP/1.1 204
```

Sample PHP script for SESSION (Cookie) authentication

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

```
<?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, 1);
  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);
  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;
}

// 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, 1);
  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);
  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;
}

// 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)
```

```
$output = do_POST($url, json_encode($login_data), $info);

if ($info['http_code'] != 200) {
  echo "Login Failed!\n";
  echo $output;
  exit(1);
}

$data = json_decode($output, 1);
$session_key = $data['session_key'];
$session_id  = $data['session_id'];

echo "Login successful, {$session_key}={$session_id}\n";

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

$info = array();
$output = do_GET($url, $session_key, $session_id, $info);

if ($info['http_code'] == 204) {
  echo "Ping is successful!\n";
} else {
  echo "Ping failed!\n";
  echo $output;
}

?>
```

Sample Perl script for SESSION (Cookie) authentication

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

```
#!/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');

  my $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}\n" 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};
}
```

Sample .Net/C# script for SESSION (Cookie) authentication

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

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Net;
using System.Runtime.Serialization.Json;
using System.Text;
using System.IO;
using System.Net.Security;
using System.Security.Cryptography.X509Certificates;
using System.Web.Script.Serialization;

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
                // validate cert by calling a function
                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 anomymous 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);
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        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>
        /// <typeparam name="T">return type</typeparam>
        /// <param name="requestUrl">url for request</param>
        /// <param name="action">Http Verb, Get, Post etc</param>
        /// <param name="header">additional header except accept and content type </param>
        /// <param name="requestData">Data posted</param>
        /// <returns></returns>
        private static HttpWebResponse MakeRequest(string requestUrl, string action, string header, string requestData = null)
        {
            HttpWebRequest request = WebRequest.Create(requestUrl) as HttpWebRequest;
            if (!string.IsNullOrWhiteSpace(header))
                request.Headers.Add(header);
            request.ContentType = "application/json";
            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);
                }
            }

            return request.GetResponse() as HttpWebResponse;
        }
    }
}
```

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": "ewoJIm5vbmNlIjogImY0MmJhZmIiLAoJImF1ZCI6ICJod..."
  "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 ewoJIm5vbmNlIjogImY0MmJhZmIiLAoJImF1ZCI6ICJod...
```

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 ewoJIm5vbmNlIjogImY0MmJhZmIiLAoJImF1ZCI6ICJod...
```

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

define('OAUTH_CODE',
'ewoJIm5vbmNlIjogImFmNTBlOTkxIiwKCSJhdWQiOiAiaHR0cHM6Ly9kZXNvLTEvYXBpL2NvbW1vbi8xLjAvb2F1dGgvdG9rZW4iLAoJImlzcyI6ICJodHRwczovL2Rlc28tMSIsCgk


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, 1);
  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($curl, CURLOPT_HTTPHEADER, array('Content-Type: application/x-www-form-urlencoded',
                                              'Accept: application/json'));
  curl_setopt($curl, CURLOPT_POST,           1);
  curl_setopt($curl, CURLOPT_POSTFIELDS,     $string);
  $output = curl_exec($curl);
  $info   = curl_getinfo($curl);
  curl_close($curl);

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

// HTTP GET
function do_GET($url, $access_token, &$info) {
  $curl = curl_init();
  curl_setopt($curl, CURLOPT_URL, $url);
  curl_setopt($curl, CURLOPT_SSLVERSION, 1);
  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($curl, CURLOPT_HTTPHEADER, array('Accept: application/json',
                                               "Authorization: Bearer {$access_token}"));
  curl_setopt($curl, CURLOPT_HTTPGET, true);
  $output = curl_exec($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;
}

// Post to get access token based on the access code

$url = 'https://' . HOST . '/api/common/1.0/oauth/token';
$output = do_POST($url, 'grant_type=access_code&assertion=' . OAUTH_CODE, $info);

if ($info['http_code'] != 200) {
  echo "Post to get access token failed!\n";
  echo $output;
  exit(1);
}

$data = json_decode($output, 1);
$access_token = $data['access_token'];
$expires_in   = $data['expires_in'];
echo "Post to get token id is successful\nToken: {$access_token}\n";
echo "The token will expire in {$expires_in} seconds\n";

// Ping to test OAuth 2.0 authentication
$url = 'https://' . HOST . '/api/profiler/1.0/ping';
echo "GET {$url}\n";

$info = array();
$output = do_GET($url, $access_token, $info);

if ($info['http_code'] == 204) {
  echo "OAuth 2.0 authentication is successful!\n";
} else {
  echo "OAuth 2.0 authentication failed!\n";
  echo $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 =
"ewoJIm5vbmNlIjogImFmNTBlOTkxIiwKCSJhdWQiOiAiaHR0cHM6Ly9kZXNvLTEvYXBpL2NvbW1vbi8xLjAvb2F1dGgvdG9rZW4iLAoJImlzcyI6ICJodHRwczovL2Rlc28tMSIsCgk


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}\n"
  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\n";
print "The token will expire in $expires_in seconds\n";

# 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.

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Runtime.Serialization.Json;
using System.Text;
using System.IO;
using System.Net.Security;
using System.Security.Cryptography.X509Certificates;

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 =
"ewoJIm5vbmNlIjogIjUyZGFhMzZjIiwKCSJhdWQiOiAiaHR0cHM6Ly9jc2MtcGVyZjE3LmxhYi5uYnR0ZWNoLmNvbS9hcGkvY29tbW9uLzEuMC9vYXV0aC90b2tlbiIsCgkiaXNzIj

                string requestUrl = rootUrl + "/api/common/1.0/oauth/token";

                string postData = "grant_type=access_code&assertion=" + OAUTH_CODE;
                OAuthResult r;

                // Login
                using (var response = MakeRequest(requestUrl, WebRequestMethods.Http.Post,"application/x-www-form-urlencoded", null, postData))
                {
                    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\nToken:" + 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);

                using (var response = MakeRequest(requestUrl, WebRequestMethods.Http.Get,"application/json", string.Format("Authorization: Bearer {0}",
r.access_token)))
                {
                    if (response.StatusCode == HttpStatusCode.NoContent)
                    {
                        Console.WriteLine("OAuth 2.0 authentication is successful!");
                        Console.WriteLine("OAuth 2.0 authentication is successful!");
                    }
                    else
                    {
                        Console.WriteLine("OAuth 2.0 authentication failed!");
                        LogResponse(response);
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.Read();
        }

        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;
                return response;
            }
            catch (Exception)
            {
                request.Abort();
                throw;
            }

        }
    }
}
```

Resources

Services: List services

Get information for supported API namespaces.

```
GET https://{device}/api/common/1.1/services
```

Authorization

This request does not require authorization.

Response Body

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

JSON

```
[
  {
    "id": string,
    "versions": [
      string
    ]
  }
]

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

Property Name	Type	Description	Notes
CommonServices	*<array of <object>>*	List of namespaces and versions supported by the system.	
CommonServices[CommonService]	*<object>*	Service (namespace and version) supported by the system.	Optional
CommonServices[CommonService].id	*<string>*	ID of the service, such as profiler.	
CommonServices[CommonService].versions	*<array of <string>>*	List of versions that the service supports.	
CommonServices[CommonService].versions [item]	*<string>*	Version of the service.	Optional

Oauth: Get Oauth code/implicit token

Get Oauth code/implicit token for the current user.

```
GET https://{device}/api/common/1.1/oauth/authorize?desc={string}&lifetime={number}&response_type={string}&redirect_uri={string}&client_id={string}&state={string}
```

Authorization

This request requires authorization.

Parameters

Property Name	Type	Description	Notes
desc	*<string>*	Description of the use of this code. Used in audit trail logs.	Optional
lifetime	*<number>*	Number, representing the generated code lifetime in seconds.	Optional
response_type	*<string>*	The value must be 'code' for requesting an access code and 'token' for an access token.	
redirect_uri	*<string>*	URI that will be used for redirect.	Optional
client_id	*<string>*	Client identifier.	Optional
state	*<string>*	Included if the state parameter was passed in the token request.	Optional

Response Body

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

Oauth: Get OAuth token

Get a OAuth token based on OAuth code.

```
POST https://{device}/api/common/1.1/oauth/token?grant_type={string}&assertion={string}&state={string}
```

Authorization

This request does not require authorization.

Parameters

Property Name	Type	Description	Notes
grant_type	*<string>*	The type of authorization method used to grant this token. The value must be access_code.	
assertion	*<string>*	The access code generated by the system on the OAuth Access page.	
state	*<string>*	Optional client-provided value that will be echoed back in the response.	Optional

Request Body

Do not provide a request body.

Response Body

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

JSON

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

Example:
{
  "access_token":
"eyJhbGciOiJub25lIn0K.ew0KICAiaXNzIjogImh0dHBzOi8vcHJvZmlsZXIubGFiLm5idHRlY2guY29tIiwNCiAgImp0aSI6ICI3MjM1IiwNCiAgInBybiI6

  "token_type": "bearer",
  "allowed_signature_types": [
    "none"
  ],
  "expires_in": 3600,
  "state": "a34rfFas"
}
```

Property Name	Type	Description	Notes
TokenResponse	*<object>*	TokenResponse object.	
TokenResponse.allowed_signature_types	*<array of <string>>*	Array of allowed signature methods.	Optional
TokenResponse.allowed_signature_types [item]	*<string>*	Allowed signature method.	Optional
TokenResponse.token_type	*<string>*	The token type. Only "bearer" is currently supported.	
TokenResponse.access_token	*<string>*	The generated access token that can be used to access protected resources.	
TokenResponse.expires_in	*<number>*	How long this token is valid for.	
TokenResponse.state	*<string>*	Included if the state parameter was passed in the token request.	Optional

Ping: Ping

Simple test of service availability.

```
GET https://{device}/api/common/1.1/ping
```

Authorization

This request does not require authorization.

Response Body

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

Login: Login

Start cookie based authentication session.

```
POST https://{device}/api/common/1.1/login?password={string}&purpose={string}&username={string}
```

Authorization

This request does not require authorization.

Parameters

Property Name	Type	Description	Notes
password	*<string>*	User account password.	Optional
purpose	*<string>*	Login purpose describing why the user logs in to the system.	Optional
username	*<string>*	User name that identifies the user to the system.	Optional

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 loging in"
}
```

Property Name	Type	Description	Notes
LoginRequest	*<object>*	Specification for request to log in.	
*LoginRequest.*password	*<string>*	Password.	
*LoginRequest.*purpose	*<string>*	Login purpose describing why the user logs in to the system.	Optional
*LoginRequest.*username	*<string>*	User name that identifies the user to the system.	

Response Body

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

JSON

```
{
  "session_key": string,
  "session_id": string
}

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

Property Name	Type	Description	Notes
LoginResponse	*<object>*	Response for login request.	
*LoginResponse.*session_key	*<string>*	Name of the session cookie that should be stored by the client and presented on subsequent requests.	
*LoginResponse.*session_id	*<string>*	Value of the session cookie.	

Auth_Info: Get authentication info

Get information for supported authentication methods.

```
GET https://{device}/api/common/1.1/auth_info
```

Authorization

This request does not require authorization.

Response Body

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

JSON

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

Property Name	Type	Description	Notes
CommonAuthInfo	*<object>*	Common authentication information.	
*CommonAuthInfo.*specify_purpose	*<string>*	Flag describing if the login purpose should be specified.	
*CommonAuthInfo.*supported_methods	*<array of <string>>*	List of supported authentication methods.	
*CommonAuthInfo.*supported_methods[item]	*<string>*	One method from the list of supported authentication methods.	Optional; Values: BASIC, COOKIE, OAUTH_2_0
*CommonAuthInfo.*login_banner	*<string>*	Login banner that a client should display before user's login.	

Logout: Quick logout

Deprecated. End cookie based authentication session.

```
GET https://{device}/api/common/1.1/logout?banner_disagree={string}
```

Authorization

This request requires authorization.

Parameters

Property Name	Type	Description	Notes
banner_disagree	*<string>*	Used when the session is being ended due to the user not agreeing to the login banner conditions.	Optional

Response Body

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

Logout: Logout

End cookie based authentication session.

```
POST https://{device}/api/common/1.1/logout?banner_disagree={string}
```

Authorization

This request requires authorization.

Parameters

Property Name	Type	Description	Notes
banner_disagree	*<string>*	Used when the session is being ended due to the user not agreeing to the login banner conditions.	Optional

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.

```
GET https://{device}/api/common/1.1/info
```

Authorization

This request does not require authorization.

Response Body

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

JSON

```
{
  "hw_version": string,
  "dispatcher_addresses": [
    string
  ],
  "serial": string,
  "sw_version": string,
  "model": string,
  "scm_id": string,
  "mgmt_addresses": [
    string
  ],
  "type": string,
  "sw_build": string,
  "device_name": string
}
```

Property Name	Type	Description	Notes
CommonInfo_1_1	*<object>*	Information about the system.	
CommonInfo_1_1.hw_version	*<string>*	Unsupported.	Optional
CommonInfo_1_1.dispatcher_addresses	*<array of <string>>*	List of dispatcher IP address that can be used to receive flows.	Optional
CommonInfo_1_1.dispatcher_addresses [item]	*<string>*	IP address.	Optional
CommonInfo_1_1.serial	*<string>*	Serial number of the device.	Optional
CommonInfo_1_1.sw_version	*<string>*	Version of the software that is running on the device.	Optional
CommonInfo_1_1.model	*<string>*	Model of the device.	Optional
CommonInfo_1_1.scm_id	*<string>*	ID of SCM manager.	Optional
CommonInfo_1_1.mgmt_addresses	*<array of <string>>*	List of management IP address that can be used to access the system.	
CommonInfo_1_1.mgmt_addresses[item]	*<string>*	IP address.	Optional
CommonInfo_1_1.type	*<string>*	Device type (NetProfiler, Gateway).	
CommonInfo_1_1.sw_build	*<string>*	Build number of the software that is running on the device.	Optional
CommonInfo_1_1.device_name	*<string>*	Name of the device that the API is running on.	Optional

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:

```
{
    "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.

Error ID	HTTP Status	Comments

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