INSURANCE TECHNOLOGIES

API SECURITY SPECIFICATIONS

FIRELIGHT

FireLight®

Electronic Applications

API SECURITY SPECIFICATIONS

Document Version: 1

Published: March 16, 2018

Insurance Technologies, LLC

Insurance Technologies, LLC
Two South Cascade Avenue
Colorado Springs, CO 80903
USA
Phone: 719.442.6400
FAX: 719.442.0600
Internet E-Mail: info@insurancetechnologies.com
Website: http://www.insurancetechnologies.com

Table of Contents

Overview

The FireLight® rest-based API services utilize JSON web tokens for security and access. Please see the documentation for each service in order to determine any differences in security.

Accessing a FireLight API service endpoint will require a JWT token first received from the FireLight token endpoint. Getting access to a service endpoint is a two-step process:

1. A call to the FireLight Token endpoint to get a token. Note this will require credentials pre-defined for your organization. The token will be returned in an encoded string format.
2. Embedding the received encoded token string into the subsequent service endpoint call's "Authorization" header.
    a. FireLight API tokens may be used multiple times within their time limit window. A FireLight API token expires after 15 minutes, after which, you will need to get a new one.

For more information on JSON web tokens, see the open standard documentation at https://tools.ietf.org/html/rfc7519.

Environment Endpoints

The below endpoints are for obtaining FireLight API tokens. API service endpoints will be specified in their respective documents.

Note that the below token endpoints require POST requests only, as getting a token requires credential details carried in the body of the request.

Environment	Endpoint
External QE	https://firelight.insurancetechnologies.com/EGApp/api/Security/GetToken
UAT	https://uat.firelighteapp.com/EGApp/api/Security/GetToken
Staging	https://staging.firelighteapp.com/EGApp/api/Security/GetToken
Production	https://www.firelighteapp.com/EGApp/api/Security/GetToken

Pre-Configuring Your Credentials

Before being able to accomplish the first step of the process, your organization will need to configure specific security options via the FireLight Admin tool. Please contact Insurance Technologies if you do not have access to the FireLight Admin tool.

Secret

Part of the formation of a request to the token endpoint will be a SHA256 hash generated off a secret, placed in the request as a hex-formatted string.

In the Admin tool, on the Profiles->Organizations page, a read-only text field called “Web Service Token Secret” will contain the secret for the request originator’s organization. If it is blank, a new secret will need to be generated, which can be done at any time by clicking the “New” button next to the field. Note that the generated secrets are randomly generated GUID/UUID strings.

Certificate

A request to the token endpoint will additionally require a digital signature made with the private key of a certificate. This certificate will be owned and controlled by your organization, and FireLight will need the exported public key of that certificate in order to verify the signature on your token requests.

In the FireLight Admin tool, on the Profiles->Organizations page, digital certificates can be placed in the “Web Service Token Certificate” pop-up text field.

The digital signature is to be generated using the private key of the configured certificate, using either a SHA1 or SHA256 algorithm. Other algorithms are not supported.

Organization Acronym

Insurance Technologies will assign your organization with a specific acronym for FireLight. Please contact Insurance Technologies for this acronym as needed.

Getting the Token

The first step of the process is to receive the token. Multiple requirements regarding headers and the body content are listed below, as well as the process to do so.

Required Headers

The following HTTP headers are required to be on the request to the token endpoint:

Header Name	Expected Value	Description
Method	POST	Method of the request.
ContentType	text/plain	Media Type of the request.
ContentLength	Integer Variable	Length of the content contained within the body of the request.
Date	Date-Formatted Variable	Date/time of the request. Must be formatted according to common specification, including UTC/GMT requirement. Note this item additionally gets used to verify the signature, as defined in the 'Generating the Signature' section below.

Body Content

The body content will be formatted in a plain-text string with a specific format. If you are planning to call out to an API service endpoint that needs a user, you must include an ACORD-formatted 1228 xml in the request for a token. The 1228 xml specifications are outlined in the *FireLight Single Sign On Methods* document.

If a user is not required, you may exclude that part from the body string, as shown below:

- With user 1228:
  - "grant_type=" + **grant_type** + "id=" + **org** + "&secret=" + **hash** + "&sig=" + **signed64** + "&xml=" + **user1228**
- Without user 1228:
  - "grant_type=" + **grant_type** + "id=" + **org** + "&secret=" + **hash** + "&sig=" + **signed64**

The variable details are detailed in the following table:

Variable Name	Expected Value	Description
grant_type	hashsig	This unchanging value must be placed on every request. It indicates the type of security mechanism to be used.
org	String Variable	This is the unique & unchanging organization acronym attributed to each FireLight client organization. Contact Insurance Technologies for this variable.
hash	Hex String Variable	This hexed string value is generated from the request originator's configured secret, unique to each environment. See the 'Generating the Hash' section for more details.
signed64	Url-Encoded Base64 String Variable	This base64 string value is generated from the request originator's cert that is configured in the FireLight Admin tool. It must be url-encoded (after being base64-encoded) before placed into the post body. See the 'Generating the Signature' section for more details.
user1228		This is a conditional ACORD 1228 xml containing the details of the user to be applied to the token, as defined in the *FireLight Single Sign On Methods* document. It must be base64-encoded, and then url-encoded, before being sent in the token request's body.

Generating the Hash

A hash is required with the request's body in order to be verified by the service. This hash is generated off a string 'Secret' configured in the FireLight Admin tool.

These steps explain the process of generating the hash value:

1. Obtain the configured 'Secret' string value for whichever FireLight environment the request will be sent.
2. Create a SHA256 hash of the 'Secret' string value.
3. Convert the hashed byte array to upper-cased hex values.
4. This generated hash is the 'hash' variable of the request body content.

Only SHA256 hash values are supported for the 'hash' variable.

Generating the Signature

A base64-encoded digital signature string must be included on the request body. One of the certificates configured in the "Web Service Token Certificate" dialog in the FireLight Admin tool must be used to create the signature. FireLight will use the configured public key to verify the signature.

The digital signature is to be generated using the private key of the configured certificate, using either a SHA1 or SHA256 algorithm. Other algorithms are not supported.

The signature will be based off the content of a specific string. If a user 1228 is included, that portion will be part of the string used to generate the signature. The following text string formats are expected to be the content that is signed:

- If user 1228 is included:
  - **NonHashedSecret** + **RequestDate** + **Org** + **Base64User1228**
- If user 1228 is not included:
  - **NonHashedSecret** + **RequestDate** + **Org**

The following describes the variables:

Signature Variable	Expected Value	Description
NonHashedSecret	String Variable	This is the plain text (not the hashed hex representation) of the configured 'Secret' from the FireLight Admin tool.
RequestDate	String Variable of Request Date	This is the exact same value of the request's Date header. When formatted into a string, use this formatting: MMddyyyyHHmmss. If the formatting is different at the time of generating the signature, or the time is different from the request's Date header, the signature will be considered invalid.
Org	String Variable	This is the same value of the 'org' variable placed in the body. Contact Insurance Technologies for this variable.
Base64User1228	Base64 String Variable	This is the base64-encoded string of the user's 1228 xml, if applicable. **It is not url-encoded yet.** If your signature is based off the url-encoded version, it will not be verified by the FireLight system.

Response Format

A successful request will receive a token, which will be passed back as a JSON object within the body content. It will contain the text value of the token, the type of token, and the expiration (in seconds).

Response Item	Expected Value	Description
access_token	String	This is the string text value of the token. It will need to be passed along when used, as defined in the "Using the Token" section.
expires_in	Integer	This is the amount of time, in seconds, that the token will remain active. After this amount of time, the token will be considered expired and no longer valid to be used.
token_type	String	This is the type of token. It will need to be passed along when the token is used, as defined in the "Using the Token" section. It is expected to return as "Bearer".

Sample Request & Response

The following shows a successful request and its subsequent response, via Fiddler (highlighted for clarity) (the hex value and signature were slightly changed for security reasons):

Request

```
POST https://localhost/EGApp/api/Security/GetToken HTTP/1.1
User-Agent: Fiddler
Host: localhost
Content-Length: 2410
Content-Type: text/plain
Date: Wed, 14 Mar 2018 21:31:52 GMT

grant_type=hashsig&id=IT&secret=767F75E18449B0880D68B314A1F9674417CE8E48569D8A46D98DF1A13BE19894&sig=F3k41NYsvvhazOwszl2wOvL
7ZDCoDtXSWWa9KEuFMbVORoa%2FL4cdECFqcmT1Uj5UGG2B85E%2FfF%2F1Rr2zm5JqLbaBl49l43BJpphYGICbtXhimHeO4NsQkUUSSO%2FLlSIjMzhS%2FxpeY
2QcSaMKKDEvLvr1WITyRyjntcrkDYfxpsHTD7aSfArN3tfXzCMDOJc8Bslki1gMNk9VlYK83pyh1FP2DGw1ffzq4XAz%2FFFxhLZoW%2FuF7PmTOFU1YZDO5jAnZ
ClML74fa5sJUrSKMXrMRRkou7bbwPM29YiTbGm6VcA9tlIG6B67OF4FqfzaYfyWyMFTzdegOPZpS7n1yJK%2FKg%3D%3D&xml=PFRYTGlmZSB4bWxucz0iaHR0cD
ovL0FDT1JELm9yZy9TdGFuZGFyZHMvTGlmZS8yIj4NCiAgPFRYTGlmZVJlcXVlc3Q%2BDQogICAgPFRyYW5zUmVmR1VJRD4yNkMxQTRGQy1DMjJDLTQwMjctOUJG
NiO1NzVBRjM4QUFFMkE8L1RyYW5zUmVmR1VJRD4NCiAgICA8VHJhbnNUeXBlIHRjPSIxMjI4Ij5PTElfVFJBTlNfVFJOUFJPRElOUTwvVHJhbnNUeXBlPg0KICAg
IDxUcmFuc1N1YlR5cGUgdGM9IjIyODAwIj5PTElfVFJBTlNTVUJfUFJPRERFVEFMTDwvVHJhbnNTdWJUeXBlPg0KICAgIDxUcmFuc0V4ZURhdGU%2BMjAxNiOxMS
OwMjwvVHJhbnNFeGVEYXRlPg0KICAgIDxUcmFuc0V4ZVRpbWU%2BMTI6MjA6MDMuNTg3NjAyNC0wNjowMDwvVHJhbnNFeGVUaE1lPg0KICAgIDxPTGlmRT4NCiAg
ICAgIDxTb3VyY2VJbmZvPg0KICAgICAgICA8Q3JlYXRpb25EYXRlPjIwMTAtMTEtMDI8LONyZWFOaW9uRGFOZT4NCiAgICAgICAgPENyZWFOaW9uVGltZT4xMjoy
MDowMy41ODc2MDI0LTA2OjAwPC9DcmVhdGlvblRpbWU%2BDQogICAgICAgIDxTb3VyY2VJbmZvTmFtZT5HcmFudDwvU291cmNlSW5mbO5hbWU%2BDQogICAgICA8
L1NvdXJjZUluZm8%2BDQogICAgICA8UGFydHkgaWQ9IkFnZW5OX1BhcnR5Ij4NCiAgICAgICAgPFBhcnR5VHlwZUNvZGUgdGM9IjEiPk9MSV9QVF9QZXJzb248L1
BhcnR5VHlwZUNvZGU%2BDQogICAgICAgIDxGdWxsTmFtZT5BZ2VudHMgRnVsbCBOYW1lPC9GdWxsTmFtZT4NCiAgICAgICAgPEVNYWlsQWRkcmVzcz4NCiAgICAg
ICAgICA8QWRkckxpbmU%2BYWdlbnRzX2VtYWlsQGluc3VyYW5jZXRlY2hub2xvZ2llcy5jb208LOFkZHJMaW5lPg0KICAgICAgICA8LOVNYWlsQWRkcmVzcz4NCi
AgICAgICAgPFBlcnNvbiAvPg0KICAgICAgICA8UHJvZHVjZXI%2BDQogICAgICAgPENhcnJpZXJBcHBvaW5ObWVudCBQYXJOeUlEPSJBZ2VudF9QYXJOeSI%
2BDQogICAgICAgICA8Q29tcGFueVByb2R1Y2VySUQ%2BZ3JhbnQlX2lOXzI8LONvbXBhbnlQcm9kdWNlcklEPg0KICAgICAgICAgPENhcnJpZXJDb2Rl
PklUPC9DYXJyaWVyQ29kZT4NCiAgICAgICAgICA8LONhcnJpZXJBcHBvaW5ObWVudD4NCiAgICAgICAgPC9Qcm9kdWNlcj4NCiAgICAgICAgICAgIDwvUGFydHk%2BDQogIC
AgICA8TOxpZkVFeHRlbnNpb24gVmVuZG9yQ29kZTOiMjUiPgOKICAgICAgICA8VXNlclJvbGVDb2RlIHRjPSJGaWxOZXJUZXNOIj5UZXNOMTwvVXNlclJvbGVDb2
RlPgOKICAgICAgPC9PTGlmRUV4dGVuc2lvbj4NCiAgICAgICAgIDxSZWxhdGlvbiBPcmlnaW5hdGluZO9iamVjdElEPSJBZ2VudF9QYXJOeSI%2BDQogICAgICAgIDxS
ZWxhdGlvbiJvbGVDb2RlIHRjPSIzNyI%2BTOxJX1JFTF9QUklNQUdFTlQ8L1JlbGF0aW9uUm9sZUNvZGU%2BDQogICAgICA8L1JlbGF0aW9uPg0KICAgIDwvTOxp
ZkU%2BDQogIDwvVFhMaWZlUmVxdWVzdD4NCjwvVFhMaWZlPg%3D%3D
```

Response

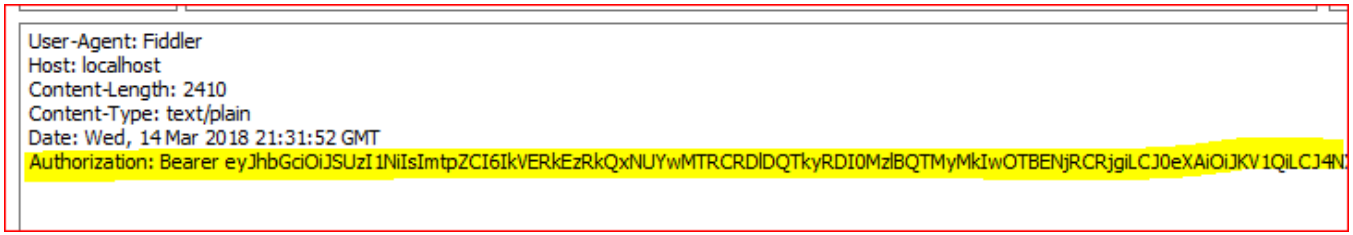
```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/10.0
X-Powered-By: ASP.NET
X-Content-Type-Options: nosniff
X-XSS-Protection: 1;mode=block
Content-Security-Policy: script-src 'self' 'unsafe-inline' 'unsafe-eval' https://entmax.comm100.com https://ent.comm100.com
https://www.google-analytics.com https://www.google.com/jsapi
Strict-Transport-Security: max-age=31536000;includeSubDomains
X-Frame-Options: SAMEORIGIN
Date: Wed, 14 Mar 2018 21:36:11 GMT
Content-Length: 767

{"access_token":"eyJhbGciOiJSUzI1NiIsImtpZCI6IkVERkEzRkQxNUYwMTRCRDlDQTkyRDI0MzlBQTMyMkIwOTBENjRCRjgiLCJ0eXAiOiJKV1QiLCJ4NXQiOiI3Zm9fMFY4QlM5bktrdEpEbXFNaXNKRFdTX2cifQ.eyJuYmYiOjE1MjEwNjMzNjYsImV4cCI6MTUyMTA2NDI2NiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDoyODUyNSIsImF1ZCI6WyJodHRwOi8vbG9jYWxob3N0OjI4NTI1L3Jlc291cmNlcyIsImVtYmVkZGVkIl0sImNsaWVudF9pZCI6IklUIiwidXNlcl9pZCI6ImdyYW50NV9pdF8yIn0.XNPsxB4O6EOv3yPd45JHReBWTiMmOvG9fWR9zPAevkNsIxF845HDT4fdz-Bb6W94rQDrJmpFizc9Oj4YxGq7qOIQsqplG1OC0kgpE2COek9ow6nOm6R5pdY5fK6fL26S7FVJYdIptpsfDK-y6pwOqd9DHEhYuPNiDB-XYN4rvUPm8U7idD5ZnxZgEDbDQKKaIuVzZsYWK8v1Ip-BsC_NPQri0nQd38Bc9kSe3vj977ECM61DjxU5IZGQ2EckRyZQ4q6O_bsuLMzDi2WK0EqgJe8JO2zz6azeLcPPwPB3co186b5VIhuWGRUc883vK5u5XGBfsZt-4thn0gce2Jv2gg","expires_in":900,"token_type":"Bearer"}
```

Using the Token

For FireLight API service endpoints that use the JWT token security scheme, the token string will need to be placed on any requests made to that service in the 'Authorization' header. The header will need to contain the 'token_type' value received from the response, followed by a space, followed by the 'access_token' value received from the response:

token_type + ' ' + **access_token**

'token_type' will be "Bearer", and so it will look like the following:



Code Sample

The following is a C# code sample of generating the request to get a token.

```
HttpWebRequest req = (HttpWebRequest)WebRequest.Create(
    "https://firelight.insurancetechnologies.com/EGApp/api/Security/GetToken");

req.Method = "POST";
req.ContentType = "text/plain";

string secret = "46098ef35a824170a584bde268586c7b";//this comes from our admin tool
byte[] secretBinary = Encoding.UTF8.GetBytes(secret);
byte[] hashBinary = new SHA256Managed().ComputeHash(secretBinary);

//convert to hex string
StringBuilder builder = new StringBuilder();
for (Int32 idx = 0; idx < hashBinary.Length; idx++)
{
    builder.Append(hashBinary[idx].ToString("X2"));
}
string hashValue = builder.ToString();//this gets appended on the request body

DateTime reqDate = DateTime.UtcNow;
String org = "IT";//this will be different for your organization

//this is the user's 1228 xml - you will need to generate yours as needed
string xml1228 = txt1228.Value;
if (!string.IsNullOrWhiteSpace(xml1228))
    xml1228 = Convert.ToBase64String(Encoding.UTF8.GetBytes(txt1228.Value));

//remember, the user 1228 is optional
String textToSign = secret + reqDate.ToString("MMddyyyyHHmmss") + org + xml1228;
byte[] nonSignedBinary = Encoding.UTF8.GetBytes(textToSign);

//load up the test cert - in this example, it comes from a pfx file in a local directory - you
// may need to get it from your machine's certificate store
X509Certificate2 cert = new X509Certificate2(Path.Combine(
    AppDomain.CurrentDomain.BaseDirectory, txtPfxFile.Text), txtPfxPassword.Text);

RSA rsa = cert.GetRSAPrivateKey();
```

```
byte[] signedBinary = rsa.SignData(nonSignedBinary, HashAlgorithmName.SHA256,
    RSASignaturePadding.Pkcs1);

//base64-encoded, then url-encoded
String signed64 = HttpUtility.UrlEncode(Convert.ToBase64String(signedBinary));

//optionally place the user's 1228 on the request if it has value (url-encode it)
String body = $"grant_type=hashsig&id={org}&secret={hashValue}&sig={signed64}"
    + (string.IsNullOrWhiteSpace(xml1228) ? "" : $"&xml={HttpUtility.UrlEncode(xml1228)}");

byte[] bodyBinary = Encoding.UTF8.GetBytes(body);

req.Date = reqDate;
req.ContentLength = bodyBinary.LongLength;

using (Stream post = req.GetRequestStream())
{
    post.Write(bodyBinary, 0, bodyBinary.Length);
}

// Get response
String result = "";
using (HttpWebResponse response = (HttpWebResponse)req.GetResponse())
{
    StreamReader reader = new StreamReader(response.GetResponseStream());
    result = reader.ReadToEnd();
}
```