Software Rights Management

Sentinel CLOUD SERVICES™

Cloud Connect Web Services Guide

SafeNet®

Software Version

This documentation is applicable for the Sentinel Cloud Version 3.4.

Revision History

Part Number **007-012160-001**, Revision **E**

Revision	Action/Change	Date
A	Initial version created for Sentinel Cloud 3.0 Release	December 2012
B	Updated for Sentinel Cloud 3.1 Release	April 2013
C	Updated for Sentinel Cloud 3.2 Release	June 2013
D	Updated for Sentinel Cloud 3.3 Release	September 2013
E	Updated for Sentinel Cloud 3.4 Release	December 2013

Disclaimer and Copyrights

http://www.safenet-inc.com/

We have attempted to make this document complete, accurate, and useful, but we cannot guarantee it to be perfect. When we discover errors or omissions, or they are brought to our attention, we endeavor to correct them in succeeding releases of the product. SafeNet, Inc., is not responsible for any direct or indirect damages or loss of business resulting from inaccuracies or omissions contained herein. The specifications contained in this document are subject to change without notice.

Table of Contents

Preface

Who Should Read This Document?

This document is intended for software providers who want to integrate Sentinel Cloud licensing programmatically in their application.

Conventions Used in This Document

Convention	Description
Bold lettering	Denotes keystrokes, menu items, window names, and fields.
`Courier`	Denotes syntax, prompts, and code examples.
Italic lettering	Denotes file names and directory names. Else, used for emphasis.

Documentation Resources

Document	What's in it?	Who Should Read it?
EMS User's Guide	Reference for using the Sentinel Cloud EMS Web portal for Sentinel Cloud Services	Product Managers and software providers responsible for delivering and deploying the products
EMS Web Services Guide	Reference for using the Sentinel Cloud EMS Web services	Developers responsible for integrating Sentinel EMS with backend systems
EMS Web Services Cheatsheet	A quick reference document that summarizes all the available EMS Web services	Developers responsible for integrating Sentinel Cloud EMS with backend systems
Cloud Run-time Guide	Sentinel Cloud Run-time API reference	Developers responsible for integrating APIs in the licensed application
Cloud Run-time Java Demo Application ReadMe	A demo application and reference for using the Cloud Run-time APIs in Java	Developers and Product Managers
Cloud Run-time .NET Reference ReadMe	Sample reference for using the Cloud Run-time APIs in .NET	Developers responsible for integrating APIs in .NET applications
Cloud Run-time C Reference ReadMe	Sample reference for using the Cloud Run-time APIs in C	Developers responsible for integrating APIs in C applications
Cloud Connect Web Services Guide	Reference for using the Sentinel Cloud Connect Web Services	Developers responsible for integrating Cloud Connect Web Services in the licensed application
Cloud Connect Web Services Python Sample ReadMe	Sample reference for using the Cloud Connect Web Services in Python	Developers responsible for integrating Cloud Connect Web Services in Python applications

Document	What's in it?	Who Should Read it?
Quick Start Guide	Document to help you quickly start with Sentinel Cloud	Developers and Project Managers
Release Notes	Contains product overview, summary of new features, and enhancements	All
Acknowledgements	Acknowledgements of the third-party software used in the product	General document
Installation Guide	Contains installation information	Administrators responsible for installing Sentinel Cloud Services
Migration Guide	Contains information helpful in migrating from one version to another	Developers and Project Managers

Obtaining Support

You can contact us using any of the following options:

- **Business Contacts** - To find the nearest office or distributor, use the following URL: http://www.safenet-inc.com/contact-us/
- **Technical Support** - To obtain assistance in using SafeNet products, feel free to contact our Technical Support team:
  - Phone: 800-545-6608 (US toll free), +1-410-931-7520 (International)
  - E-mail: support@safenet-inc.com
  - URL: http://sentinelcustomer.safenet-inc.com/sentinelsupport/
- **Downloads** - You may want to check out updated installers and other components here: www.sentinelcustomer.safenet-inc.com/sentineldownloads/

Documentation Feedback

To help us improve future versions of the documentation, we want to know about any corrections, clarifications, or further information you would find useful. When you contact us, please include the following information:

- The title, part number (if applicable), and version of the document you are referring to
- The version of the product you are using
- Your name, company name, job title, phone number, and e-mail ID

Send us e-mail at: support@safenet-inc.com

1

Introduction

Sentinel Cloud Connect Web services offer a framework for integrating Sentinel Cloud Services in applications written for any language or platform.

1.1. Overview

Sentinel Cloud Connect Web services expose licensing and monetization abilities of Sentinel Cloud for direct use in an ISV application. Any platform or language that supports HTTP call with SSL server authentication can integrate Sentinel Cloud Services.

With Sentinel Cloud Connect Web services, you can integrate Sentinel Cloud in applications running on platforms which are not supported by existing Cloud Run-time.

1.2. Cloud Connect Web Services or Run-time: How to Choose?

There are two methods of integrating Sentinel Cloud licensing in your application—Sentinel Cloud Run-time and Sentinel Cloud Connect Web Services. Each is designed to benefit a different target audience. Cloud Connect Web Services provide a simple programming model and can be used for a wide range of platforms/languages. Cloud Run-time provides a more complex programming model but is available only for specific platforms/languages.

Benefits of Run-time	Benefits of Cloud Connect Web Services

▪ Available for Java, .NET, and C platforms.
▪ Better ease of use for applications developed on standard platforms.
▪ Better performance because of the following built-in features:
  ○ Caching (for non-concurrent licensing models)
  ○ Asynchronous usage log update
▪ Ease of integration as the Run-time encapsulates all remote HTTPS calls.
	▪ Platform-independent and thus best suited for applications running on non-standard platforms or in restricted programming environments.
	▪ Flexible and provides more control .
	▪ Simple programming model and easy to integrate in ISV application.

1.3. Cloud Connect Web Services Highlights

Please note the following points about Sentinel Cloud Connect Web services:

- Cloud Connect Web services provide support only for Cloud licensing, not for On-premise licensing.
- All Cloud Connect Web services are license agnostic in nature. This means that licensing Web service calls are independent of the license model, whether globally concurrent or not.
- Communication between ISV application and Sentinel Cloud components happens over SSL with server authentication.
- The HMAC-SHA1 algorithm is used to sign and authenticate a Web service request.
- All Cloud Connect Web services support internationalization so they can handle inputs provided in any local language.
- In the case of an error, the error code with error description is set in the Cloud Connect Web service response; otherwise, a proper response XML is returned.
- All the request messages must comply with the XML schema provided by SafeNet. Otherwise, an error is returned in response.
- Request-response messages should be in UTF-8.

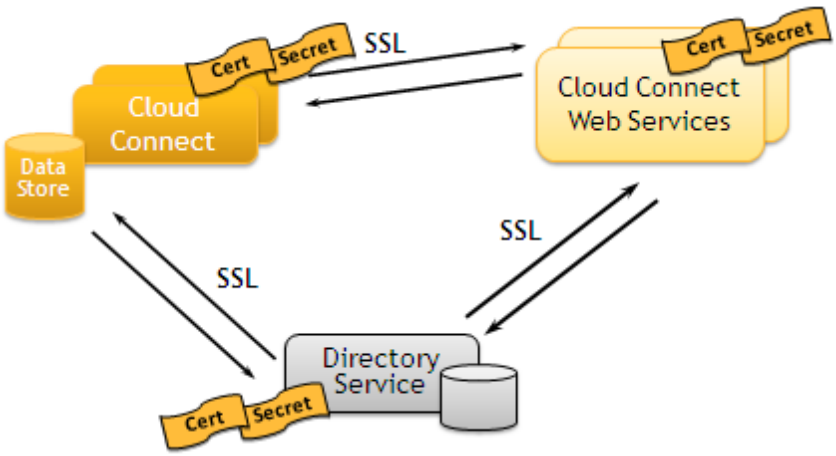
1.4. Secure Communication

Critical communication takes place between the various components of Sentinel Cloud Services. To ensure safety, the communication between the following happens over SSL (https):

- Cloud Connect Web Services and Cloud Directory Services
- Cloud Connect Web Services and Cloud Connect

If a component is validated, a session will be established, otherwise the communication will be denied. This secure communication channel:

- Prevents acknowledgment of any external data on each end.
- Prevents tampering of data in transit.
- Maintains the integrity of the usage data.



A server component authenticates itself to a client application by presenting the certificate issued by a public Certificate Authority (CA). All server-side components (Cloud Connect, EMS, and Directory Services) use certificates signed by a well-known public CA for authentication.

The client requests are authenticated by using the process of *Message Signing*, as explained in the next section.

1.5. Authentication

Authentication is the process of verifying the ISV client that consumes Sentinel Cloud Connect Web services. A Cloud Connect Web service request is accepted or rejected based on authentication. Sentinel Cloud Connect Web services use the method of *Message Signing* for authenticating a Web request.

The ISV client signs a well-defined string by using a *Secret Key*. The HMAC-SHA1 (Hash Message Authentication Code) algorithm is used for signing the string. The output is a signed string, called *Signature*, which is added as a parameter to each Web service request. The subsequent sections explain how to create a signature and include the signature in a Web service request.

When a Sentinel Cloud component (Cloud Connect/Directory Services) receives a Web service request, it derives the secret key and computes the signature by the same mechanism. This signature is compared with the signature received in Web service request. If both the signatures match, the authentication is successful and the Cloud component accepts the Web service request. If the two signatures do not match, the Cloud component denies the Web service request and sends an error message.

The Message Signing method for authentication includes the following main components:

- Secret Key/Secret Key ID
- String to sign
- Signature
- Headers
- Time stamp

1.5.1. Secret Key

Every ISV client is assigned a Secret Key and Secret Key ID pair, which is used for authenticating requests.

1.5.2. String to Sign

The first step in message signing is to concatenate selected elements of the request to form a string. Following is pseudo-grammar that illustrates the construction of string to sign:

```
StringToSign =  HTTP-Verb + "\n" +
                Content-length + "\n" +
                Content-MD5 + "\n" +
                Content-Type + "\n" +
```

```
                CanonicalizedSFNTHeaders +
                CanonicalizedResource;
```

The parts of the above format are described below:

- **HTTP-verb**: Since HTTP request type for Cloud Connect Web services is POST, the HTTP-verb is POST.
- **Content-length**: Length of request message.
- **Content-MD5**: MD5 of request message.
- **Content-Type**: Type of request message. For example, `text/xml; charset=utf-8.`
- **CanonicalizedSFNTHeaders**: Contains a customized header "x-sfnt-date". It is mandatory to set this header for client authentication. The value of x-sfnt-date specifies time in milliseconds, since January 1, 1970, 00:00:00 GMT.
- **CanonicalizedResource**: This is the string containing the name and version of the Cloud Connect Web service used in request URI.Example: `/login?version=1.0`

In the above format, Content-length, Content-MD5, and Content-Type are standard HTTP headers. The client must set these headers in each Web service request, with the same values as specified in string to sign.

Example

Request URI: *https://scc.na.sentinelcloud.com/cloudconnect/clientAlias/login?version=1.0*

Request Message:

```
<loginRequest>
  <user>myUser</user>
  <customer>myCustomer</customer>
  <featureId>1</featureId>
  <vendorData>vSpecificData</vendorData>
  <machineId>hostName</machineId>
  <vendorId>a8e06c3</vendorId>
</loginRequest>
```

Sample String to Sign

```
POST
198
MDc5OWJiMzIzYzBhNTM5NjZmZDM2NmYxN2YxYzViODU=
text/xml; charset=utf-8
x-sfnt-date:1354862060857
/login?version=1.0
```

1.5.3. Signature

After forming a string to sign, the secret key is used to calculate the HMAC of the string, which is called signature. Following is the pseudo-grammar that illustrates the construction of a signature.

```
Signature = Base64( HMAC-SHA1( YourSecretKey, UTF-8-Encoding-Of( StringToSign ) ) )
```

HMAC-SHA1 () is an algorithm defined by RFC 2104 (RFC 2104 - Keyed-Hashing for Message Authentication). The algorithm takes as input two byte-strings: a key and a message. Use your Cloud Connect Web service Secret Key as the key, and the UTF-8 encoding of the string to sign as the message. The output of HMAC-SHA1 is also a byte string, called the digest. The Signature request parameter is calculated by Base64 encoding of this digest.

1.5.4. Headers

The Sentinel Cloud Connect Web services use the following HTTP headers:

- Authorization
- Content-Length
- Content-MD5
- Content- Type

Authorization Header

Sentinel Cloud Connect Web services use the standard HTTP Authorization header to pass authentication information. The Authorization header has the following format.

```
Authorization: SCWS SecretKeyId:Signature
```

The ISV client that will consume Sentinel Cloud Connect Web services is issued a Secret Key Id and a Secret Key. For request authentication, the Secret Key Id element identifies the Secret Key that was used to compute the signature, and the user making the request.

The signature of the Authorization header will vary from request to request. If the request signature calculated by the system matches the signature included with the request, the requester is considered to be holding Sentinel Cloud Connect Web services Secret Key. The request is then processed further.

Example: A Sample Authorization Header

```
Authorization: SCWS ISVKEYID:bWq2s1WEIj+Ydj0vQ697zp+IXMU=
```

Content-Length, Content-MD5, and Content-Type Headers

These headers should also be part of HTTP request. Please note:

- The Content-Type header must be set to inform the Cloud component to read request message.
- The values of headers, which are used in stringToSign for signature calculation, must be same as set in request headers.Otherwise, the client authentication will fail.

1.5.5. Time Stamp

Time stamp must be set in string to sign, in x-sfnt-date. It is a customized header that contains time in milliseconds, since January 1, 1970, 00:00:00 GMT.

For Example:

```
x-sfnt-date:1354699124799
```

If a request is older than 15 minutes, it will be discarded and the client authentication will fail.

2

Sentinel Cloud Connect Web Services

Sentinel Cloud provides the following Cloud Connect Web services:

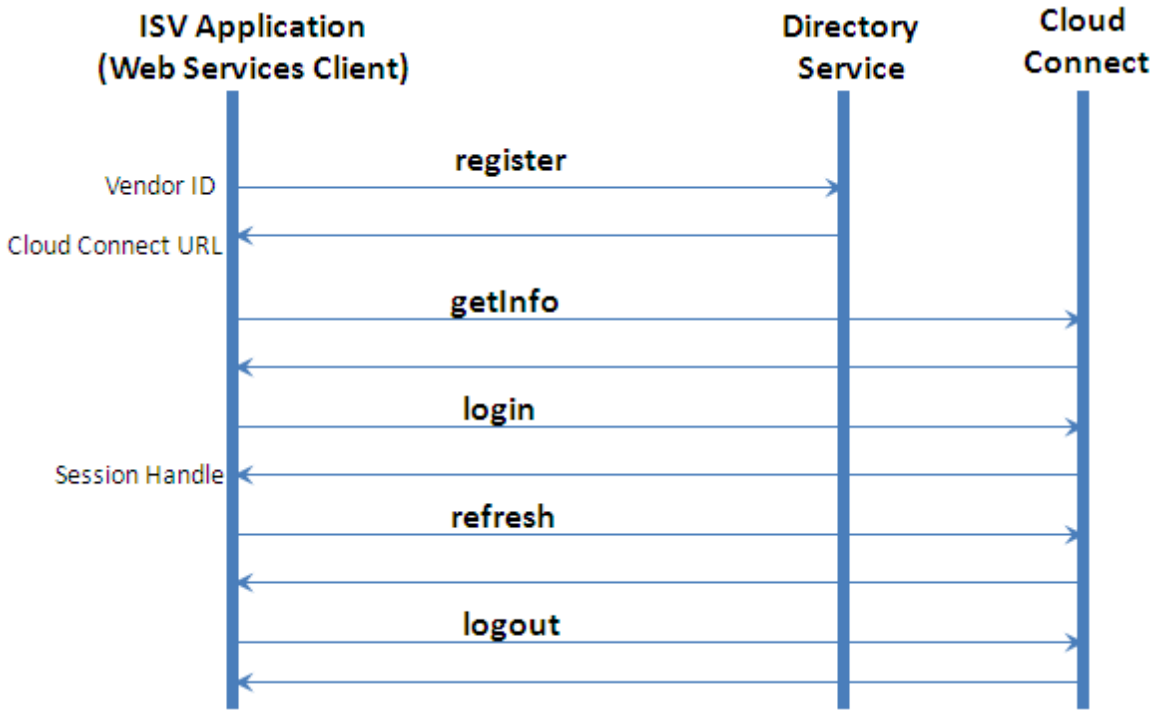
- register: Used to register an ISV client with Directory services and retrieve Cloud Connect URL.
- getInfo: Used to return information about entitlements.
- login: Used to consume a feature.
- refresh: Used to refresh a concurrent session.
- logout: Used to release a feature.

2.1. Workflow of Sentinel Cloud Connect Web Services

Before calling a Cloud Connect Web service from an application, an ISV needs to develop the following modules:

1. **Communication Module**: Create HTTPS communication module to send and receive messages. The ISV application should establish SSL session with Directory Services and Cloud Connect using CA certificate. The message exchange happens over SSL.
2. **Signature Module**: Create HMAC-SHA1 signature as specified by SafeNet to authenticate an ISV node. Each ISV request is authenticated by Sentinel Cloud Services.
3. **Message Module**: Directory Services and Cloud Connect use XML messages for sending request and response. Therefore ISV client application should develop module to create and read XML

The ISV application (Web services client) should call Cloud Connect Web Services in the following order:



1	Send the *register* request to Directory Services with Vendor ID, to register the client and acquire Cloud Connect URL for subsequent Web service calls. Directory Services will return XML response.
	After the success of the *register* request, the client can call other Web services by using the acquired URL.
2	Send the *getInfo* request for entitlement details for customer, identity, format, and scope. Cloud Connect will return response.
3	Send the *login* request to consume feature with details of customer, feature, and user. Cloud Connect will return session handle.
	The client will store this session handle to use the feature. The same session handle is used later either to *refresh* this session or *logout* from session.
4	Send a periodic *refresh* request to Cloud Connect with the acquired session handle to refresh a particular session. This request is sent for features with concurrent license models.
5	Send *logout* request, to Cloud Connect to release a feature. The request is sent with the acquired session handle.
	On calling *login*/*logout*/*refresh*/*getInfo*, if the HTTP error 410 occurs, retrieve the Cloud Connect URL again by calling *register* Web service.

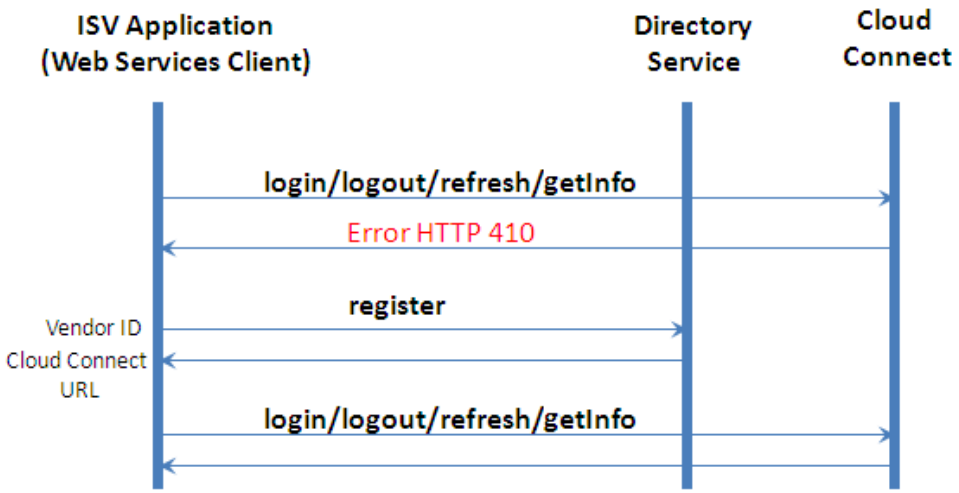
About HTTP 410 Error

Cloud Connect sets HTTP 410 error when the resource requested is no longer available. If Cloud connect does not support a Vendor ID, the ISV application communication module traps HTTP 410 error, connects with Directory Services again, and retrieves a new Cloud Connect URL.

For other errors (such as licensing and parameter errors), the Web services client gets an error in XML which need to be parsed. The HTTP error is not used in this case.

Sample flow to handle the HTTP 410 error is:

1. Call the *register* Web service to get the Cloud Connect URL.
2. Call *login/logout/refresh/getInfo* on Cloud Connect URL.
3. If HTTP error 410 is encountered in step 2 , go to Step 1.



2.2. URI Conventions

A Web Service call or URI has the following syntax:

```
<url>/<clientAlias>/<webServiceName>?<webServiceVersion>
```

Example:

```
https://yps.na.sentinelcloud.com/YPServer/register?version=1.0
```

```
https://scc.na.sentinelcloud.com/cloudconnect/clientAlias/login?version=1.0
```

Here:

- `url`: This is the URL of Directory Services or Cloud Connect used in Web service calls. You receive Directory Service URL from SafeNet and Cloud Connect URL from *register* Web service.

  The `url` in the above examples are: *https://yps.na.sentinelcloud.com/YPServer/* and *https://scc.na.sentinelcloud.com/cloudconnect*.

- `clientAlias`: This is usually the vendor name specified in the vendor registration request. For the register Web service, clientAlias is not required.

- `webServiceName`: The name of the Web service you want to call.
- `webServiceVersion`: The version of Cloud Connect Web Services, which is 1.0.

Notes

- The request message should be part of request body.
- HTTP POST method is used.

2.3. register

2.3.1. Description

Registers the ISV node with Directory Service.The workflow of this Web service is:

- Registers the ISV node with Directory Service.
- Directory Service searches the Cloud Connect URL for the registered ISV node, and returns it in response to the Web service call.

2.3.2. Example URI

```
https://yps.na.sentinelcloud.com/YPServer/register?version=1.0
```

2.3.3. Query Parameter

Parameter	Description	Valid Values	Optional
version	Version of Cloud Connect Web Services	1.0	No

2.3.4. HTTPS Method

POST

2.3.5. Request Message

```
<registerRequest>
 <vendorId>a8e06c3</vendorId/>
 <machineId>hostName</machineId>
 <nodeDesc>ISV node for sample application</node-desc>
</registerRequest>
```

Request Parameters

Parameter	Data Type	Description
vendorId	String	Unique identifier of the ISV.
machineId	String	Host name

Parameter	Data Type	Description
nodeDesc	String	Description of the ISV node from which the request is received.

2.3.6. Response Message

In case of success:

```
<registerResponse>
  <status>OK</status>
  <urlList>
    <url value="https://scc.na.sentinelcloud.com/licenseserver"/>
  </urlList>
</registerResponse>
```

In case of error:

```
<registerResponse>
       <status>Fail</status>
       <errorCode>1001</errorcode>
       <errorDesc>error_message</errorDesc>
</registerResponse>
```

2.3.7. Error Codes

Error Code	Description
1001	This web service version is not supported
1004	Invalid parameter: machineId
1007	Invalid parameter: vendorID
1008	Invalid request message
1009	Request from this vendorID is not supported
1011	The request XML is not well formed
1015	Internal error
1027	Authentication Failed
1028	Authorization header not found
1029	x-sfnt-date header not found
1031	No authorization server mapped to given vendor Id

2.4. getInfo

Returns information of entitlements, products, and features for a given user of the customer. It also helps check the license availability for a user.

2.4.1. Description

This Web service helps get authorization information of an entitlement. It fetches information based upon provided scope and returns results in the specified format.

2.4.2. When to use the getInfo Web service?

You can use the getInfo Web service in the following situations:

- **Rendering application for a particular customer/user for better user experience**

  The getInfo Web service obtains a list of features that are available for use for a given user. You can use this information to design an interface for the end user based on the features the user is authorized to use. For example, you can display only those features to the user for which the user is authorized, or you can enable the allowed features and disable the rest. This helps in creating a better end user experience.

  You can retrieve information about features and its attributes—such as the start date, end date, maximum count and count consumed for pre-paid licenses, and seats and maximum limit for concurrent licenses. You can find all the details pertaining to an entitlement, such as the entitlement ID and products belonging to it. You can provide this information to your end users to keep them updated about their entitlement status, thus creating a better user experience.

  
  The getInfo provides contents of an entitlement, but does not provide its usage-related information (except in case of prepaid licenses).

- **Configuring Notifications**

  You can use getInfo to identify if the license is in grace state and can configure notifications for the end user accordingly.

- **Retrieving Prepaid Usage**

  You can call getInfo to show the usage available for prepaid license models.

  You can retrieve usage count of the pre-paid licenses. Usually, you call getInfo at the time when the user logs on to the licensed application. However for pre-paid licenses, you might need to call it after logon to show the usage available depending on your application design.

- **Seamless Integration with EMS Web Services**

  The getInfo Web service returns the EID which can be used for EMS login and retrieving usage-related details from EMS. For example, you can use the EID returned by getInfo in the loginByEID Web service to log in to EMS. By using other Web services, you can get license attribute data and information about consumption of a particular entitlement, product, or feature.

  
  The EMS URL cannot be auto-discovered and you need to pass it. For details about other information that needs to be passed along with an EMS Web service, please refer to *EMS Web Services Guide*.

2.4.3. When NOT to use the getInfo Web service?

- Do not use getInfo if you want to retrieve usage of subscription, postpaid, and concurrent license models.
- Do not use getInfo to retrieve static and aggregated data as required in case of subscription and postpaid licenses. Examples of static and aggregated data include details of an entitlement, product, feature, and license attributes (start date, end date, etc.).

In all the cases above, you should use *EMS Web Services* instead of getInfo.

2.4.4. Example URI

```
https://scc.na.sentinelcloud.com/cloudconnect/clientAlias/getInfo?version=1.0
```

2.4.5. Query Parameter

Parameter	Description	Valid Values	Optional
version	Version of Cloud Connect Web Services	1.0	No

2.4.6. HTTPS Method

POST

2.4.7. Request Message

```
<getInfoRequest>
       <user>t1</user>
       <customer>t1</customer>
       <featureId>1</featureId>
       <productName>pname^version</productName>
       <entitlementId>eid</entitlementId>
       <format>4</format>
       <vendorId>a8e06c3</vendorId>
</getInfoRequest>
```

Request Parameters

Parameter	Data Type	Description
user	String	Identifies the user. Required for both enterprise (named and unnamed) and retail users.  Make sure that user passed in this Web service and the one used for configuring the license are exactly same.
customer	String	The unique reference ID of the customer as provided in EMS.  This parameter is optional for a retail user. However, it is must for supporting unnamed licenses in an enterprise.

Parameter	Data Type	Description
featureId	Integer	Unique feature ID used to identify a feature.
productName	String	Combination of product name and version. It is specified in the format: `<productname>^<version>`. For example, `Trial^1.0`.
entitlementId	String	Unique entitlement ID used to identify an entitlement.
format	Integer	Sets the output format (level of detail) of the getInfo Web service. See Format Type for details.
vendorId	String	Unique identifier of the ISV.

Scope

The scope of *getInfo* call is defined by a combination of entitlementId, productName, and featureId, as decribed below.

Possible Combinations

Following are the possible combinations for the request parameter of the getInfo Web service:

entitlementId	productName	featureId	Result
Null	Null	Null	Information of all features of all products in all entitlements related to that customer and user.
Not null	Null	Null	Information of all features and products for a given EID.
Not null	Not null	Null	Information of all features for the given productName and EID.
Not null	Not null	Not null	Information for the given featureId, productName, and EID.
Null	Not null	Not null	Information for the given featureId and

entitlementId	productName	featureId	Result
			productName in all entitlements.
Null	Null	Not null	Information for the given featureId in all the products and entitlements.
Not null	Null	Not null	Information for given featureId in all the products for the given EID.

Format Type

The format type specifies the level of detail you want to obtain from the getInfo Web service, and how to display the obtained information.

There are three format types all of which show entitlement ID, product name, feature ID, and feature name. The following table describes the three format types:

Format Type	Value	Description
HIERARCHY_INFO	1	Returns the entitlement ID, product name, feature ID, and feature name
FEATURE_AUTHORIZATION	2	Returns the usability flag, usability status, and time zone; in addition to the information returned by HIERARCHY_INFO. The usability flag returns a Boolean value to show whether the usability status is available (true) or not (false).
FEATURE_DETAILS	4	Shows detailed information of a feature in addition to the information returned by FEATURE_ AUTHORIZATION. In the case of Cloud deployment, the feature information can include license model, start date, end date, concurrency, prepaid count and other license attributes.

If no value or a value other than the above formats is specified, HIERARCHY_INFO (1) is returned by default.

2.4.8. Response Message

```
<getInfoResponse>
  <status>OK</status>
  <entitlements>
    <entitlement entitlementId="c3245cae-8c44-45e2-9deb-6e1c963c2064" timeZone="(GMT) Greenwich
Mean Time, : Dublin, Edinburgh, Lisbon, London">
      <products>
        <product name="Product-1^2.1">
          <features>
            <feature id="1" name="Postpaid-1" usable="true" usabilityStatus="Available">
              <notifications>0</notifications>
              <license>
                <licenseModelType>PostPaid-Time</licenseModelType>
                <licenseAttributes>
                  <attribute name="Start Date" value="2012-12-12 00:00:00"/>
                  <attribute name="End Date" value="2013-12-12 23:59:00"/>
                  <attribute name="Vendor Attribute" value=""/>
                </licenseAttributes>
              </license>
            </feature>
            <feature id="2" name="Concurrent-2" usable="true" usabilityStatus="Available">
              <notifications>0</notifications>
              <license>
                <licenseModelType>Concurrent-Subscription-Time</licenseModelType>
                <licenseAttributes>
                  <attribute name="Start Date" value="2012-12-12 00:00:00"/>
                  <attribute name="End Date" value="2013-12-12 23:59:00"/>
                  <attribute name="Vendor Attribute" value=""/>
                  <attribute name="ConcurrentLimit" value="2"/>
                  <attribute name="ConcurrentCounting Type" value="Login"/>
                </licenseAttributes>
              </license>
            </feature>
            <feature id="3" name="Prepaid-3" usable="true" usabilityStatus="Available">
              <notifications>0</notifications>
              <license>
                <licenseModelType>Prepaid-Count</licenseModelType>
                <licenseAttributes>
                  <attribute name="Start Date" value="2012-12-12 00:00:00"/>
                  <attribute name="End Date" value="2013-12-12 23:59:00"/>
                  <attribute name="Vendor Attribute" value=""/>
                  <attribute name="Max Count" value="2"/>
                  <attribute name="Count Consumed" value="1"/>
                  <attribute name="Grace Limit" value="5"/>
                  <attribute name="Measurement Unit" value="Count"/>
                </licenseAttributes>
              </license>
            </feature>
            <feature id="4" name="Subscription-4" usable="true" usabilityStatus="Available">
              <notifications>1</notifications>
              <license>
                <licenseModelType>Subscription-Time</licenseModelType>
                <licenseAttributes>
                  <attribute name="Start Date" value="2012-12-04 00:00:00"/>
                  <attribute name="End Date" value="2012-12-11 23:59:00"/>
                  <attribute name="Vendor Attribute" value="Grace of 30 days for Subscription-4
"/>
                  <attribute name="Grace Limit" value="30"/>
```

```
                <attribute name="Measurement Unit" value="Days"/>
              </licenseAttributes>
            </license>
          </feature>
        </features>
      </product>
    </products>
  </entitlement>
 </entitlements>
</getInfoResponse>
```

In case of error:

```
<getInfoResponse>
  <status>Fail</status>
  <errorCode>1001</errorCode>
  <errorDesc>error_message</errorDesc>
</getInfoResponse>
```

2.4.9. Error Codes

Error Code	Description
1001	This web service version is not supported
1002	Invalid parameter: user
1003	Invalid parameter: Customer
1004	Invalid parameter: machineId
1005	Invalid parameter: featureId
1006	Invalid parameter: vendorData
1007	Invalid parameter: vendorID
1008	Invalid request message
1010	Invalid web service URL
1011	The request XML is not well formed
1012	Not authorized to process any request
1015	Internal error
1017	License is not in active state
1018	License is expired
1019	License is disabled
1020	License is revoked
1021	Maximum concurrent user limit reached
1022	Maximum usage count reached
1023	License does not exist or license is not in active state
1024	Invalid parameter

Error Code	Description
1026	Access denied to the requested feature
1027	Authentication Failed
1028	Authorization header not found
1029	x-sfnt-date header not found
1030	Invalid parameter: format
1032	Invalid parameter: productName
1033	Invalid parameter: entitlementId

2.5. login

Obtains authorization for a feature requested by user, establishing a session between the licensed application and Cloud Connect.

2.5.1. Description

Only an authorized user can access features of the licensed application. The login Web service is used to authorize the user access to a feature. Once an authorization is obtained, the login call establishes a session between the licensed application and the Cloud Connect. Thereafter, this session is used for exchanging information.

2.5.2. Example URI

```
https://scc.na.sentinelcloud.com/cloudconnect/clientAlias/login?version=1.0
```

2.5.3. Query Parameter

Parameter	Description	Valid Values	Optional
version	Version of Cloud Connect Web Services	1.0	No

2.5.4. HTTPS Method

POST

2.5.5. Request Message

```
<loginRequest>
       <user>t1</user>
       <customer>t1</customer>
       <featureId>1</featureId>
       <vendorData>vSpecificData</vendorData>
       <machineId>hostName</machineId>
       <vendorId>a8e06c3</vendorId>
</loginRequest>
```

Request Parameters

Parameter	Data Type	Description
user	String	Identifies the user to be authorized. Required for both enterprise (named and unnamed) and retail users. For named and retail users, make sure that user passed in this Web service and the one used for configuring the license are exactly same.
customer	String	The unique reference ID of the customer as provided in EMS. This parameter is optional for a retail user. However, it is must for supporting unnamed licenses in an enterprise.
featureId	Integer	Unique feature ID used to identify a feature.
vendorData	String	Vendor information or the remarks to store in usage record if a feature is authorized successfully.
machineId	String	Host name
vendorId	String	Unique identifier of the ISV.

2.5.6. Response Message

In case of success:

```
<loginResponse>
  <status>OK</status>
  <sessionHandle>rT9LfprcD%2FI6fwBwn7L9nK3mNAall25C%0D%0AvZAizsyn%2BVdJIT8GoRsCvAZrty</sessionHan-
dle>
</loginResponse>
```

In case of error:

```
<loginResponse>
  <status>Fail</status>
  <errorCode>1001</errorCode>
  <errorDesc>error_message</errorDesc>
</loginResponse>
```

2.5.7. Error Codes

Error Code	Error Description
1001	This web service version is not supported
1002	Invalid parameter: user
1003	Invalid parameter: Customer
1004	Invalid parameter: machineId
1005	Invalid parameter: featureId

Error Code	Error Description
1006	Invalid parameter: vendorData
1007	Invalid parameter: vendorID
1008	Invalid request message
1010	Invalid web service URL
1011	The request XML is not well formed
1012	Not authorized to process any request
1015	Internal error
1016	Error occurred in usage update
1017	License is not in active state
1018	License is expired
1019	License is disabled
1020	License is revoked
1021	Maximum concurrent user limit reached
1022	Maximum usage count reached
1023	License does not exist or license is not in active state
1024	Invalid parameter
1026	Access denied to the requested feature
1027	Authentication Failed
1028	Authorization header not found
1029	x-sfnt-date header not found

2.6. logout

Releases the user authorization.

2.6.1. Description

Call this Web service when the user has finished using the feature.

The licensed application must manage the session handle received from the login call for each user.

Cloud Connect Web Services are stateless, so there is no need to free any resources using the logout Web service. However, it is important to call this Web service, in a match with the login Web service, to log the license usage.

Special Cases

- If the licensed application exits without calling logout, the actual session logout time will not be updated in the usage logs. It is the software publisher's decision to account

for such incomplete sessions. A fixed session period could be assumed for such cases (for billing purposes). This period needs to be set on the Cloud Connect side. Contact SafeNet Technical Support for assistance.

- In the case of concurrent licenses, if logout is called while the Cloud Connect is down, the license will be released automatically after a pre-defined period. The default value is 24 hours and can be configured by SafeNet. For count-based licenses, one count will be consumed and currently this is not configurable by the publisher.

2.6.2. Example URI

```
https://scc.na.sentinelcloud.com/cloudconnect/clientAlias/logout?version=1.0
```

2.6.3. Query Parameter

Parameter	Description	Valid Values	Optional
version	Version of Cloud Connect Web Services	1.0	No

2.6.4. HTTPS Method

POST

2.6.5. Request Message

```
<logoutRequest>

<sessionHandle>g0vfdYO81yjgIC%2BR57bvZ7cmxVvYfZJW9nbHb51BW%2Bd0nLvMj6l6YMtPl5c%3D</sessionHandle>
  <usageCountMultiplier>1</usageCountMultiplier>
  <machineId>hostName</machineId>
  <vendorId>a8e06c3</vendorId>
</logoutRequest>
```

Request Parameters

Parameter	Data Type	Description
sessionHandle	String	The session handle returned by the login Web service. This is used at the time of logout.
usageCountMultiplier	Integer (>0)	Stores the value of custom usage that you want to push in Cloud Connect database. This count specifies the number of executions consumed by a user during a session. ▪ Supported only for Count-based licenses. ▪ Not supported for Concurrent licenses. The usage count does not increase the concurrency count for a concurrent license.

Parameter	Data Type	Description
		▪ Postpaid licenses are supported to be eventually consistent (not always consistent). ▪ Since the usageCountMultiplier attribute is passed at the logout time, it is possible for the user to consume more counts than allowed. Suppose a user is allowed 10 counts, and he consumes counts in the following manner: 3 in first session, 3 in second session, and 8 in the last session. Here, the user consumed 14 counts in total, though only 10 were allowed. If the user renews the license, the additional number of counts consumed (4 in the above case) will be deducted in the next licenses term. For example, on license renewal with 10 counts, the previously consumed 4 counts will be deducted and the user will have 6 effective counts remaining. This way the actual license usage is taken into account eventually, though not always.
machineId	String	Host name.
vendorId	String	Unique identifier of the ISV.

2.6.6. Response Message

In case of success:

```
<logoutResponse>
   <status>Ok</status>
</logoutResponse>
```

In case of error:

```
<logoutResponse>
  <status>Fail</status>
  <errorCode>1001</errorCode>
  <errorDesc>error_message</errorDesc>
</logoutResponse>
```

2.6.7. Error Codes

Error Code	Description
1001	This web service version is not supported
1004	Invalid parameter: machineId
1007	Invalid parameter: vendorID
1008	Invalid request message
1010	Invalid web service URL

Error Code	Description
1011	The request XML is not well formed
1012	Not authorized to process any request
1013	Invalid parameter: sessionHandle
1014	Invalid parameter: usageCountMultiplier
1015	Internal error
1016	Error occurred in usage update
1027	Authentication Failed
1028	Authorization header not found
1029	x-sfnt-date header not found

2.7. refresh

Refreshes the concurrent sessions periodically.

2.7.1. Description

The refresh Web service performs the following functions:

- **Avoids premature termination of concurrent sessions**: This Web service refreshes the concurrent sessions periodically to keep the sessions alive for a longer period. Refreshing a session ensures that the background process does not kill the session after a specified time.
- **Notifies an ISV application about terminated sessions**: This Web service helps identify if a concurrent session has been terminated or is still valid. For example, If a session is killed by a background process or an ISV administrator, the Web service returns an error and ISV application does not allow the user to consume the feature, thus avoiding the license overuse.
- **Cleans abandoned sessions**: The background process checks for the presence of abandoned sessions and closes them. An abandoned session is one where the logout and refresh Web service calls do not reach Sentinel Cloud Connect, for example, in the case of browser crash. This way the background process cleans abandoned sessions and makes the concurrency available for consumption.
- **Helps in usage tracking**: The abandoned sessions are forced complete to ensure integrity of usage data.

For more details about abandoned sessions, refer to Section *Handling of Abandoned Sessions* of *Cloud Run-time Guide*.

2.7.2. Usage Notes

The licensed application should call this Web service periodically for refreshing sessions. For example, if the application allows online movie watching, then the application brings buffer only for certain duration, then it fetches next buffer for next duration, and so on.The application calls the refresh Web

service and fetches the next buffer only if the session is still valid. If a user exits the browser without proper logout or due to any reason, the refresh Web service will return failure, and the background process will auto complete the session according to the lastRefreshTimeStaleValue [Minutes] property and concurrency will get free.

You need to set certain configurations for background process so that it can identify abandoned sessions and forced them complete to free the concurrency.

The refresh Web service call:

- Is synchronous and is sent to Cloud Connect instantly.
- Is supported only for cloud.
- Functions only for the Subscription Concurrent license model.For other license models, this Web service returns success without updating the last refresh time.

2.7.3. Example URI

```
https://scc.na.sentinelcloud.com/cloudconnect/clientAlias/refresh?version=1.0
```

2.7.4. Query Parameter

Parameter	Description	Valid Values	Optional
version	Version of Cloud Connect Web Services	1.0	No

2.7.5. HTTPS Method

POST

2.7.6. Request Message

```
<refreshRequest>

<sessionHandle>g0vfd4zLk09AFOvNs5Rtl73TiqQpmoVAwyrHf%2F3ydo61Aos89JZUJ1XOiNW%2Bd0nLvMj6l6YMtPl5c%-
3D/sessionHandle>
  <machineId>hostName</machineId>
  <vendorId>a8e06c3</vendorId>
</refreshRequest>
```

Request Parameters

Parameter	Data Type	Description
sessionHandle	String	The session handle returned by the login Web service. This is used at the time of logout.
machineId	String	Host name
vendorId	String	Unique identifier of the ISV.

2.7.7. Response Message

```
<refreshResponse>
       <status>Ok</status>
</refreshResponse>
```

In case of error:

```
<refreshResponse>
  <status>Fail</status>
  <errorCode>1001</errorCode>
  <errorDesc>error_message</errorDesc>
</refreshResponse>
```

2.7.8. Error Codes

Error Code	Error Description
1001	This web service version is not supported
1004	Invalid parameter: machineId
1007	Invalid parameter: vendorID
1008	Invalid request message
1010	Invalid web service URL
1011	The request XML is not well formed
1012	Not authorized to process any request
1013	Invalid parameter: sessionHandle
1025	Session terminated
1015	Internal error
1026	Access denied to the requested feature
1027	Authentication Failed
1028	Authorization header not found
1029	x-sfnt-date header not found

3

Integrating Sentinel Cloud Connect Web Services

This section summarizes the steps required to integrate Cloud Connect Web services in your source code.

3.1. Prerequisites

Before you integrate Cloud Connect Web services in your application, you need to develop communication module, signature module, and message module as described in Workflow of Sentinel Cloud Connect Web Services.

3.2. Required Settings

You will need the following configuration information:

- **YPSAddress** : Address of the Cloud Directory Services.
- **ClientAlias**: Vendor name provided in the vendor registration request file.
- **Web service version**: This is set in URL as query parameter.
- **VendorId** : Unique identifier of the ISV.
- **Schema**: Request-response XML message schema.
- **CA Certificate**: Certificate issued by a well-known Certificate Authority.
- **Secret Key ID and Secret Key**: Used for authenticating requests.

3.3. Calling Cloud Connect Web Services

Following are the steps you need to follow to call Cloud Connect Web services:

1. Create a URL for the *register* Web service. Example: https://yps.na.sentinelcloud.com/YPServer/register?version=1.0.
2. Create a request message based on schema for the requested Web service. For example, the request message for *register* is:

```
<registerrequest>
   <vendorid>6c395aa</vendorid>
   <machineid>hostName</machineid>
```

```
   <nodedesc>ISV node for sample application</nodedesc>
   </vendorid>
</registerrequest>
```

3. Create a signature and set headers for the Web service, as specified in the section Authentication
4. Send request on complete Web service URL with POST request.

Use CA certificate for SSL communication.

5. Get the Cloud Connect address from *register* Web service response.
6. For the given ClientAlias, form complete Web service URL for subsequent Web service calls (getInfo, login, logout, or refresh). For example:

```
https://scc.na.sentinelcloud.com/cloudconnect/clientAlias/getInfo?version=1.0
```

7. Follow steps 2, 3, and 4 for the requested Web service, and process response.
8. Insert appropriate Web service calls in the application. The decision of which Web service calls to insert and where, depends on your authorization strategy.

4

Sample Codes

4.1. Sample Code - register

Here is the pseudo-code for `register` Web service in Python:

```
machine = 'test machine'


#
# load property file
#
props = scc_props.scc_props()
props.load(propertyFile)

#
# build data extarcting values from properties file
#


ypsUrl = props.getProperty('RuntimelessClientConfiguration', 'YPSAddress')
vendorId = props.getProperty('RuntimelessClientConfiguration', 'Vendor Id')
wsVersion = props.getProperty('RuntimelessClientConfiguration', 'WsVersion')
params = {'version': wsVersion}
key = props.getProperty('RuntimelessClientConfiguration', 'SecretKey')
keyId = props.getProperty('RuntimelessClientConfiguration', 'SecretKeyId')
registerUrl = props.getProperty('RuntimelessClientConfiguration', 'WsRegisterUrl')
nodeDescription = props.getProperty('RuntimelessClientConfiguration',"NodeDescription")
#
# build request
#
request = '<registerRequest>' +
       '<nodeDesc>' + nodeDescription + '</node-desc>' +
       '<machineId>' + machineId + '</machineId>' +
       '<vendorId>' + vendorId + '</vendorId>' +
'</registerRequest>'

result = scc_objects.CreateFromDocument(doPost(ypsUrl,registerUrl,request,key,keyId))
```

4.2. Sample Code - getInfo

Here is the pseudo-code for the `getInfo` Web service in Python:

```
machine = 'test machine'
```

```
#
# load property file
#
props = scc_props.scc_props()
props.load(propertyFile)

#
# extract values from properties file
#


vendorId = props.getProperty('RuntimelessClientConfiguration', 'Vendor Id')
wsVersion = props.getProperty('RuntimelessClientConfiguration', 'WsVersion')
params = {'version': wsVersion}
key = props.getProperty('RuntimelessClientConfiguration', 'SecretKey')
keyId = props.getProperty('RuntimelessClientConfiguration', 'SecretKeyId')
getInfoUrl= props.getProperty('RuntimelessClientConfiguration', 'WsGetInfoUrl')
clientAlias = props.getProperty('RuntimelessClientConfiguration', 'ClientAlias')

#
# build request:
#
# user,customerId and featureId must be existing valued precreated in EMS
# format has value 1

user = 'safenet_test'
customerId = 'safenet'
featureId = -1

request = '<getInfoRequest>' +
       '<user>' + user + '</user>' +
       '<customer>' + customer + '</customer>' +
       '<featureId>'+ str(featureId) + '</featureId>' +
       '<format>' + str(1) + '</format>' +
       '<vendorId>' + vendorId + '<vendorId/>' +
'</getInfoRequest>'
```

Result:

```
result = doPost(host,getInfoUrl,clientAlias,request,key,keyId)
```

4.3. Sample Code - login

Here is the pseudo-code for the `login` Web service in Python:

```
machine = 'test machine'


#
# load property file
#
props = scc_props.scc_props()
props.load(propertyFile)
```

```
#
# build data extarcting values from properties file
#


ypsUrl = props.getProperty('RuntimelessClientConfiguration', 'YPSAddress')
vendorId = props.getProperty('RuntimelessClientConfiguration', 'Vendor Id')
wsVersion = props.getProperty('RuntimelessClientConfiguration', 'WsVersion')
params = {'version': wsVersion}
key = props.getProperty('RuntimelessClientConfiguration', 'SecretKey')
keyId = props.getProperty('RuntimelessClientConfiguration', 'SecretKeyId')
loginUrl = props.getProperty('RuntimelessClientConfiguration', 'WsLoginUrl')
clientAlias = props.getProperty('RuntimelessClientConfiguration', 'ClientAlias')
vendorData = props.getProperty('RuntimelessClientConfiguration','VendorData')
#
# build request:
#
# user,customerId and featureId must be existing valued precreated in EMS
#

user = 'safenet_test'
customerId = 'safenet'
featureId = 1

request = '<loginRequest>' +
       '<user>' + user + '</user>' +
       '<customer>' + customerId + '</customer>' +
       '<featureId>' + str(featureId) + '</featureId>' +
       '<vendorData>' + vendorData + '</vendorData>' +
       '<machineId>+ machine + '</machineId>' +
       '<vendorId>' + vendorId + '<vendorId/>' +
'</loginRequest>'
```

Result

```
result = doPost(host,loginUrl,clientAlias,request,key,keyId)
```

The result contains sessionId that will be used further in the refresh and logout calls.

4.4. Sample Code - logout

Here is the pseudo-code for the `logout` Web service in Python:

```
machine = 'test machine'


#
# load property file
#
props = scc_props.scc_props()
props.load(propertyFile)

#
# build data extarcting values from properties file
#
```

```
vendorId = props.getProperty('RuntimelessClientConfiguration', 'Vendor Id')
wsVersion = props.getProperty('RuntimelessClientConfiguration', 'WsVersion')
params = {'version': wsVersion}
key = props.getProperty('RuntimelessClientConfiguration', 'SecretKey')
keyId = props.getProperty('RuntimelessClientConfiguration', 'SecretKeyId')
logoutUrl = props.getProperty('RuntimelessClientConfiguration', 'WsLogoutUrl')
clientAlias = props.getProperty('RuntimelessClientConfiguration', 'ClientAlias')

#
# build request:
#
# sessionId is the one previously saved at login time
#

request = '<logoutRequest>' +
       '<sessionHandle>' + sessionId + '</sessionHandle>' +
       '<usageCountMultiplier>' + str(usageMultiplier) + '</usageCountMultiplier>' +
       '<machineId>' + machine '</machineId>' +
       '<vendorId>' + vendorId + '</vendorId>' +
'</logoutRequest>'
```

Result

```
result = doPost(host,logoutUrl,clientAlias,request,key,keyId)
```

In the result, status has value **OK** on success or **Fail** in case of error. In case of failure, the error code is also returned.

4.5. Sample Code - refresh

Here is the pseudo-code for `refresh` Web service in Python:

```
#
# load property file
#
props = scc_props.scc_props()
props.load(propertyFile)

#
# build data extarcting values from properties file
#


ypsUrl = props.getProperty('RuntimelessClientConfiguration', 'YPSAddress')
vendorId = props.getProperty('RuntimelessClientConfiguration', 'Vendor Id')
wsVersion = props.getProperty('RuntimelessClientConfiguration', 'WsVersion')
params = {'version': wsVersion}
key = props.getProperty('RuntimelessClientConfiguration', 'SecretKey')
keyId = props.getProperty('RuntimelessClientConfiguration', 'SecretKeyId')
refreshUrl = props.getProperty('RuntimelessClientConfiguration', 'WsRefreshUrl')
clientAlias = props.getProperty('RuntimelessClientConfiguration', 'ClientAlias')
vendorData = props.getProperty('RuntimelessClientConfiguration','VendorData')
#
# build request:
#
```

```
# sessionId must be one saved previously during a login
#


request = '<refreshRequest>' +
       '<sessionHandle>' + sessionId + '</sessionHandle>' +
       '<machineId>' + machine + '</machineId>' +
       '<vendorId>' + vendorId + '<vendorId/>' +
'</refreshRequest>'
```

Result

```
result = doPost(host,refreshUrl,clientAlias,request,key,keyId)
```

The status field is set to **OK** if the session is still valid or **Fail** if not valid anymore. The error code is also returned.

5 Troubleshooting

This section describes solutions to common problems you may encounter when using the Cloud Connect Web Services:

5.1. Error in SSL Communication

This error may occur when:

- Invalid CA certificate is used for communication with Sentinel Cloud Connect or Directory Services.
- The address of Directory Services server or Cloud Connect is incorrect.
- Directory Services server or Cloud Connect is down.

5.2. Authentication Failure

The possible cause of this error is:

- Authorization header is not set with request as per specification. Please refer section Headers to verify if the correct header is set or not.
- SecretKeyID is wrong.
- Incorrect secret is used in signature calculation.
- Timestamp is not correctly set in the x-sfnt-date header.
- Web service request is older than 15 minutes.

5.3. VendorID Not Supported

This error occurs when no instance of the Cloud Connect, corresponding to the given vendor ID, is up. To resolve this condition, ensure that:

- VendorID is correct (as shared by SafeNet).
- Cloud Connect is up and running.
- Get the new Cloud connect URL by calling register Web service.

5.4. Invalid XML

All the request messages should follow the schema (*.xsd*) given with the installation. Please ensure that request XML is according to schema.

6

Request/Response Schema for Cloud Connect Web Services

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:jxb="http://java.sun.com/xml/ns/jaxb" version="1.0" jxb:version="2.0">

	<xsd:element name="registerRequest" type="registerRequest"/>
	<xsd:element name="registerResponse" type="registerResponse"/>
	<xsd:element name="loginRequest" type="loginRequest"/>
	<xsd:element name="loginResponse" type="loginResponse"/>
	<xsd:element name="logoutRequest" type="logoutRequest"/>
	<xsd:element name="logoutResponse" type="logoutResponse"/>
	<xsd:element name="refreshRequest" type="refreshRequest"/>
	<xsd:element name="refreshResponse" type="refreshResponse"/>
	<xsd:element name="getInfoRequest" type="getInfoRequest"/>
	<xsd:element name="getInfoResponse" type="getInfoResponse"/>


	<xsd:complexType name="registerRequest">
		<xsd:sequence>
			<xsd:element name="vendorId" type="xsd:string" />
			<xsd:element name="machineId" type="xsd:string" minOccurs="0"/>
			<xsd:element name="nodeDesc" type="xsd:string" />
		</xsd:sequence>
	</xsd:complexType>

	<xsd:complexType name="registerResponse">
		<xsd:sequence>
			<xsd:element name="status" type="xsd:string"/>
			<xsd:element name="urlList" type="urlList" minOccurs="0"/>
			<xsd:element name="errorCode" type="xsd:positiveInteger" minOccurs="0"/>
			<xsd:element name="errorDesc" type="xsd:string" minOccurs="0"/>
		</xsd:sequence>
	</xsd:complexType>

	<xsd:complexType name="urlList">
		<xsd:sequence>
			<xsd:element name="url" type="url" maxOccurs="unbounded"/>
		</xsd:sequence>
	</xsd:complexType>

	<xsd:complexType name="url">
		<xsd:attribute name="value" type="xsd:string" />
	</xsd:complexType>

	<xsd:complexType name="loginRequest">
			<xsd:sequence>
			<xsd:element name="user" type="xsd:string" />
			<xsd:element name="customer" type="xsd:string" minOccurs="0"/>
```

```
                <xsd:element name="featureId" type="xsd:int" />
                <xsd:element name="vendorData" type="xsd:string" minOccurs="0"/>
                <xsd:element name="machineId" type="xsd:string" />
                <xsd:element name="vendorId" type="xsd:string" />
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="loginResponse">
        <xsd:sequence>
                <xsd:element name="status" type="xsd:string"/>
                <xsd:element name="sessionHandle" type="xsd:string" minOccurs="0"/>
                <xsd:element name="errorCode" type="xsd:positiveInteger" minOccurs="0"/>
                <xsd:element name="errorDesc" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="logoutRequest">
        <xsd:sequence>
                <xsd:element name="sessionHandle" type="xsd:string"/>
                <xsd:element name="usageCountMultiplier" type="xsd:positiveInteger" minOccurs="0"/>
                <xsd:element name="machineId" type="xsd:string" />
                <xsd:element name="vendorId" type="xsd:string" />
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="logoutResponse">
        <xsd:sequence>
                <xsd:element name="status" type="xsd:string"/>
                <xsd:element name="errorCode" type="xsd:positiveInteger"  minOccurs="0" />
                <xsd:element name="errorDesc" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="refreshRequest">
        <xsd:sequence>
                <xsd:element name="sessionHandle" type="xsd:string"/>
                <xsd:element name="machineId" type="xsd:string" />
                <xsd:element name="vendorId" type="xsd:string" />
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="refreshResponse">
        <xsd:sequence>
                <xsd:element name="status" type="xsd:string"/>
                <xsd:element name="errorCode" type="xsd:positiveInteger" minOccurs="0" />
                <xsd:element name="errorDesc" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="getInfoRequest">
        <xsd:sequence>
                <xsd:element name="user" type="xsd:string" />
                <xsd:element name="customer" type="xsd:string" minOccurs="0" />
                <xsd:element name="featureId" type="xsd:int"/>
                <xsd:element name="productName" type="xsd:string" minOccurs="0"/>
                <xsd:element name="entitlementId" type="xsd:string" minOccurs="0"/>
                <xsd:element name="format" type="xsd:int" />
                <xsd:element name="vendorId" type="xsd:string" />
        </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="getInfoResponse">
        <xsd:sequence>
                <xsd:element name="status" type="xsd:string"/>
                <xsd:element name="entitlements" type="entitlements" minOccurs="0"/>
                <xsd:element name="errorCode" type="xsd:positiveInteger" minOccurs="0"/>
                <xsd:element name="errorDesc" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="entitlements">
        <xsd:sequence>
                <xsd:element name="entitlement" type="entitlement" maxOccurs="unbounded"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="entitlement">
        <xsd:sequence>
                <xsd:element name="products" type="products"/>
        </xsd:sequence>
        <xsd:attribute name="entitlementId" type="xsd:string" />
        <xsd:attribute name="timeZone" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="products">
<xsd:sequence>
        <xsd:element name="product" type="product" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="product">
        <xsd:sequence>
                <xsd:element name="features" type="features"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="features">
        <xsd:sequence>
                <xsd:element name="feature" type="feature" maxOccurs="unbounded"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="feature">
        <xsd:sequence>
                <xsd:element name="notifications" type="xsd:int" minOccurs="0"/>
                <xsd:element name="license" type="license" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" />
        <xsd:attribute name="name" type="xsd:string" />
        <xsd:attribute name="usable" type="xsd:string" />
        <xsd:attribute name="usabilityStatus" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="license">
        <xsd:sequence>
                <xsd:element name="licenseModelType" type="xsd:string"/>
                <xsd:element name="licenseAttributes" type="licenseAttributes" maxOccurs="unbounded"/>
        </xsd:sequence>
</xsd:complexType>
```

```
	<xsd:complexType name="licenseAttributes">
		<xsd:sequence>
			<xsd:element name="attribute" type="attribute" maxOccurs="unbounded"/>
		</xsd:sequence>
	</xsd:complexType>

	<xsd:complexType name="attribute">
		<xsd:attribute name="name" type="xsd:string" />
		<xsd:attribute name="value" type="xsd:string" />
	</xsd:complexType>

</xsd:schema>
```

7

Error Codes

Given below is a master list of the Cloud Connect Web services error codes:

Error Code	Description
1001	This web service version is not supported
1002	Invalid parameter: user
1003	Invalid parameter: Customer
1004	Invalid parameter: machineId
1005	Invalid parameter: featureId
1006	Invalid parameter: vendorData
1007	Invalid parameter: vendorID
1008	Invalid request message
1009	Request from this vendorID is not supported
1010	Invalid web service URL
1011	The request XML is not well formed
1012	Not authorized to process any request
1013	Invalid parameter: sessionHandle
1014	Invalid parameter: usageCountMultiplier
1015	Internal error
1016	Error occurred in usage update
1017	License is not in active state
1018	License is expired
1019	License is disabled
1020	License is revoked
1021	Maximum concurrent user limit reached
1022	Maximum usage count reached
1023	License does not exist or license is not in active state
1024	Invalid parameter
1025	Session terminated
1026	Access denied to the requested feature

Error Code	Description
1027	Authentication Failed
1028	Authorization header not found
1029	x-sfnt-date header not found
1030	Invalid parameter: format
1031	No authorization server mapped to given vendor Id
1032	Invalid parameter: productName
1033	Invalid parameter: entitlementId

8

Handling of Abondened Sessions

An abandoned session is one where the following calls do not reach Sentinel Cloud Connect:

- logout
- refresh

In the case of concurrent license model, if refresh or logout does not reach Cloud Connect for a specified interval, the concurrent session is abandoned.

In the case of license models except concurrent, if logout does not reach Cloud Connect for a specified interval,the session is treated as abandoned.

You need to set certain configurations for background process so that it can identify abandoned sessions and force them complete to free the concurrency.

For more details about abandoned sessions and related configurations, refer to Section *Handling of Abandoned Sessions* of *Cloud Run-time Guide*.

Index