Oracle® Fusion Middleware

REST API for Oracle Platform Security Services

12c (12.2.1.4.0)
F11958-01
September 2019

Oracle Fusion Middleware REST API for Oracle Platform Security Services, 12c (12.2.1.4.0)

F11958-01

Contents

4 Managing Keystores

5 Creating and Validating Trust Tokens

6 Authorizing Access

Preface

This preface describes the document accessibility features and conventions used in this guide—*REST API for Oracle Platform Security Services*.

Audience

This document is intended for software developers and architects who are interested in using Oracle Platform Security Services (OPSS) through a RESTful API. The audience must already be familiar with OPSS to use this guide.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Platform Security Services documentation set:

- *Release Notes for Oracle Platform Security Services*
- *Securing Applications with Oracle Platform Security Services*
- *WLST Command Reference for Infrastructure Security*
- *Java API Reference for Oracle Platform Security Services*
- *Java API Reference for Oracle Platform Security Services MBeans*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New In This Guide

This section summarizes the new features and significant product changes for Oracle Platform Security Services (OPSS) in Oracle Fusion Middleware 12*c* (12.2.1.x).

New and Changed Features for 12c (12.2.1.4.0)

In release 12.2.1.4, the changes to this document include:

The removal of the `key password` parameter from the GET Trusted Certificate KSS Keystore Method.

New and Changed Features for 12c (12.2.1.3.0)

This document does not contain any new or changed features.

New and Changed Features for 12c (12.2.1.2.0)

In release 12.2.1.2.0, the changes to this document include:

Minor corrections throughout the document, which includes the addition of DELETE Credential Using Resource ID.

New and Changed Features for 12c (12.2.1)

Oracle Platform Security Services 12*c* (12.2.1) includes the following new and changed features for this document.

- Registration service RESTful API, which provides REST clients with the ability to register with the security platform.
- Credentials service RESTful API, which provides REST clients with the ability to use the Credential Store Framework (CSF) to manage credentials in a secure form. See Managing Credentials in the Credential Store .
- Keystore service RESTful API, which provides REST clients with the ability to use the Keystore Service (KSS) to view and manage keystores. See Managing Keystores .
- Trust service RESTful API, which provides REST clients with the ability to manage trust tokens. See Creating and Validating Trust Tokens.
- Authorization service RESTful API, which provides REST clients with the ability to manage XACML3.0 REST profile authorization. See Authorizing Access .

1

About the OPSS REST API

Oracle Platform Security Services (OPSS) provides a representational state transfer (REST) API for managing OPSS services including registration, credentials, keystore, trust, and authorization.
This chapter includes the following sections:

Introducing the OPSS REST API

The OPSS REST API provides access to core OPSS functionality over a REST interface. The REST API enables a wider range of languages and platforms to use OPSS services. The API also provides applications with the flexibility to use newer functionality without having to wait for the corresponding language-specific APIs to be implemented.

The services discussed in this reference include:

- Registration Service – A service that is used to register a client with OPSS. A client must register with OPSS in order to use any of the other services. See Registering OPSS Clients .
- Credentials Service – A service that is used to create and view credentials. See Managing Credentials in the Credential Store .
- Keystore Service – A service that is used to manage keysores. See Managing Keystores .
- Trust Service – A service that is used to create and validate trust tokens. See Creating and Validating Trust Tokens.
- Authorization Service – A service that is used to authorize access to resources using a policy decision point system. See Authorizing Access .

> **Note:**
>
> To deploy OPSS REST API services, your domain must include the OPSS REST Service Application Template. You can select this template when creating your domain, or you can extend an existing domain to include it. For more information, see:
>
> – Configuring Fusion Middleware Domains in *Creating WebLogic Domains Using the Configuration Wizard*
>
> – Oracle OPSS REST Service Application Template in *Domain Template Reference*

General URL Structure for OPSS Resources

You use a specific URL structure to manage OPSS resources.

Use the following URL to manage OPSS security:

`https://host:port/opss/v2/resource`

Where:

- *host*:*port*—Host and port where Oracle Fusion Middleware is running.
- *resource*—Relative path that defines the REST resource. Available resources are described throughout this guide. To access the Web Application Definition Language (WADL) document which defines each of the resources, specify `application.wadl` in the URL. For example:

  `https://host:port/opss/v2/application.wadl`

Authenticating REST Resources

You access the Oracle Fusion Middleware REST resources over HTTP and must provide your Oracle WebLogic Server administrator user name and password for authentication.

For example, to authenticate using cURL, pass the user name and password using the `-u` cURL option.

```
curl -i -X GET -u username:password https://myhost:7001/opss/v2/keystore
```

For `GET` and `DELETE` methods, which do not send data in the request body, if a keystore or key is password-protected, you must pass the Base64-encrypted keystore and key passwords, respectively, in custom headers. For example:

```
curl -i -X DELETE -u username:password -H keystorePassword:cHdkMQ== -H
keyPassword:bXlQd2Qy  https://myhost:7001/opss/v2/
keystoreservice?"stripeName=myStripe&keystoreName=myKeystore"
```

Using HTTP Methods with OPSS REST

The OPSS REST endpoints support standard HTTP semantics.

REST Method	Task
GET	Retrieve information about the REST resource.
POST	Add a REST resource.
PUT	Update a REST resource.
DELETE	Delete a REST resource.

HTTP Status Codes for HTTP Methods

OPSS REST HTTP methods return standard HTTP response status codes.

The HTTP methods used to manipulate the OPSS resources described in this guide return one of the following HTTP status codes:

HTTP Status Code	Description
200 OK	The request was successfully completed. A 200 status is returned for successful GET or POST method.
201 Created	The request has been fulfilled and resulted in a new resource being created. The response includes a Location header containing the canonical URI for the newly created resource. A 201 status is returned from a synchronous resource creation or an asynchronous resource creation that completed before the response was returned.
202 Accepted	The request has been accepted for processing, but the processing has not been completed. The request may or may not eventually be acted upon, as it may be disallowed at the time processing actually takes place. When specifying an asynchronous (__detached=true) resource creation (for example, when deploying an application), or update (for example, when redeploying an application), a 202 is returned if the operation is still in progress. If __detached=false, a 202 may be returned if the underlying operation does not complete in a reasonable amount of time. The response contains a Location header of a job resource that the client should poll to determine when the job has finished. Also, returns an entity that contains the current state of the job
400 Bad Request	The request could not be processed because it contains missing or invalid information (such as, a validation error on an input field, a missing required value, and so on).
401 Unauthorized	The request is not authorized. The authentication credentials included with this request are missing or invalid.
403 Forbidden	The user cannot be authenticated. The user does not have authorization to perform this request.
404 Not Found	The request includes a resource URI that does not exist.
405 Method Not Allowed	The HTTP verb specified in the request (DELETE, GET, POST, PUT) is not supported for this request URI.
406 Not Acceptable	The resource identified by this request is not capable of generating a representation corresponding to one of the media types in the Accept header of the request. For example, the client's Accept header request XML be returned, but the resource can only return JSON.
415 Not Acceptable	The client's ContentType header is not correct (for example, the client attempts to send the request in XML, but the resource can only accept JSON).

HTTP Status Code	Description
500 Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
503 Service Unavailable	The server is unable to handle the request due to temporary overloading or maintenance of the server. The Oracle WSM REST web application is not currently running.

2
Registering OPSS Clients

Oracle Platform Security Services (OPSS) uses the Registration service to provision an authorization policy for a client. The Security service uses these policies to make authorization decisions. REST clients are required to register themselves to access security services.

Section	Method	Resource Path
POST Registration Method	POST	/opss/v2/
GET Registration Method	GET	/opss/v2/
PUT Registration Method	PUT	/opss/v2/
DELETE Registration Method	DELETE	/opss/v2/

POST Registration Method

Use the POST method to register a new client. An application role with a unique name inside the OPSS rest application stripe is created. Users and groups that are passed as input of the POST method are made members of the application role. Grants to the specified resources are automatically provisioned in the OPSS REST application stripe.

> **Note:**
>
> The same `clientName` attribute value is required to identify the client when making changes to registration data.

REST Request

```
POST /opss/v2/opssRestClient/
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the details of the register request:

Table 2-1 Registration Attributes

Attribute	Description	Required
“clientName“	Unique name that identifies the client.	Yes
“policystoreStripe“	Policy store stripe to which the client is assigned.	No

Table 2-1 (Cont.) Registration Attributes

Attribute	Description	Required
"keystore"	List of keystores used for the client.	No
"credentialMap"	Name of the credential map that is used to store credential keys.	No
"auditComponent"	Unique name to identify the audit rules for a client.	No
"trustIssueIDD"	List of identity domains that can issue trust tokens	No
"trustValidateIDD"	List identity domains that can validate trust tokens	No
"adminGroup"	Group with the administrator role.	No
"operatorGroup"	Group with the operator role.	No
"viewerGroup"	Group with the viewer role.	No

All attributes other than `clientName` can be specified multiple times. A user should specify at least one of either: `policystoreStripe`, `keystore`, `credentialMap`, `auditComponent`, `trustIssueIDD`, or `trustValidateIDD` for the service scopes. In addition, a user should specify at least one of either: `adminGroup`, `operatorGroup`, or `viewerGroup` so that some group has privileges.

For service scope attributes, a wild card (*) can be specified to grant all scopes to the client. The wildcard should be used carefully.

Response Body

The output of a POST request is a Resource ID.

cURL Example

The following example shows how to register a client by submitting a POST request on the REST resource using cURL

```
curl -i -X POST -u username:password --data @register.json
   -H Content-Type:application/json https://myhost:7001/opss/v2/opssRestClient
```

Example of Request Body

The following shows an example of the request body in JSON format.

```
{
    "clientName": "myClientName",
    "policystoreStripe": "CRM",
    "keystore": ["appA", "appB/store1"],
    "credentialMap": "mapA",
    "auditComponent": "myComponent",
    "trustIssueIDD" : ["cisco", "intel"],
    "trustValidateIDD" : ["cisco", "intel"],
    "adminGroup":"myGroup1",
    "operatorGroup":"myGroup2",
    "viewerGroup":"myGroup3"
}
```

GET Registration Method

Use the GET method to view the client attributes for a registered client.

REST Request

`GET /opss/v2/opssRestClient/clientName`

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains the client registration attributes. For details about the registration attributes, see Table 2-1.

cURL Example

The following example shows how to view the registered client by submitting a GET request on the REST resource using cURL

```
curl -i -X GET -u username:password https://myhost:7001/opss/v2/opssRestClient/
   myClientName
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "clientName": "myClientName",
    "policystoreStripe": "CRM",
    "keystore": ["appA", "appB/store1"],
    "credentialMap": "mapA",
    "auditComponent": "myComponent",
    "trustIssueIDD" : ["cisco", "intel"],
    "trustValidateIDD" : ["cisco", "intel"],
    "adminGroup":"myGroup1",
    "operatorGroup":"myGroup2",
    "viewerGroup":"myGroup3"
}
```

PUT Registration Method

Use the PUT method to update the attributes of a registered client.

REST Request

`PUT /opss/v2/opssRestClient/clientName`

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the client registration attributes. For details about the registration attributes, see Table 2-1.

cURL Example

The following example shows how to update client attributes by submitting a PUT request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @register.json
   -H Content-Type:application/json https://myhost:7001/opss/v2/opssRestClient/
   myClientName
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "clientName": "myClientName",
    "policystoreStripe": "CRM",
    "keystore": ["appA", "appB/store1"],
    "credentialMap": "mapA",
    "auditComponent": "myComponent",
    "trustIssueIDD" : ["cisco", "intel"],
    "trustValidateIDD" : ["cisco", "intel"],
    "adminGroup":"myGroup1",
    "operatorGroup":"myGroup2",
    "viewerGroup":"myGroup3"
}
```

DELETE Registration Method

Use the DELETE method to remove a registered client.

REST Request

`DELETE /opss/v2/opssRestClient/`*`clientName`*

cURL Example

The following example shows how to delete a delete a registered client by submitting a DELETE request on the REST resource using cURL.

```
curl -i -X DELETE -u username:password https://myhost:7001/opss/v2/opssRestClient/
   myClientName
```

3

Managing Credentials in the Credential Store

Oracle Platform Security Services (OPSS) uses the Credential Store Framework (CSF) to manage credentials in a secure form. You can view and manage credentials in the store using REST.

Section	Method	Resource Path
POST Credentials Method	`POST`	`/opss/v2/credentials`
GET Credentials Using Map and Key Method	`GET`	`/opss/v2/credentials/`
GET Credentials Using Map Method	`GET`	`/opss/v2/credentials`
GET Credential Using Resource ID	`GET`	`/opss/v2/credentials/`*`resourceId`*
PUT Credential Using Resource ID	`PUT`	`/opss/v2/credentials/`*`resourceId`*
DELETE Credential Using Resource ID	`DELETE`	`/opss/v2/credentials/`*`resourceId`*

POST Credentials Method

Use the POST method to create new credentials in the credential store.

REST Request

```
POST /opss/v2/credentials
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the details of the create request:

Table 3-1 Credentials Attributes

Attribute	Description	Required
`“userName“`	Username for the credential.	Yes
`“password“`	Password for the credential.	Yes
`“description“`	Description for the credential.	Optional
`“expiration“`	Expiration date for the credential formatted as `yyyy-MM-dd' T'HH:mm:ss.SSSZ`.	Optional
`“namespace“`	Unique name for the credential namespace.	Yes

Table 3-1 (Cont.) Credentials Attributes

Attribute	Description	Required
"name"	Unique name that identifies the credential.	Yes

Response Body

The output of a POST request is a Resource ID.

cURL Example

The following example shows how to create a credential in the credential store by submitting a POST request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @createcred.json -H Content-
Type:application/json https://myhost:7001/opss/v2/credentials
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "userName": "myUser3",
    "password": "mypass123",
    "description": "mydescription",
    "expiration": "5000-07-04T12:08:56.235-0700",
    "namespace: "MyMap",
    "name":"myKey"
}
```

GET Credentials Using Map and Key Method

Use the GET method to search the entire CSF for a credential given its map and key name.

REST Request

```
GET /opss/v2/credentials
```

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains attributes for the credential. For details about credential attributes, see Table 3-1.

cURL Example

The following example shows how to view credentials in a credential store by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password https://myhost:7001/idaas/platform/
   opss/v2/credentials?filter=map=mymap,key=mykey
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "id": "1234567890"
    "userName": "myUser3",
    "password": "mypass123",
    "description": "mydescription",
    "expiration": "5000-07-04T12:08:56.235-0700",
    "type": "PasswordCredential"
}
```

GET Credentials Using Map Method

Use the GET method to search the entire CSF for a list of credentials given a `map` name.

> **Note:**
>
> If a map contains generic credentials, then it will not be present in the list.

REST Request

```
GET /opss/v2/credentials
```

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains attributes for the credentials. For details about credential attributes, see Table 3-1.

cURL Example

The following example shows how to view credentials in a credential store by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password https://myhost:7001/opss/v2/credentials?
   filter=map=mymap
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "credentials": [
        {
            "id": "1234567890",
            "userName": "myUser",
            "password": "mypass123",
            "description": "mydescription",
            "expiration": "5000-07-04T12:08:56.235-0700",
            "type": "PasswordCredential"
        },
        {
            "id": "1234567890",
            "userName": "myUser2",
            "password": "mypass123",
            "description": "mydescription",
            "expiration": "5000-07-04T12:08:56.235-0700",
            "type": "PasswordCredential"
        }
    ]
}
```

GET Credential Using Resource ID

Use the GET method to search the entire CSF for a credential given its Resource ID.

REST Request

`GET /opss/v2/credentials/`*`resourceId`*

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains attributes for the credential. For details about credential attributes, see Table 3-1.

cURL Example

The following example shows how to view credentials in a credential store by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password https://myhost:7001/opss/v2/credentials/
   1234567890
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "id": "1234567890"
```

```
    "userName": "myUser3",
    "password": "mypass123",
    "description": "mydescription",
    "expiration": "5000-07-04T12:08:56.235-0700",
    "type": "PasswordCredential"
}
```

PUT Credential Using Resource ID

Use the PUT method to update an existing credential in the credential store. The entry must exist for the operation to succeed.

REST Request

`PUT /opss/v2/credentials/resourceId`

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the details of the create request.

Table 3-2 Credentials Attributes

Attribute	Description	Required
“userName“	Username for the credential.	No
“password“	Password for the credential.	No
“description“	Description for the credential.	No
“expiration“	Expiration date for the credential formatted as yyyy-MM-dd' T'HH:mm:ss.SSSZ.	No
“namespace“	“myMap4”	No
“name“	“myKey22”	No

cURL Example

The following example shows how to replace an existing credential in the credential store by submitting a PUT request on the REST resource using cURL.

```
curl -i -X PUT -u username:password --data @replacecred.json -H Content-
Type:application/json https://myhost:7001/opss/v2/credentials
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "userName": "myUser3",
    "password": "mypass123",
    "description": "mydescription",
    "expiration": "5000-07-04T12:08:56.235-0700",
```

DELETE Credential Using Resource ID

Use the DELETE method to remove the entire CSF for a credential given its Resource ID. The entry must exist for the operation to succeed.

REST Request

```
DELETE /opss/v2/credentials/resourceId
```

cURL Example

The following example shows how to delete a credential from a credential store by submitting a DELETE request on the REST resource using cURL.

```
curl -i -X DELETE -u username:password https://myhost:7001/opss/v2/credentials/
   1234567890
```

4

Managing Keystores

Oracle Platform Security Services (OPSS) uses the Keystore Service (KSS) to view and manage keystores. You can view and manage keystores using a set of REST resources.

Section	Method	Resource Path
POST New KSS Keystore Method	POST	/opss/v2/keystoreservice
POST Import KSS Keystore Method	POST	/opss/v2/keystoreservice/keystore
PUT Password Update KSS Keystore Method	PUT	/opss/v2/keystoreservice
POST Trusted Certificate KSS Keystore Method	POST	/opss/v2/keystoreservice/certificates
GET Stripe KSS Keystores Method	GET	/opss/v2/keystoreservice/{stripeName}
GET Alias KSS Keystore Method	GET	/opss/v2/keystoreservice/alias/{stripeName}/{keystoreName}/{entryType}
GET Trusted Certificate KSS Keystore Method	GET	/opss/v2/keystoreservice/certificates
DELETE Trusted Certificate KSS Keystore Method	DELETE	/opss/v2/keystoreservice/certificates
POST Secret Key KSS Keystore	POST	/opss/v2/keystoreservice/secretkey
GET Secret Key Properties KSS Keystore Method	GET	/opss/v2/keystoreservice/secretkey
DELETE Secret Key KSS Keystore Method	DELETE	/opss/v2/keystoreservice/secretkey
POST Key Pair KSS Keystore	POST	/opss/v2/keystoreservice/keypair
GET Key Pair KSS Keystore Method	GET	/opss/v2/keystoreservice/keypair
DELETE Key Pair KSS Keystore Method	DELETE	/opss/v2/keystoreservice/keypair
DELETE Keystore Service KSS Keystore Method	DELETE	/opss/v2/keystoreservice

POST New KSS Keystore Method

Use the POST method to create a new Keystore Service (KSS) Keystore.

REST Request

```
POST /opss/v2/keystoreservice
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the details of the create request:

Attribute	Description	Required
"stripeName"	Name of the stripe to contain the KSS keystore.	Yes
"keystoreName"	Name for the KSS keystore.	Yes
"keystorePassword"	Password for the KSS keystore.	No
"permissionBased"	Boolean value that specifies whether to create a permission-based keystore.	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the create operation, including:

Attribute	Description
"ERROR_CODE"	If "STATUS" is set to "Failed", provides the error code.
"ERROR_MSG"	If "STATUS" is set to "Failed", provides the contents of the error message.
"STATUS"	Status of operation. For example, "SUCCEEDED" or "FAILED".

cURL Example

The following example shows how to create a KSS keystore by submitting a POST request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @createkss.json -H Content-
Type:application/json https://myhost:7001/opss/v2/keystoreservice
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "stripeName" : "myStripe",
    "keystoreName" : "myKeystore",
    "keystorePassword" : "myPwd",
    "permissionBased" : "false"
}
```

> **Note:**
>
> A password is required unless creating a permission-based keystore ("permission" : "true").

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "STATUS": "SUCCEEDED"
}
```

Example of Creating Permission-Based Keystore

The following example shows the request body in JSON format.

```
{
    "stripeName" : "myStripe",
    "keystoreName" : "permKeystore",
    "permissionBased" : "true"
}
```

POST Import KSS Keystore Method

Use the POST method to import a Keystore Service (KSS) keystore from a JKS keystore file.

REST Request

```
POST /opss/v2/keystoreservice/keystore
```

Request Body

Media types for the request or response body.

Media Types: `multipart/form-data`

The response body contains information about the import request, including:

Attribute	Description	Required
`"stripeName"`	Name of the stripe.	Yes
`"keystoreImportByte"`	Byte array of keystore data.	Yes
`"keystoreName"`	Name for the JKS keystore.	Yes
`"keystorePassword"`	Password for the local keystore file that is being imported and the keystore entry, if password-protected.	No
`"keystoreType"`	Keystore type. This value must be set to JKS.	Yes
`"keyAliasList"`	List of aliases for the keys to be imported from the `keystoreFile`.	Yes
`"keyPasswordList"`	List of passwords for the keys to be imported from the `keystoreFile`.	No

Attribute	Description	Required
"permissionBased"	Boolean value that specifies whether to import as a permission-based keystore.	No

Response Body

Media types for the request or response body.

Media Types: application/json

The response body contains information about the import operation, including:

Attribute	Description
"ERROR_CODE"	If "STATUS" is set to "Failed", provides the error code.
"ERROR_MSG"	If "STATUS" is set to "Failed", provides the contents of the error message.
"SUCCES MSGS"	Success message.

cURL Example

The following example shows how to import a KSS keystore by submitting a POST request on the REST resource using cURL.

```
curl -i -X PUT -u username:password --data @updatekss.json -H
Content-Type:application/json https://myhost:7001/opss/v2/keystoreservice/keystore
```

Example of Request Body

The following example shows the request body in JSON format.

```
"stripeName" : "myStripe",
"keystoreName" : "myKeystore",
"keyAliasList" : ["myAlias"],
"keystorePassword" : "password1",
"keyPasswordList" : ["password"],
"keystoreType" : "JKS",
"permissionBased" : "false",
"keystoreImportBytes" : [-2, -19, -2, -19, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 2, 0, 6,
109, 121, 99, 101, 114, 116, 0, 0, 1, 86, 125, 119, -27, 113, 0, 5, 88, 46, 53, 48,
57, 0, 0, 3, -61, 48, -126, 3, -65, 48, -126, 2, -89, -96, 3, 2, 1, 2, 2, 16, 64, 4,
72, -122, -60, 65, -17, 59, 100, 58, -128, 102, 64, -102, -4, -96, 48, 13, 6, 9, 42,
-122, 72, -122, -9, 13, 1, 1, 11, 5, 0, 48, 120, 49, 11, 48, 9, 6, 3, 85, 4, 6, 19,
2, 85, 83, 49, 16, 65, -117, -74]
}
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "SUCCESS_MSG":"Aliases: myAlias imported
    successfully"
}
```

PUT Password Update KSS Keystore Method

Use the PUT method to update the password for a Keystore Service (KSS) keystore.

REST Request

```
PUT /opss/v2/keystoreservice
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains information about the Load Balancer patches, including:

Attribute	Description	Required
“stripeName”	Name of the stripe.	Yes
“keystoreName”	Name of the KSS keystore.	Yes
“newPassword”	New password for the keystore.	Yes
“oldPassword”	Old password for the keystore.	Yes

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the update operation, including:

Attribute	Description
“ERROR_CODE”	If “STATUS” is set to "Failed", provides the error code.
“ERROR_MSG”	If “STATUS” is set to “Failed”, provides the contents of the error message.
“STATUS”	Status of operation. For example, “SUCCEEDED” or “FAILED”.

cURL Example

The following example shows how to import a KSS keystore by submitting a PUT request on the REST resource using cURL.

```
curl -i -X PUT -u username:password --data @updatekss.json -H Content-
Type:application/json https://myhost:7001/opss/v2/keystoreservice
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "stripeName" : "myStripe",
    "keystoreName" : "mykssstore",
    "oldPassword" : "myPwd",
    "newPassword" : "myNewPwd"
}
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "STATUS": "SUCCEEDED"
}
```

POST Trusted Certificate KSS Keystore Method

Use the POST method to import a trusted certificate into a Keystore Service (KSS) keystore.

REST Request

```
POST /opss/v2/keystoreservice/certificates
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains information about the import request, including:

Attribute	Description	Required
`"keyAliasList"`	List that contains alias for the trusted certificate.	Yes
`"keystoreEntry"`	Base64-encoded certificate.	Yes
`"keystoreType"`	Keystore entry type. Valid values include: `Certificate`, `TrustedCertificate`, or `SecretKey`.	Yes
`"keystoreName"`	Name of the KSS keystore.	Yes
`"stripeName"`	Name of the stripe.	Yes
`"keystorePassword"`	Password for the KSS keystore.	No
`"keyPasswordList"`	List that contains key password for the trust certificate.	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the import operation, including:

Attribute	Description
“ERROR_CODE”	If “STATUS” is set to "Failed", provides the error code.
“ERROR_MSG”	If “STATUS” is set to “Failed”, provides the contents of the error message.
“STATUS”	Status of operation. For example, “SUCCEEDED” or “FAILED”.
“SUBJECT_DN”	Subject DN list that was imported.

cURL Example

The following example shows how to create a KSS keystore by submitting a POST request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @importcertkss.json -H Content-
Type:application/json https://myhost:7001/opss/v2/keystoreservice/certificates
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "keyAliasList" : ["myAlias"],
    "keystoreEntry":
"MIIC7DCCAqqgAwIBAgIEalhBSjALBgcqhkjOOAQDBQAwSDEKMAgGA1UEBhMBeTEKMAgGA1UECBMB
\neTEKMAgGA1UEBxMBeTEKMAgGA1UEChMBeTEKMAgGA1UECxMBeTEKMAgGA1UEAxMBeTAeFw0xNDA3\nMDMxM
TAwMTZaFw0xNDEwMDExMTAwMTZaMEgxCjAIBgNVBAYTAXkxCjAIBgNVBAgTAXkxCjAIBgNV
\nBAcTAXkxCjAIBgNVBAoTAXkxCjAIBgNVBAsTAXkxCjAIBgNVBAMTAXkwggG3MIIBLAYHKoZIzjgE
\nATCCAR8CgYEA/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow
\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVU
\nE1oWkTL2dfOuK2HXKu/
yIgMZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu/
o66oL5V0wLPQeCZ1FZV4661FlP5nEHEIGAtEkWcSPoTCgWE7fPCTKMyKbh
\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qFGQiaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7/
s9JKgOBhAACgYBrvzkjozmv6t6T0GNJES1R3ypRsBs8VLX2g3GotHd7Kht/TCj4HikelZDd
\nuL0t96R5Q4A3srOgSIZ
+0INRs1ER8y1Q37LyJNfyqYn5KqLBlN9bhSYAfcuIpjwIXGVfLQGdByD7\ntr4PSvZQx18K6p68HUCh
+jXQT9+7n3ZUIBzH5aMhMB8wHQYDVR0OBBYEFPdMpcEBbYSCYMdJiE4r
\ncQxf7Me4MAsGByqGSM44BAMFAAMvADAsAhQH/G1ixrEaWAG3lGWafkHgXxnzhwIUW5eSctgmaQBj
\nvKaY0E6fYJzcp5c=",
    "keystoreType" : "TrustedCertificate",
    "keystoreName" : "myKeystore",
    "stripeName" : "myStripe",
    "keystorePassword" : "myPwd"
    "keyPasswordList" : ["mykeyPwd"]
}
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "STATUS": "SUCCEEDED"
    "SUBJECT_DN": "CN=y,OU=y,O=y,L=y,ST=y,C=y"
}
```

GET Stripe KSS Keystores Method

Use the GET method to return all Keystore Service (KSS) keystores for a stripe.

REST Request

```
GET /opss/v2/keystoreservice/{stripeName}
```

Parameters

The following table summarizes the GET request parameters.

Name	Description	Type	Required
`"stripeName"`	Name of stripe for which you want to view all KSS keystores.	Path	Yes

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains information about the certificate, including:

Attribute	Description
`"keystores"`	List of keystores in the stripe.

cURL Example

The following example shows how to view all certificates for an alias by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password https://myhost:7001/opss/v2/keystoreservice/
myStripe
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "keystores":["trust","castore"]
}
```

GET Alias KSS Keystore Method

Use the GET method to view the alias for the Keystore Service (KSS) keystore.

REST Request

```
GET /opss/v2/keystoreservice/alias/{stripeName}/{keystoreName}/{entryType}
```

Parameters

The following table summarizes the GET request parameters.

Name	Description	Type	Required
“stripeName”	Name of the stripe.	Path	Yes
“keystoreName”	Name of the keystore.	Path	Yes
“entryType”	Keystore type. Valid values include `Certificate`, `TrustedCertificate`, or `SecretKey`. Wildcard "*" means all the types.	Path	Yes
"keystorePassword"	Base64 encoded keystore password	Header	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains information about the certificate, including:

Attribute	Description
“Alias”	List of keystore aliases in the stripe.

cURL Example

The following example shows how to view all certificates for an alias by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password https://myhost:7001/opss/v2/keystoreservice/
alias/myStripe/myKeystore/TrustedCertificate
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
   "Alias":["myAlias"]
}
```

GET Trusted Certificate KSS Keystore Method

Use the GET method to view trusted certificates in the Keystore Service (KSS) keystore. If the keystore is password-protected, you must provide a Base64-encoded header value for the keystore password.

REST Request

```
GET /opss/v2/keystoreservice/certificates
```

Parameters

The following table summarizes the GET request parameters.

Name	Description	Type	Required
“stripeName“	Name of the stripe.	Query	Yes
“keystoreName“	Name of the keystore.	Query	Yes
“keyAlias“	Alias for trusted certificate.	Query	Yes
“keystoreEntryType“	Type of keystore entry. Valid values include `Certificate`, `TrustedCertificate`, or `CertificateChain`.	Query	Yes
“keystorePassword“	Password for the KSS keystore.	Header	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains information about the certificate, including:

Attribute	Description
“CONTENT“	Contents of the Base64-encoded certificate.
“Extensions“	Optional extensions that are used to issue a certificate for a specific purpose. Each extension includes the following: • Object identifier (oid) that uniquely identifies it • Flag indicating whether the extension is critical • Set of values
“ISSUER_DN“	List of trusted distinguished names.
“NOT_AFTER“	Date the certificate expires.
“NOT_BEFORE“	Date the certificate is activated.
“SERIAL_NO“	Serial number of the JKS keystore.
“SIGNATURE“	Base64-encoded signature key.

Attribute	Description
“SIGNING_ALGORITHM“	Signing algorithm for the alias.
“SUBJECT_DN“	Subject distinguished names list.
“PUBLIC KEY“	String of public key value.

cURL Example

The following example shows how to view all certificates for an alias by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password -H keystorePassword:cHdkMQ== -H https://myhost:
7001/opss/v2/keystoreservice/
certificates?"stripeName=myStripe&keystoreName=myKeystore&keyAlias=client&keystoreEnt
ryType=Certificate"
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "SUBJECT_DN":"CN=y,OU=y,O=y,L=y,ST=y,C=y",
    "ISSUER_DN":"CN=y,OU=y,O=y,L=y,ST=y,C=y",
    "NOT_BEFORE":"Fri Jul 25 02:45:11 PDT 2014",
    "NOT_AFTER":"Thu Oct 23 02:45:11 PDT 2014",
    "SERIAL_NO":"982191050",
    "SIGNING_ALGORITHM":"1.2.840.10040.4.3",
    "PUBLIC_KEY": "MIIBtzCCASwGByqGSM44BAEwggEfAoGBAP1/
U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/
JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E
+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAhUAl2BQjxUjC8yykrmCouuEC/
BYHPUCgYEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT
+ZxBxCBgLRJFnEj6EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx
+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKLZl6Ae1UlZAFMO/7PSSoDgYQAAoGAa785I6M5r
+rek9BjSREtUd8qUbAbPFS19oNxqLR3eyobf0wo+B4pHpWQ3bi9LfekeUOAN7KzoEiGftCDUbNREfMtUN
+y8iTX8qmJ+SqiwZTfW4UmAH3LiKY8CFxlXy0BnQcg+7a+D0r2UMdfCuqevB1Aofo10E/fu592VCAcx
+U=""CONTENT":"-----BEGIN CERTIFICATE-----
\nMIIC7DCCAqqgAwIBAgIEOosLyjALBgcqhkjOOAQDBQAwS
EKMAgGA1UEBhMBcjEKMAgGA1UECBMB\ncjEKMAgGA1UEBxMBcjEKMAgGA1UEChMBcjEKMAgGA1UECxM
cjEKMAgGA1UEAxMBUjAeFw0xNDA3\nMjUwOTQ1MTFaFw0xNDEwMjMwOTQ1MTFaMEgxCjAIBgNVBAYTA
IxCjAIBgNVBAgTAXIxCjAIBgNV\nBAcTAXIxCjAIBgNVBAoTAXIxCjAIBgNVBAsTAXIxCjAIBgNVBAM
AVIwggG3MIIBLAYHKoZIzjgE\nATCCAR8CgYEA\/X9TgR11EilS30qcLuzk5\/YRt1I870QAwx4\/gL
RJmlFXUAiUftZPY1Y+r\/F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5\/oBHsQIsJPu6nX\/rfGG
/g7V+fGqKYVDwT7g\/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu\/yIgMZndFIAccCFQCXYFCPFSMLzLKS
YKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu\/o66oL5V0wLPQeCZ1FZV4661FlP5nEHE
GAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qFG
iaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7\/s9JKgOBhAACgYAjhpZybXj6rlXDow8srnSFE9dZJJpCKaQV
ACagQogePV+xlqPClDOoiQJ\nuvuUGHerDrThC1\/Wq5Uj1+TnkSKTy0qYxmQoq56xALa47np9TKtqt
4Vy8eUUorakG4lrjNt\/EgR\nfO675n+qINkKXKpcxaCicupRCYPkPXlnT4mtyKMhMB8wHQYDVR0OBB
EFDKbmPa2Il6SylJRPTv8\nQ+4CqpEhMAsGByqGSM44BAMFAAMvADAsAhQbkmlaUG5QDR5mXUiYC74p
\/FBOwIUGx5lc5Y01ppo\nvK3UgL7M8E3eOfc=\n-----END CERTIFICATE-----",
    "SIGNATURE":FEZN2l4SPFEK5jt2QZRb5Q==",
```

```
    "Extensions":"{subjectKeyIDExtension {oid = 2.5.29.14 critical = false, value =
329b98f6b6225e92ca52513d3bfc43ee02aa9121}}"
}
```

DELETE Trusted Certificate KSS Keystore Method

Use the Delete method to delete a certificate from a Keystore Service (KSS) keystore. If the keystore is password-protected, you must provide Base64-encoded header values for the keystore and key passwords.

REST Request

```
DELETE /opss/v2/keystoreservice/certificates
```

Parameters

The following table summarizes the DELETE request parameters.

Name	Description	Type	Required
“stripeName“	Name of stripe.	Query	Yes
“keystoreName“	Name of the keystore.	Query	Yes
“keyAlias“	Alias for the certificate in the KSS keystore.	Query	Yes
“keystorePassword“	Base64 encoded keystore password.	Header	No
“keyPassword“	Base64 encoded key password.	Header	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the import operation, including:

Attribute	Description
“ERROR_CODE”	If “STATUS” is set to “Failed”, provides the error code.
“ERROR_MSG”	If “STATUS” is set to “Failed”, provides the contents of the error message.
“STATUS”	Status of operation. For example, “SUCCEEDED”.

cURL Example

The following example shows how to delete a trusted certificate from the keystore by submitting a DELETE request on the REST resource using cURL.

```
curl -i -X DELETE -u username:password -H keystorePassword:cHdkMQ== -H
keyPassword:bXlQd2Qy  https://myhost:7001/opss/v2/keystoreservice/
certificates?"stripeName=myStripe&keystoreName=myKeystore&keyAlias=myAlias"
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "STATUS": "SUCCEEDED"
}
```

POST Secret Key KSS Keystore

Use the POST method to create a secret key used in symmetric encryption/decryption for a KSS keystore.

REST Request

```
POST /opss/v2/keystoreservice/secretkey
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the details of the create request:

Attribute	Description	Required
“stripeName“	Name of the stripe.	Yes
“keystoreName“	Name for the KSS keystore.	Yes
“keyAliasList“	List that contains alias for the secret key.	Yes
“keySize“	Size measured in bits of the of the key used in cryptographic algorithm.	Yes
“keyAlgorithm“	Controls the cryptographic characteristics of the algorithms that are used when securing messages.	Yes
“keystorePassword“	Password for the KSS keystore.	No
“keyPasswordList“	List that contains the password for the key.	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the import operation, including:

Attribute	Description
“ERROR_CODE“	If “STATUS“ is set to “Failed“, provides the error code.
“ERROR_MSG“	If “STATUS“ is set to “Failed“, provides the contents of the error message.

Attribute	Description
"STATUS"	Status of operation. For example, "SUCCEEDED" or "FAILED".

cURL Example

The following example shows how to create a secret key by submitting a POST request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @secretkey.json -H Content-
Type:application/json https://myhost:7001/opss/v2/keystoreservice/secretkey
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "stripeName" : "myStripe",
    "keystoreName" : "myKeystore",
    "keyAliasList" : ["myKeyAlias"],
    "keySize" : "56",
    "keyAlgorithm" : "DES",
    "keystorePassword" : "myPwd",
    "keyPasswordList" : ["myKeyPwd"]
}
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "STATUS": "SUCCEEDED"
}
```

GET Secret Key Properties KSS Keystore Method

Use the GET method to view the secret key properties for a KSS keystore. If the keystore is password-protected, you must provide Base64-encoded header values for the keystore and key passwords.

REST Request

```
GET /opss/v2/keystoreservice/secretkey
```

Parameters

The following table summarizes the GET request parameters.

Name	Description	Type	Required
stripeName	Name of the stripe.	Query	Yes

Name	Description	Type	Required
`keystoreName`	Name of the keystore.	Query	Yes
`keyAlias`	Alias of the secret key.	Query	Yes
`"returnKeyInResponse"`	Whether the key should be returned in the output.	Query	No
`"keystorePassword"`	Base64 encoded keystore password.	Header	No
`"keyPassword"`	Base64 encoded key password.	Header	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body contains information about the certificate, including:

Attribute	Description
`"keystore properties"`	List of secret key properties.
`"secret key"`	String of secret key data if "returnKeyInResponse" set to true.

cURL Example

The following example shows how to view all certificates for an alias by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password -H keystorePassword:bXlQd2Q= -H
keyPassword:bXlLZXlQd2Q= https://myhost:7001/opss/v2/keystoreservice/
secretkey?"stripeName=myStripe&keystoreName=myKeystore&keyAlias=myKeyAlias"
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
   "keystore properties":["DES"]
}
```

The following example shows how to view the properties of the secret key for an alias, including the secret key value.

```
curl -i -X GET -u username:password -H keystorePassword:bXlQd2Q= -H
keyPassword:bXlLZXlQd2Q= https://myhost:7001/opss/v2/keystoreservice/secretkey?
stripeName=myStripe&keystoreName=myKeystore&keyAlias=myKeyAlias&returnKeyInResponse=t
rue
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
"keystore properties":["DES"],
"secret key": "f65uMWvxAdM="
}
```

DELETE Secret Key KSS Keystore Method

Use the DELETE method to delete a secret key.

REST Request

```
DELETE /opss/v2/keystoreservice/secretkey
```

Parameters

The following table summarizes the DELETE request parameters.

Name	Description	Type	Required
`"stripeName"`	Name of the stripe.	Query	Yes
`"keystoreName"`	Name of the keystore.	Query	Yes
`"keyAlias"`	Alias of the secret key.	Query	Yes
`"keystorePassword"`	Password for the KSS keystore.	Header	No
`"keyPassword"`	Password for the key.	Header	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the delete operation, including:

Attribute	Description
`"ERROR_CODE"`	If `"STATUS"` is set to `"Failed"`, provides the error code.
`"ERROR_MSG"`	If `"STATUS"` is set to `"Failed"`, provides the contents of the error message.
`"STATUS"`	Status of operation. For example, `"SUCCEEDED"` or `"FAILED"`.

cURL Example

The following example shows how to delete a secret key from the keystore by submitting a DELETE request on the REST resource using cURL.

```
curl -i -X DELETE -u username:password -H keystorePassword:bXlQd2Q=  -H
keyPassword:bXlLZXlQd2Q=  https://myhost:7001/opss/v2/keystoreservice/
secretkey?"stripeName=myStripe&keystoreName=myKeystore"
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Header

The following example shows the response body in JSON format.

```
{
"STATUS": "SUCCEEDED"
}
```

POST Key Pair KSS Keystore

Use the POST method to create a key pair used in symmetric encryption/decryption for a KSS keystore.

REST Request

```
POST /opss/v2/keystoreservice/keypair
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the details of the create request:

Attribute	Description	Required
`"stripeName"`	Name of the stripe.	Yes
`"keystoreName"`	Name for the KSS keystore.	Yes
`"keyAliasList"`	List that contains alias for the secret key.	Yes
`"keySize"`	Size, measured in bits, of the key used in the cryptographic algorithm.	Yes
`"keyAlgorithm"`	Controls the cryptographic characteristics of the algorithms that are used when securing messages.	Yes
`"DN"`	Distinguished name for the key	Yes
`"keystorePassword"`	Password for the KSS keystore.	No
`"keyPassword"`	Password for the key.	No
`"keyPasswordList"`	List that contains password for the list.	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the import operation, including:

Attribute	Description
"ERROR_CODE"	If "STATUS" is set to "Failed", provides the error code.
"ERROR_MSG"	If "STATUS" is set to "Failed", provides the contents of the error message.
"STATUS"	Status of operation. For example, "SUCCEEDED" or "FAILED".

cURL Example

The following example shows how to create a key pair by submitting a POST request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @keypair.json -H Content-
Type:application/json https://myhost:7001/opss/v2/keystoreservice/keypair
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "stripeName" : "myStripe",
    "keystoreName" : "myKeystore",
    "keyAliasList" : ["myKeyAlias"],
    "keySize" : "256",
    "algorithm" : "EC",
    "DN" :
"CN=CertGenCA,OU=FORTESTINGONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US",
    "keystorePassword" : "myPwd",
    "keyPasswordList" : ["myKeyPwd"]
}
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "STATUS": "SUCCEEDED"
}
```

GET Key Pair KSS Keystore Method

Use the GET method to view to view a key pair for a KSS keystore. If the keystore is password-protected, you must provide Base64-encoded header values for the keystore and key passwords.

REST Request

```
GET /opss/v2/keystoreservice/keypair
```

Parameters

The following table summarizes the GET request parameters.

Name	Description	Type	Required
`"stripeName"`	Name of the stripe.	Query	Yes
`"keystoreName"`	Name of the keystore.	Query	Yes
`"keyAlias"`	Alias of the secret key.	Query	Yes
`"keystorePassword"`	Password for the KSS keystore.	Header	No
`"keyPassword"`	Password for the key.	Header	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the get operation, including:

Attribute	Description
`"private key"`	Base64 encoded private key.

cURL Example

The following example shows how to view a key pair by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password -H keystorePassword:bXlQd2Q= -H
keyPassword:bXlLZXlQd2Q= https://myhost:7001/opss/v2/keystoreservice/keypair?
stripeName=myStripe&keystoreName=myKeystore&keyAlias=myKeyAlias
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "private key":
"MEECAQAwEwYHKoZIzj0CAQYIKoZIzj0DAQcEJzAlAgEBBCBzQbYz6xUZjr/
XuwVMJj1XXQCquis0f9q5SD9NXhlBjw=="
}
```

DELETE Key Pair KSS Keystore Method

Use the DELETE method to delete a key pair.

REST Request

```
DELETE /opss/v2/keystoreservice/keypair
```

Parameters

The following table summarizes the DELETE request parameters.

Name	Description	Type	Required
"stripeName"	Name of the stripe.	Query	Yes
"keystoreName"	Name of the keystore.	Query	Yes
"keyAlias"	Alias of key pair.	Query	Yes
"keystorePassword"	Base64 encoded keystore password.	Header	No
"keyPassword"	Base64 encoded key password.	Header	No

Response Body

Media Types for the request or response body.

Media Types: `application/json`

The response body returns the status of the delete operation, including:

Attribute	Description
"ERROR_CODE"	If "STATUS" is set to "Failed", provides the error code.
"ERROR_MSG"	If "STATUS" is set to "Failed", provides the contents of the error message.
"STATUS"	Status of operation. For example, "SUCCEEDED" or "FAILED".

cURL Example

The following example shows how to delete a key pair from the keystore by submitting a DELETE request on the REST resource using cURL.

```
curl -i -X DELETE -u username:password -H keystorePassword:bXlQd2Q= https://myhost:
7001/opss/v2/keystoreservice/
keypair?"stripeName=myStripe&keystoreName=myKeystore&keyAlias=myKeyAlias"
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "STATUS": "SUCCEEDED""
}
```

DELETE Keystore Service KSS Keystore Method

Use the DELETE method to delete a Keystore Service (KSS) keystore. If the keystore is password-protected, you must provide Base64-encoded header values for the keystore password.

REST Request

```
DELETE /opss/v2/keystoreservice
```

Parameters

The following table summarizes the DELETE request parameters.

Name	Description	Type	Required
"stripeName"	Name of the stripe.	Query	Yes
"keystoreName"	Name of the keystore.	Query	Yes
"keyStorePassword"	Password for the key store.	Header	No

Response Body

Media types for the request or response body.

Media Types: `application/json`

The response body returns the status of the delete operation, including:

Attribute	Description
"ERROR_CODE"	If "STATUS" is set to "Failed", provides the error code.
"ERROR_MSG"	If "STATUS" is set to "Failed", provides the contents of the error message.
"STATUS"	Status of operation. For example, "SUCCEEDED" or "FAILED".

cURL Example

The following example shows how to delete a trusted certificate from the keystore by submitting a DELETE request on the REST resource using cURL.

```
curl -i -X DELETE -u username:password -H keystorePassword:bXlQd2Q= https://myhost:
7001/opss/v2/keystoreservice?"stripeName=myStripe&keystoreName=myKeystore"
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 204 No Content
```

5

Creating and Validating Trust Tokens

Oracle Platform Security Services (OPSS) uses the Trust service to manage trust tokens. You can get and validate tokens using REST. Only REST clients that have permission to issue and validate tokens for users in a particular Identity Domain (IDD) are allowed to issue and validate tokens. A client must declare an IDD during registration so that privileges to the client can be granted.

For details on registration, see POST Registration Method.

Section	Method	Resource Path
POST Trust Service Issue Token Method	`POST`	`/opss/v2/trustService`
POST Trust Service Validate Token Method	`POST`	`/opss/v2/trustService`

POST Trust Service Issue Token Method

Use the POST method to get a trust token.

REST Request

```
POST opss/v2/trustService/issue
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the details of the create request:

Table 5-1 Trust Attributes

Attribute	Description	Required
`"protocol"`	Trust protocol. Only the embedded protocol is supported.	No
`"tokenType"`	Type of token. Supported token types are: `SAML`, `SAML2`, and `JWT`.	Yes
`"username"`	User name for which the token is issued.	Yes
`"tokenSigningMethod"`	Cryptographic algorithms to sign the contents of the JWT token. This attribute is only used with the JWT-Token type. Only PKI signing methods are supported: RS-256 (RSA using SHA-256 hash algorithm), RS-384(RSA using SHA-384 hash algorithm), and RS-512(RSA using SHA-512 hash algorithm). (JWT-Token only)	Yes

Table 5-1 (Cont.) Trust Attributes

Attribute	Description	Required
`"confirmationMethod"`	Method that a relying party uses to verify the correspondence of the subject of the assertion with the party presenting the assertion. Supported confirmation methods are `sender-vouches`, `holder-of-key`, and `bearer`. (SAML2 only)	Yes
`"scdAddress"`	Subject confirmation data address. The network address/location from which an attesting entity can present the assertion. (SAML2 only)	Yes
`"addAuthenticatingAuthorities"`	List of identity providers trusted by the requester to authenticate the presenter. (SAML2 only)	Yes
`"nameIdFormat"`	Name identifier formats supported by the identity provider. Name identifiers are a way for providers to communicate with each other regarding a user. • urn:oasis:names:tc:SAML:2.0:nameid-format:persistent • urn:oasis:names:tc:SAML:2.0:nameid-format:transient • urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress • urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified • urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName • urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName • urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos • urn:oasis:names:tc:SAML:2.0:nameid-format:entity (SAML and SAML2 only)	No
`"idd"`	Identity domain.	Yes
`"expirationDate"`	Date the token expires and can no longer be accepted for processing. Must be in the format: `yyyy-MM-dd' T'HH:mm:ss.SSSZ`.	Yes
`"appliesTo"`	Scope (endpoint target) to which the token applies.	No
`"additionalClaims"`	JWT claims to add to the claim segment. This attribute is only used with the JWT-Token type.	No

cURL Example

The following example shows how to get a trust token by submitting a POST request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @issuetoken.json -H Content-
Type:application/json https://myhost:7001/opss/v2/trustService/issue
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "tokenType" : "JWT",
    "username" : "john.doe",
    "tokenSigningMethod" : "RS-256",
    "idd" : "cisco",
    "expirationDate" : "2015-10-19T12:08:56.235-0700",
}
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 201 Created
```

POST Trust Service Validate Token Method

Use the POST method to validate a trust token.

REST Request

```
POST opss/v2/trustService/validate
```

Request Body

Media types for the request or response body.

Media Types: `application/json`

The request body contains the details of the create request:

Table 5-2 Trust Attributes

Attribute	Description	Required
`"token"`	Identity token.	Yes
`"protocol"`	Trust protocol. Only the `ws-trust` protocol is supported.	No
`"tokenType"`	Type of token. Supported token types are: `SAML`, `SAML2`, and `JWT`.	Yes
`"username"`	User name for which the token is issued.	Yes

Table 5-2 (Cont.) Trust Attributes

Attribute	Description	Required
"tokenSigningMethod"	Cryptographic algorithms to sign the contents of the JWT token. This attribute is only used with the JWT-Token type. Only PKI signing methods are supported: RS-256 (RSA using SHA-256 hash algorithm), RS-384(RSA using SHA-384 hash algorithm), and RS-512(RSA using SHA-512 hash algorithm). (JWT-Token only)	Yes
"confirmationMethod"	SAML method that is used to provide proof for a subject and a SAML assertion. Supported confirmation methods are sender-vouches, holder-of-key, and bearer. (SAML2 only)	Yes

Response Body

Media types for the request or response body.

Media Types: application/json

The response body contains details about the validate operation, including:

Attribute	Description
"username"	User name for which the token is issued.
"idd"	Identity domain.
"expirationDate"	Date the token expires and can no longer be accepted for processing.
"appliesTo"	Scope (endpoint target) to which the token applies.
"additionalClaims"	JWT claims to add to the claim segment. This attribute is only used with the JWT-Token type.

cURL Example

The following example shows how to import a KSS keystore by submitting a POST request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @validatetoken.json -H Content-
Type:application/json https://myhost:7001/opss/v2/trustService/validate
```

Example of Request Body

The following example shows the request body in JSON format.

```
{
    "token" : "eyThbGciOiJRUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzY290F2
guaW8iLCJleHAiOjEzMDA4MTszODAsIm5hbWUiOiJDfHJpcyBTWXZpbGxlamEiDCJhZG1pbi
I6dHJ1ZR0.03f329983b83f7d9a9f5fef85305880101d5e402afafa20154d094s229f7578",
    "protocol" : "ws-trust",
```

```
    "tokenType" : "JWT",
    "username" : "john.doe",
    "tokenSigningMethod" : "RS-256",
    "confirmationMethod" : "bearer"
}
```

Example of Response Header

The following example shows the response header. For more about the HTTP status codes, see HTTP Status Codes for HTTP Methods.

```
HTTP/1.1 200 OK
```

Example of Response Body

The following example shows the response body in JSON format.

```
{
    "username" : "john.doe",
    "idd" : "cisco",
    "expirationDate" : "2015-10-19T12:08:56.235-0700",
}
```

6
Authorizing Access

Oracle Platform Security Services (OPSS) uses the XACML3.0 REST profile based authorization service to manage authorization. You can manage authorization using REST.

Section	Method	Resource Path
GET PDP Link Method	GET	/opss/v2/authz/xacml/
POST Policy Decision Method	POST	/opss/v2/authz/xacml/

GET PDP Link Method

Use the GET method to get the Policy Decision Point (PDP) for an application.

REST Request

```
GET /opss/v2/authz/xacml/appName
```

Response Body

Media types for the request or response body.

Media Types: `application/json` or `application/xml`

The response body contains details about the PDP link, including:

Attribute	Description
"rel"	PDP definition provider
"href"	PDP link.

cURL Example

The following example shows how to get the PDP link for an application by submitting a GET request on the REST resource using cURL. Examples for both JSON and XML are provided.

JSON Example

```
curl -i -X GET -u username:password -H Content-Type:application/json https://myhost:
7001/opss/v2/authz/xacml/MyApp
```

Example of Response Body with JSON

The following example shows the response body when using JSON.

```
{    "resources": {        "resource": {            "link": {                "rel":
"https://docs.oasis-open.org/ns/xacml/relation/pdp",                "href": "/
opss/v2/xacml/MyApp/pdp"            }        }    }}
```

XML Example

```
curl -i -X GET -u username:password -H Content-Type:application/xml https://myhost:
7001/opss/v2/authz/xacml/MyApp
```

Example of Response Body with XML

The following example shows the response body when using XML.

```
<resources xmlns=http://ietf.org/ns/home-documents
   mlns:atom="http://www.w3.org/2005/Atom">
   <resource rel="http://docs.oasis-open.org/ns/xacml/relation/pdp">
      <atom:link href="/opss/v2/xacml/MyApp/pdp"/>
   </resource>
</resources>
```

POST Policy Decision Method

Use the POST method to send a policy decision authorization request to the PDP system.

REST Request

`POST /opss/v2/authz/xacml/`*`appName`*`/pdp/`

The URI can also specify the resource type. If the name of resource type is decided by application name, then it can be omitted. The resource type is optional, and it is specified by query parameter if needed.

`POST /opss/v2/authz/xacml/`*`appName`*`/pdp/?resType=`*`resType`*

Request Body

Media types for the request or response body.

Media Types: `application/xacml+json;version=3.0` or `application//xacml +xml;version=3.0`

Response Body

Media types for the request or response body.

Media Types: `application/xacml+json;version=3.0` or `application//xacml +xml;version=3.0`

cURL Example

The following example shows how to request a policy decision for an application by submitting a POST request on the REST resource using cURL. Examples for both JSON and XML are provided.

JSON Example

```
curl -i -X GET -u username:password --data @policyRequest.json -H Content-
Type:application/xacml+json;version=3.0 https://myhost:7001/opss/v2/authz/xacml/
MyApp/pdp
```

Example of Request with JSON

The following example shows the request body when using JSON.

```
{
    "Request": {
        …
    }
 }
```

Example of Response Body with JSON

The following example shows the response body when using JSON.

```
{
    "Response": [
        …
    ]
 }
```

XML Example

```
curl -i -X GET -u username:password --data @policyRequest.xml -H Content-
Type:application/xacml+xml;version=3.0 https://myhost:7001/opss/v2/authz/xacml/MyApp/
pdp
```

Example of Request with XML

The following example shows the request body when using XML.

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"> … </Request>
```

Example of Response with XML

The following example shows the response body when using XML.

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"> … </Request>
```