Oracle® Fusion Middleware
REST API for Oracle Platform Security Services
12c (12.2.1)
E68152-01

December 2015
Documentation that describes how to use the Oracle Platform Security Services REST API for credential store, keystore, and trust store management.
Contents

Preface .......................................................................................................................................................... v
  Audience ..................................................................................................................................................... v
  Documentation Accessibility ........................................................................................................................ v
  Related Documents ....................................................................................................................................... v
  Conventions ................................................................................................................................................. v

What's New In This Guide ........................................................................................................................... vii
  New and Changed Features for 12c (12.2.1)............................................................................................. vii

1 About the OPSS REST API
  1.1 Introducing the OPSS REST API ........................................................................................................... 1-1
  1.2 General URL Structure for OPSS Resources ........................................................................................ 1-1
  1.3 Authenticating REST Resources ............................................................................................................. 1-2
  1.4 Using HTTP Methods with OPSS REST ............................................................................................... 1-2
  1.5 HTTP Status Codes for HTTP Methods ............................................................................................... 1-2

2 Registering OPSS Clients
  POST Registration Method ........................................................................................................................... 2-2
  GET Registration Method ........................................................................................................................... 2-4
  PUT Registration Method ........................................................................................................................... 2-5
  DELETE Registration Method ..................................................................................................................... 2-6

3 Managing Credentials in the Credential Store
  POST Credentials Method .......................................................................................................................... 3-2
  GET Credentials Using Map and Key Method ............................................................................................ 3-3
  GET Credentials Using Map Method ........................................................................................................... 3-4
  GET Credentials Using Resource ID ............................................................................................................ 3-6
  PUT Credential Method .............................................................................................................................. 3-7

4 Managing Keystores
  POST New KSS Keystore Method ............................................................................................................... 4-2
  POST Import KSS Keystore Method ............................................................................................................. 4-4
PUT Password Update KSS Keystore Method ................................................................. 4-6
POST Trusted Certificate KSS Keystore Method ............................................................ 4-8
GET Stripe KSS Keystores Method .............................................................................. 4-10
GET Alias KSS Keystore Method ............................................................................... 4-11
GET Trusted Certificate KSS Keystore Method .......................................................... 4-12
DELETE Trusted Certificate KSS Keystore Method ...................................................... 4-14
POST Secret Key KSS Keystore ................................................................................... 4-16
GET Secret Key Properties KSS Keystore Method ...................................................... 4-18
DELETE Secret Key KSS Keystore Method .................................................................. 4-20
POST Key Pair KSS Keystore .................................................................................... 4-21
GET Key Pair KSS Keystore Method ......................................................................... 4-23
DELETE Key Pair KSS Keystore Method .................................................................... 4-24
DELETE Keystore Service KSS Keystore Method ........................................................ 4-25

5 Creating and Validating Trust Tokens

POST Trust Service Issue Token Method ................................................................. 5-2
POST Trust Service Validate Token Method ............................................................. 5-5

6 Authorizing Access

GET PDP Link Method .............................................................................................. 6-2
POST Policy Decision Method .................................................................................. 6-4
This preface describes the document accessibility features and conventions used in this

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

Access to Oracle Support

Oracle customers that 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
- Infrastructure Security WLST Command Reference
- 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:
<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
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 12c (12.2.1).

New and Changed Features for 12c (12.2.1)

Oracle Platform Security Services 12c (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. See "Registering OPSS Clients" on page 2-1.

- 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" on page 3-1.

- 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" on page 4-1.

- Trust service RESTful API, which provides REST clients with the ability to manage trust tokens. See "Creating and Validating Trust Tokens" on page 5-1.

- Authorization service RESTful API, which provides REST clients with the ability to manage XACML3.0 REST profile authorization. See "Authorizing Access" on page 6-1.
About the OPSS REST API

This section introduces the Oracle Fusion Middleware representational state transfer (REST) API for managing Oracle Platform Security Services (OPSS).

This chapter includes the following sections:

- Introducing the OPSS REST API
- General URL Structure for OPSS Resources
- Authenticating REST Resources
- Using HTTP Methods with OPSS REST
- HTTP Status Codes for HTTP Methods

1.1 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 keystores. 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.

1.2 General URL Structure for OPSS Resources

Use the following URL to manage security:

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

Where:
1.3 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 example, to authenticate using cURL, pass the user name and password using the -u cURL option.


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"

1.4 Using HTTP Methods with OPSS REST

The OPSS REST endpoints support standard HTTP semantics.

<table>
<thead>
<tr>
<th>REST Method</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieve information about the REST resource.</td>
</tr>
<tr>
<td>POST</td>
<td>Add a REST resource.</td>
</tr>
<tr>
<td>PUT</td>
<td>Update a REST resource.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete a REST resource.</td>
</tr>
</tbody>
</table>

1.5 HTTP Status Codes for HTTP Methods

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

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>The request was successfully completed. A 200 status is returned for successful GET or POST method.</td>
</tr>
<tr>
<td>201 Created</td>
<td>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.</td>
</tr>
<tr>
<td>HTTP Status Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>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 (<em>\text{detached}=\text{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 (</em>\text{detached}=\text{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.</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>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).</td>
</tr>
<tr>
<td>401 Unauthorized</td>
<td>The request is not authorized. The authentication credentials included with this request are missing or invalid.</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>The user cannot be authenticated. The user does not have authorization to perform this request.</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>The request includes a resource URI that does not exist.</td>
</tr>
<tr>
<td>405 Method Not Allowed</td>
<td>The HTTP verb specified in the request (DELETE, GET, POST, PUT) is not supported for this request URI.</td>
</tr>
<tr>
<td>406 Not Acceptable</td>
<td>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.</td>
</tr>
<tr>
<td>415 Not Acceptable</td>
<td>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).</td>
</tr>
<tr>
<td>500 Internal Server Error</td>
<td>The server encountered an unexpected condition that prevented it from fulfilling the request.</td>
</tr>
<tr>
<td>503 Service Unavailable</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Section</th>
<th>Method</th>
<th>Resource Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST Registration Method</td>
<td>POST</td>
<td>/opss/v2/</td>
</tr>
<tr>
<td>GET Registration Method</td>
<td>GET</td>
<td>/opss/v2/</td>
</tr>
<tr>
<td>PUT Registration Method</td>
<td>PUT</td>
<td>/opss/v2/</td>
</tr>
<tr>
<td>DELETE Registration Method</td>
<td>DELETE</td>
<td>/opss/v2/</td>
</tr>
</tbody>
</table>
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: | application/json |

The request body contains the details of the register request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;clientName&quot;</td>
<td>A unique name that identifies the client.</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;policystoreStripe&quot;</td>
<td>The policy store stripe to which the client is assigned</td>
<td>No</td>
</tr>
<tr>
<td>&quot;keystore&quot;</td>
<td>A list of keystores used for the client</td>
<td>No</td>
</tr>
<tr>
<td>&quot;credentialMap&quot;</td>
<td>A name of the credential map that is used to store credential keys.</td>
<td>No</td>
</tr>
<tr>
<td>&quot;auditComponent&quot;</td>
<td>A unique name to identify the audit rules for a client</td>
<td>No</td>
</tr>
<tr>
<td>&quot;trustIssueIDD&quot;</td>
<td>A list identity domains that can issue trust tokens</td>
<td>No</td>
</tr>
<tr>
<td>&quot;trustValidateIDD&quot;</td>
<td>A list identity domains that can validate trust tokens</td>
<td>No</td>
</tr>
<tr>
<td>&quot;adminGroup&quot;</td>
<td>A group with the operator role</td>
<td>No</td>
</tr>
<tr>
<td>&quot;operatorGroup&quot;</td>
<td>A group with the operator role</td>
<td>No</td>
</tr>
<tr>
<td>&quot;viewerGroup&quot;</td>
<td>A group with the viewer role</td>
<td>No</td>
</tr>
</tbody>
</table>

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/v1/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: | 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/v1/opssRestClient/myClientName

Example of Response Header

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

```json
{
    "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

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>

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/v1/opssRestClient/myClientName

Example of Request Body

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

```json
{
  "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 registered client by submitting a DELETE request on the REST resource using cURL.

```
curl -i -X DELETE -u username:password https://myhost:7001/opss/v1/opssRestClient/myClientName
```
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.

<table>
<thead>
<tr>
<th>Section</th>
<th>Method</th>
<th>Resource Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST Credentials Method</td>
<td>POST</td>
<td>/opss/v2/credentials</td>
</tr>
<tr>
<td>GET Credentials Using Map and Key Method</td>
<td>GET</td>
<td>/opss/v2/credentials</td>
</tr>
<tr>
<td>GET Credentials Using Map Method</td>
<td>GET</td>
<td>/opss/v2/credentials</td>
</tr>
<tr>
<td>GET Credentials Using Resource ID</td>
<td>GET</td>
<td>/opss/v2/credentials</td>
</tr>
<tr>
<td>PUT Credential Method</td>
<td>PUT</td>
<td>/opss/v2/credentials</td>
</tr>
</tbody>
</table>
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: application/json

The request body contains the details of the create request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;username&quot;</td>
<td>Username for the credentials</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;password&quot;</td>
<td>Password for the credentials</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;description&quot;</td>
<td>A description for the credentials</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;expiration&quot;</td>
<td>The expiration date for the credentials formatted as yyyy-MM-dd'T'HH:mm:ss.SSSZ.</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;type&quot;</td>
<td>The type of the credentials</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;namespace&quot;</td>
<td>a unique name for the credential namespace</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;name&quot;</td>
<td>A unique name that identifies the credential</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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 shows an example of the request body in JSON format.

```
{
    "userName": "myUser3",
    "password": "mypass123",
    "description": "mydescription",
    "expiration": "2023-07-04T12:08:56.235-0700",
    "type": "PasswordCredential"
    "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: | 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/admin/v1/credentials/?filter="map=mymap,key=mykey"
```

Example of Response Header

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

```
{
  "id": "1234567890",
  "userName": "myUser3",
  "password": "mypass123",
  "description": "mydescription",
  "expiration": "2000-07-04T12:08:56.235-0700",
  "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**

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>

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.

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

**Example of Response Header**

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

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

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>

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.


Example of Response Header

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

```json
{
    "id": "1234567890",
    "userName": "myUser3",
    "password": "mypass123",
    "description": "mydescription",
    "expiration": "2000-07-04T12:08:56.235-0700",
    "type": "PasswordCredential"
}
```
PUT Credential Method

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

REST Request

PUT /opss/v2/credentials

Request Body

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>

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

Response Body

The output of a PUT request is a Resource ID.

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 POST -u username:password --data @replacecred.json -H Content-Type:application/json https://myhost:7001/opss/v2/credentials

Example of Request Body

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

```json
{
  "id": "1234567890",
  "userName": "myUser3",
  "password": "mypass123",
  "description": "mydescription",
  "expiration": "2000-07-04T12:08:56.235-0700",
  "type": "PasswordCredential",
  "namespace": "MyMap",
  "name": "myKey"
}
```
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.

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

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>

The request body contains the details of the create request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripe&quot;</td>
<td>Name of the stripe to contain the KSS keystore.</td>
</tr>
<tr>
<td>&quot;keystore&quot;</td>
<td>Name for the KSS keystore.</td>
</tr>
<tr>
<td>&quot;pwd&quot;</td>
<td>Password for the KSS keystore.</td>
</tr>
<tr>
<td>&quot;permission&quot;</td>
<td>Boolean value that specifies whether to create a permission-based keystore.</td>
</tr>
</tbody>
</table>

Response Body

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ERROR_CODE&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>&quot;ERROR_MSG&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>

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 shows an example of the request body in JSON format.

```
{
    "stripe" : "myStripe",
    "keystore" : "myKeystore",
    "pwd" : "myPwd",
    "permission" : "false"
}```
Note: A password is required unless creating a permission-based keystore ("permission": "true").

Example of Response Header
The following shows an example of the response header. For more about the HTTP status codes, see "HTTP Status Codes for HTTP Methods."

HTTP/1.1 201 Created

Example of Response Body
The following shows an example of the response body in JSON format.

```json
{
  "STATUS": "Succeeded"
}
```
POST Import KSS Keystore Method

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

REST Request

POST /idaas/platform/admin/v1/keystoreservice/keystore

Request Body

Media Types: multipart/form-data

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</td>
</tr>
<tr>
<td>&quot;keystoreFile&quot;</td>
<td>Name of a valid local JKS keystore file</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name for the JKS keystore.</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the local keystore file that is being imported and the keystore entry, if password-protected.</td>
</tr>
<tr>
<td>&quot;keystoreType&quot;</td>
<td>Keystore type. This value must be set to JKS.</td>
</tr>
<tr>
<td>&quot;keyAliases&quot;</td>
<td>Comma-separated list of aliases for the keys to be imported from the keystoreFile.</td>
</tr>
<tr>
<td>&quot;keyPasswords&quot;</td>
<td>Comma-separated list of passwords for the keys to be imported from the keystoreFile.</td>
</tr>
<tr>
<td>&quot;permission&quot;</td>
<td>Boolean value that specifies whether to import as a permission-based keystore.</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;alias n&quot;</td>
<td>List of keystores in the stripe, where n serves as an index that starts at 1 and is incremented by 1 for each additional keystore.</td>
</tr>
<tr>
<td>&quot;ERROR_CODE&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>&quot;ERROR_MSG&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>
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 -H Content-Type:multipart/form-data --form "stripeName=myStripe" --form "keystoreFile=@clientkeystore" --form "keystoreName=myKeystore" --form "keystorePassword=myPwd" --form "keystoreType=JKS" --form "keyAliases=client" --form "keyPasswords=myPwd2" --form "permission=false" https://myhost:7001/opss/v2/keystoreservice/keystore

Example of Response Header

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

HTTP/1.1 201 Created

Example of Response Body

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

```json
{
  "STATUS":"Succeeded",
  "SUCCESS_MSG":"Aliases:client imported successfully",
  "alias 1":"client"
}
```
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

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripe&quot;</td>
<td>Name of the stripe.</td>
</tr>
<tr>
<td>&quot;keystore&quot;</td>
<td>Name of the KSS keystore.</td>
</tr>
<tr>
<td>&quot;newpass&quot;</td>
<td>New password for the keystore.</td>
</tr>
<tr>
<td>&quot;oldpass&quot;</td>
<td>Old password for the keystore.</td>
</tr>
</tbody>
</table>

Response Body

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ERROR_CODE&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>&quot;ERROR_MSG&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>

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 shows an example of the request body in JSON format.

```
{
    "stripe" : "myStripe",
    "keystore" : "mykssstore",
    "oldpass" : "myPwd",
    "newpass" : "myNewPwd"
}
```
Example of Response Header
The following shows an example of 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 shows an example of the response body in JSON format.

```json
{
  "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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;keyAlias&quot;</td>
<td>Alias for the trusted certificate.</td>
</tr>
<tr>
<td>&quot;keystoreEntry&quot;</td>
<td>Base64-encoded certificate.</td>
</tr>
<tr>
<td>&quot;keystoreEntryType&quot;</td>
<td>Keystore entry type. Valid values include: Certificate, TrustedCertificate, or SecretKey.</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name of the KSS keystore.</td>
</tr>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
</tr>
</tbody>
</table>

Response Body

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ERROR_CODE&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>&quot;ERROR_MSG&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
<tr>
<td>&quot;SUBJECT_DN&quot;</td>
<td>Subject DN list that was imported.</td>
</tr>
</tbody>
</table>

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 shows an example of the request body in JSON format.

```
{
  "keyAlias": "myAlias",
  "keystoreEntry": "MIIC7DCCAggAwIBAgIBALBgcgcghjOQADBQawSDEKAgGA1UEBhMeTEKMAgGA1UECBMB
  "keystorePassword": "myPwd"
}
```

**Example of Response Header**

The following shows an example of 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 shows an example of 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of stripe for which you want to view all KSS keystores.</td>
<td>Path</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

The response body contains information about the certificate, including:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;keystore n&quot;</td>
<td>List of keystores in the stripe, where n serves as an index that starts at 1 and is incremented by 1 for each additional keystore.</td>
</tr>
</tbody>
</table>

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 shows an example of 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 shows an example of the response body in JSON format.

```
{
  "keystore 1":"trust",
  "keystore 2":"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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</td>
<td>Path</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name of the keystore.</td>
<td>Path</td>
</tr>
<tr>
<td>&quot;entryType&quot;</td>
<td>Keystore type. Valid values include Certificate,</td>
<td>Path</td>
</tr>
<tr>
<td></td>
<td>TrustedCertificate, or SecretKey.</td>
<td></td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

The response body contains information about the certificate, including:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;keystore n&quot;</td>
<td>List of keystore aliases in the stripe where n serves as an index that starts at 1 and is incremented by 1 for each additional property.</td>
</tr>
</tbody>
</table>

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

Example of Response Header

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

```json
{
    "keystore 1":"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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>stripeName</em></td>
<td>Name of the stripe.</td>
<td>Query</td>
</tr>
<tr>
<td><em>keystoreName</em></td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td><em>keyAlias</em></td>
<td>Alias for trusted certificate.</td>
<td>Query</td>
</tr>
<tr>
<td><em>keystoreEntryType</em></td>
<td>Type of keystore entry. Valid values include Certificate, TrustedCertificate, or CertificateChain.</td>
<td>Query</td>
</tr>
<tr>
<td><em>keystorePassword</em></td>
<td>Password for the KSS keystore.</td>
<td>Header</td>
</tr>
<tr>
<td><em>keyPassword</em></td>
<td>Password for the key.</td>
<td>Header</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

The response body contains information about the certificate, including:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>CONTENT</em></td>
<td>Contents of the Base64-encoded certificate.</td>
</tr>
<tr>
<td><em>Extensions</em></td>
<td>Optional extensions that are used to issue a certificate for a specific purpose. Each extension includes the following:</td>
</tr>
<tr>
<td></td>
<td>■ Object identifier (oid) that uniquely identifies it</td>
</tr>
<tr>
<td></td>
<td>■ Flag indicating whether the extension is critical</td>
</tr>
<tr>
<td></td>
<td>■ Set of values</td>
</tr>
<tr>
<td><em>ISSUER_DN</em></td>
<td>List of trusted distinguished names.</td>
</tr>
<tr>
<td><em>NOT_AFTER</em></td>
<td>Date the certificate expires.</td>
</tr>
<tr>
<td><em>NOT_BEFORE</em></td>
<td>Date the certificate is activated.</td>
</tr>
<tr>
<td><em>SERIAL_NO</em></td>
<td>Serial number of the JKS keystore.</td>
</tr>
<tr>
<td><em>SIGNATURE</em></td>
<td>Base64-encoded signature key.</td>
</tr>
<tr>
<td><em>SIGNING_ALGORITHM</em></td>
<td>Signing algorithm for the alias.</td>
</tr>
<tr>
<td><em>SUBJECT_DN</em></td>
<td>Subject distinguished names list.</td>
</tr>
</tbody>
</table>
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 keyPassword:bXlQd2Qy https://myhost:7001/opss/v2/keystoreservice/certificates?"stripeName=myStripe&keystoreName=myKeystore&keyAlias=client&keystoreEntryType=Certificate"

Example of Response Header

The following shows an example of 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 shows an example of 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_BEFORER":Fri Jul 25 02:45:11 PDT 2014",
    "NOT_AFTERR":Thu Oct 23 02:45:11 PDT 2014",
    "SERIAL_NO":"982191050",
    "SIGNING_ALGORITHM":":1.2.840.10040.4.3",
    "CONTENT":":----BEGIN CERTIFICATE-----
    "SIGNATURE":FEZN2l4SPFEK5jt2QZRb5Q==",
    "Extensions":{}
}
```
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of stripe.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keyAlias&quot;</td>
<td>Alias for the certificate in the KSS keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
<td>Header</td>
</tr>
<tr>
<td>&quot;keyPassword&quot;</td>
<td>Password for the key.</td>
<td>Header</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ERROR_CODE&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>&quot;ERROR_MSG&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>

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 shows an example of 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 shows an example of 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

The request body contains the details of the create request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>stripeName</em></td>
<td>Name of the stripe.</td>
</tr>
<tr>
<td><em>keystoreName</em></td>
<td>Name for the KSS keystore.</td>
</tr>
<tr>
<td><em>keyAlias</em></td>
<td>Alias for the secret key.</td>
</tr>
<tr>
<td><em>keySize</em></td>
<td>Size measured in bits of the of the key used in cryptographic algorithm.</td>
</tr>
<tr>
<td><em>algorithm</em></td>
<td>Controls the cryptographic characteristics of the algorithms that are used when securing messages.</td>
</tr>
<tr>
<td><em>keystorePassword</em></td>
<td>Password for the KSS keystore.</td>
</tr>
<tr>
<td><em>keyPassword</em></td>
<td>Password for the key.</td>
</tr>
</tbody>
</table>

Response Body

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ERROR_CODE</em></td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td><em>ERROR_MSG</em></td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td><em>STATUS</em></td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>

cURL Example

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

```bash
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

4-16  Oracle Fusion Middleware REST API for Oracle Platform Security Services
The following shows an example of the request body in JSON format.

```json
{
  "stripeName": "myStripe",
  "keystoreName": "myKeystore",
  "keyAlias": "myKeyAlias",
  "keySize": "56",
  "algorithm": "DES",
  "keystorePassword": "myPwd",
  "keyPassword": "myKeyPwd"
}
```

**Example of Response Header**

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

```json
{
  "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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stripeName</td>
<td>Name of the stripe.</td>
<td>Query</td>
</tr>
<tr>
<td>keystoreName</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>keyAlias</td>
<td>Alias of the secret key.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;returnKeyInResponse&quot;</td>
<td>Whether the key should be returned in the output.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
<td>Header</td>
</tr>
<tr>
<td>&quot;keyPassword&quot;</td>
<td>Password for the key.</td>
<td>Header</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

The response body contains information about the certificate, including:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Property n&quot;</td>
<td>List of secret key properties, where n serves as an index that starts at 1 and is incremented by 1 for each additional property.</td>
</tr>
</tbody>
</table>

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.

```bash
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 shows an example of 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 shows an example of the response body in JSON format.

```json
{
   "Property 1": "DES"
}
```
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keyAlias&quot;</td>
<td>Alias of the secret key.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
<td>Header</td>
</tr>
<tr>
<td>&quot;keyPassword&quot;</td>
<td>Password for the key.</td>
<td>Header</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ERROR_CODE&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>&quot;ERROR_MSG&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>

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.


Example of Response Header

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

HTTP/1.1 204 No Content
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: application/json

The request body contains the details of the create request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name for the KSS keystore.</td>
</tr>
<tr>
<td>&quot;keyAlias&quot;</td>
<td>Alias for the secret key.</td>
</tr>
<tr>
<td>&quot;keySize&quot;</td>
<td>Size measured in bits of the of the key used in cryptographic algorithm.</td>
</tr>
<tr>
<td>&quot;algorithm&quot;</td>
<td>Controls the cryptographic characteristics of the algorithms that are used when securing messages</td>
</tr>
<tr>
<td>&quot;DN&quot;</td>
<td>Distinguished name for the key</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
</tr>
<tr>
<td>&quot;keyPassword&quot;</td>
<td>Password for the key.</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ERROR_CODE&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>&quot;ERROR_MSG&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>

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 shows an example of the request body in JSON format.

```json
{
    "stripeName" : "myStripe",
    "keystoreName" : "myKeystore",
    "keyAlias" : "myKeyAlias",
    "keySize" : "56",
    "algorithm" : "DES",
    "DN" : "MyDistinguishedName",
    "keystorePassword" : "myPwd",
    "keyPassword" : "myKeyPwd"
}
```

Example of Response Header
The following shows an example of 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 shows an example of the response body in JSON format.

```json
{
    "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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keyAlias&quot;</td>
<td>Alias of the secret key.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
<td>Header</td>
</tr>
<tr>
<td>&quot;keyPassword&quot;</td>
<td>Password for the key.</td>
<td>Header</td>
</tr>
</tbody>
</table>

**Response Body**

| Media Types: | application/json |

**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 shows an example of the response header. For more about the HTTP status codes, see "HTTP Status Codes for HTTP Methods."

HTTP/1.1 200 OK
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keyalias&quot;</td>
<td>Alias of the secret key.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
<td>Header</td>
</tr>
<tr>
<td>&quot;keyPassword&quot;</td>
<td>Password for the key.</td>
<td>Header</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ERROR_CODE&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>&quot;ERROR_MSG&quot;</td>
<td>If &quot;STATUS&quot; is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>

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 shows an example of the response header. For more about the HTTP status codes, see "HTTP Status Codes for HTTP Methods."

HTTP/1.1 204 No Content
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>stripeName</td>
<td>Name of the stripe.</td>
<td>Query</td>
</tr>
<tr>
<td>keystoreName</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>keyStorePassword</td>
<td>Password for the key store.</td>
<td>Header</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_CODE</td>
<td>If 'STATUS' is set to 'Failed', provides the error code.</td>
</tr>
<tr>
<td>ERROR_MSG</td>
<td>If 'STATUS' is set to 'Failed', provides the contents of the error message.</td>
</tr>
<tr>
<td>STATUS</td>
<td>Status of operation. For example, 'Succeeded' or 'Failed'.</td>
</tr>
</tbody>
</table>

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.


Example of Response Header

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

HTTP/1.1 204 No Content
DELETE Keystore Service KSS Keystore Method
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" on page 2-2.

<table>
<thead>
<tr>
<th>Section</th>
<th>Method</th>
<th>Resource Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST Trust Service Issue Token Method</td>
<td>POST</td>
<td>/opss/v2/trustService</td>
</tr>
<tr>
<td>POST Trust Service Validate Token Method</td>
<td>POST</td>
<td>/opss/v2/trustService</td>
</tr>
</tbody>
</table>
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: | application/json |

The request body contains the details of the create request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol*</td>
<td>The trust protocol. Only the embedded protocol is supported.</td>
<td>No</td>
</tr>
<tr>
<td>tokenType*</td>
<td>The type of token. Supported token types are: SAML, SAML2, and JWT.</td>
<td>Yes</td>
</tr>
<tr>
<td>username*</td>
<td>The user name for which the token is issued.</td>
<td>Yes</td>
</tr>
<tr>
<td>tokenSigningMethod*</td>
<td>The 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)</td>
<td>Yes</td>
</tr>
<tr>
<td>confirmationMethod*</td>
<td>The 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)</td>
<td>Yes</td>
</tr>
<tr>
<td>scdAddress*</td>
<td>The subject confirmation data address. The network address/location from which an attesting entity can present the assertion. (SAML2 only)</td>
<td>Yes</td>
</tr>
<tr>
<td>addAuthenticatingAuthorities*</td>
<td>A list of identity providers trusted by the requester to authenticate the presenter. (SAML2 only)</td>
<td>Yes</td>
</tr>
</tbody>
</table>
POST Trust Service Issue Token Method

Creating and Validating Trust Tokens

Table 5–1 (Cont.) Trust Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nameIdFormat</code></td>
<td>Defines the name identifier formats supported by the identity provider. Name identifiers are a way for providers to communicate with each other regarding a user.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>- urn:oasis:names:tc:SAML:2.0:nameid-format:transient</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- urn:oasis:names:tc:SAML:2.0:nameid-format:kerberos</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- urn:oasis:names:tc:SAML:2.0:nameid-format:entity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(SAML and SAML2 only)</td>
<td></td>
</tr>
<tr>
<td><code>idd</code></td>
<td>The identity domain</td>
<td>Yes</td>
</tr>
<tr>
<td><code>expirationDate</code></td>
<td>The 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</td>
<td>Yes</td>
</tr>
<tr>
<td><code>appliesTo</code></td>
<td>The scope (endpoint target) to which the token applies</td>
<td>No</td>
</tr>
<tr>
<td><code>additionalClaims</code></td>
<td>JWT claims to add to the claim segment. This attribute is only used with the JWT-Token type.</td>
<td>No</td>
</tr>
</tbody>
</table>

**cURL Example**

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

```bash
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 shows an example of the request body in JSON format.

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

**Example of Response Header**
The following shows an example of 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**

```
<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>
```

The request body contains the details of the create request:

<table>
<thead>
<tr>
<th>Table 5–2 Trust Attributes</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>token</code></td>
<td>The identity token.</td>
<td>Yes</td>
</tr>
<tr>
<td><code>protocol</code></td>
<td>The trust protocol. Only the ws-trust protocol is supported.</td>
<td>No</td>
</tr>
<tr>
<td><code>tokenType</code></td>
<td>The type of token. Supported token types are: SAML, SAML2, and JWT.</td>
<td>Yes</td>
</tr>
<tr>
<td><code>username</code></td>
<td>The user name for which the token is issued.</td>
<td>Yes</td>
</tr>
<tr>
<td><code>tokenSigningMethod</code></td>
<td>The 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)</td>
<td>Yes</td>
</tr>
<tr>
<td><code>confirmationMethod</code></td>
<td>The 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)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Response Body**

```
<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/json</th>
</tr>
</thead>
</table>
```

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>username</code></td>
<td>The user name for which the token is issued</td>
</tr>
<tr>
<td><code>idd</code></td>
<td>The identity domain</td>
</tr>
</tbody>
</table>
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 shows an example of the request body in JSON format.

```
{
  "token": "eyJhbGciOiJRUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzY290F2guaW81LCJlci5yZmFsc2hlaW1lc3M6IiwiZ2FuaWFfcGc6IiwiYXN5cnZpYSJCb3VmbGVnaW5hZmFsc2hlaW1lc3MiOjEzMDA4MTszODAsImlhdCI6MTY1NDk1MjIwMCwiZXhwIjoxNTMxMjIyMDE2fQ.03f329983b83f7d9a9f5f6f85305880101d5e402afafa20154d094a229f7578",
  "protocol": "ws-trust",
  "tokenType": "JWT",
  "username": "john.doe",
  "tokenSigningMethod": "RS-256",
  "confirmationMethod": "bearer"
}
```

Example of Response Header

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

```
{
  "username": "john.doe",
  "idd": "cisco",
  "expirationDate": "2015-10-19T12:08:56.235-0700",
}
```
Oracle Platform Security Services (OPSS) uses the XACML 3.0 REST profile based authorization service to manage authorization. You can manage authorization using REST.

<table>
<thead>
<tr>
<th>Section</th>
<th>Method</th>
<th>Resource Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET PDP Link Method</td>
<td>GET</td>
<td>/opss/v2/authz/xacml/</td>
</tr>
<tr>
<td>POST Policy Decision Method</td>
<td>POST</td>
<td>/opss/v2/authz/xacml/</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;rel&quot;</td>
<td>The PDP definition provider</td>
</tr>
<tr>
<td>&quot;href&quot;</td>
<td>The PDP link.</td>
</tr>
</tbody>
</table>

Media Types: application/json or application/xml

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

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

Example of Response Body with JSON

The following shows an example of 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

Example of Response Body with XML

The following shows an example of the response body when using XML.

```xml
<resources xmlns=http://ietf.org/ns/home-documents
xmlns: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

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/xacml+json;version=3.0 or application/xacml+xml;version=3.0</th>
</tr>
</thead>
</table>

Response Body

<table>
<thead>
<tr>
<th>Media Types:</th>
<th>application/xacml+json;version=3.0 or application/xacml+xml;version=3.0</th>
</tr>
</thead>
</table>

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 -H Connection:close

Example of Request with JSON

The following shows an example of the request body when using JSON.

```json
{
   "Request": {
      ...
   }
}
```

Example of Response Body with JSON

The following shows an example of the response body when using JSON.

```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 shows an example of the request body when using XML.

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

Example of Response with XML
The following shows an example of the response body when using XML.

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