Oracle® Fusion Middleware
REST API for Managing Credentials and Keystores with Oracle Web Services Manager
12c Release 12.2.1
E64994-01

October 2015
Documentation that describes how to use the Oracle Web Services Manager REST API for credential store, keystore, and trust store management.
## Contents

**Preface**

What's New In This Guide

### Part I  Getting Started with the REST API

1 **About the REST API**
   1.1 Introducing the REST API
   1.2 URL Structure for Security Stores
   1.3 Creating and Managing Oracle WSM Instances Using REST
   1.4 Authenticating REST Resources
   1.5 HTTP Status Codes for HTTP Methods

2 **Use Cases for the REST API**
   2.1 Managing the Credential Store Framework Using the REST API
   2.2 Managing JKS Keystores Using the REST API
   2.3 Managing KSS Keystores Using the REST API
   2.4 Managing Token Issuer Trust Using the REST API

### Part II  REST API Reference

3 **Managing Credentials in the Credential Store**
   - POST Credential Method
   - GET Credential Method
   - PUT Credential Method
   - DELETE Credential Method

4 **Managing Java Keystore Keystores**
   - GET All Aliases Trusted Certificate JKS Keystore Method
   - POST Specified Alias Trusted Certificate JKS Keystore Method
   - POST PKCS#7 Trusted Certificate JKS Keystore Method
   - GET Specified Alias Trusted Certificate JKS Keystore Method
   - DELETE Trusted Certificate JKS Keystore Method
5 Viewing and Managing Keystore Service Keystores

POST New KSS Keystore Method ................................................................. 5-2
POST Import KSS Keystore Method ............................................................ 5-4
PUT Password Update KSS Keystore Method ........................................... 5-6
POST Trusted Certificate KSS Keystore Method ....................................... 5-8
GET Stripe KSS Keystores Method ............................................................ 5-10
GET Alias KSS Keystore Method ............................................................... 5-11
GET Trusted Certificate KSS Keystore Method ......................................... 5-12
DELETE Trusted Certificate KSS Keystore Method ................................... 5-14
POST Secret Key KSS Keystore ................................................................. 5-16
GET Secret Key Properties KSS Keystore Method ................................... 5-18
DELETE Keystore Service KSS Keystore Method ....................................... 5-19

6 Managing Token Issuer Trust Configurations

POST TrustDocument Name Method ......................................................... 6-2
POST Domain Trusted Issuers and Distinguished Name Lists Method ....... 6-3
POST Document Trusted Issuers and Distinguished Name Lists Method .... 6-5
GET All Trusted Issuer and Distinguished Name Lists Method .................. 6-8
GET Specified Document Trusted Issuer and Distinguished Name Lists Method ..... 6-10
POST Token Attribute Rule Distinguished Name Method (Domain Context) .... 6-12
POST Token Attribute Rule Distinguished Name Method (Document Context) ... 6-16
GET All Token Attribute Rules Method ..................................................... 6-20
GET Specified Document Token Attribute Rules Method ......................... 6-23
Import TrustDocument Name Configurations Method .............................. 6-26
GET TrustDocument Method ..................................................................... 6-28
DELETE Trust Document Method ............................................................ 6-29

A Summary of REST APIs
This preface describes the document accessibility features and conventions used in this guide—REST API for Managing Credentials and Keystores with Oracle Web Services Manager.

**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.

**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.

**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><strong>italic</strong></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 Web Services Manager in Oracle Fusion Middleware 12c (12.2.1).

New and Changed Features for 12c (12.2.1)

This release introduces new Web Services Manager REST APIs to manage Token Issuer Trust artifacts.

The current release of the REST API for Managing Credentials and Keystores with Oracle Web Services Manager documents the new REST APIs. It also includes bug updates and editorial corrections.
Part I

Getting Started with the REST API

Get started using the Oracle Fusion Middleware REST API for managing credentials and keystores.

Part I contains the following chapters:

- Chapter 1, "About the REST API"
- Chapter 2, "Use Cases for the REST API"
About the REST API

This section introduces the Oracle Fusion Middleware representational state transfer (REST) API for managing credentials and keystores.

This chapter includes the following sections:

■ Introducing the REST API
■ URL Structure for Security Stores
■ Creating and Managing Oracle WSM Instances Using REST
■ Authenticating REST Resources
■ HTTP Status Codes for HTTP Methods

1.1 Introducing the REST API

The credential and keystore management REST API provides endpoints for creating and configuring credential stores, keystores, and trust stores for your domain or web services.

You can access the REST endpoints through Web browsers and client applications.

You can also use the Oracle WSM REST endpoints in REST client applications that are developed in languages such as:

■ JavaScript
■ Ruby
■ Perl
■ Java
■ JavaFX

Before using the REST API, you need to understand a few important concepts, as described in the following sections.

1.2 URL Structure for Security Stores

Use the following URL to manage security stores:

http[s]://host:port/idaas/contextpath/admin/v1/resource

Where:

■ host:port—Host and port where Oracle Fusion Middleware is running.
Creating and Managing Oracle WSM Instances Using REST

1.3 Creating and Managing Oracle WSM Instances Using REST

The Oracle WSM REST endpoints support standard methods for creating and managing Oracle WSM instances.

<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>

1.4 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 (for example, weblogic and welcome1) using the -u cURL option.

curl -i -X GET -u weblogic:welcome1 http://myhost:7001/idaas/platform/admin/v1/keystore

For POST 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 weblogic:welcome1 -H keyPassword:bXlQd2Qy -H keystorePassword:c1dBdMQ== http://myhost:7001/idaas/platform/admin/v1/keystoreservice/certificates?"stripeName=myStripe&keystoreName=myKeystore&keyAlias=myAlias"

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>
</tbody>
</table>
## HTTP Status Codes for HTTP Methods

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 (__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.</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>
Use Cases for the REST API

This section demonstrates several use cases using the REST API.

- Managing the Credential Store Framework Using the REST API
- Managing JKS Keystores Using the REST API
- Managing KSS Keystores Using the REST API
- Managing Token Issuer Trust Using the REST API

2.1 Managing the Credential Store Framework Using the REST API

You can view and manage the credential store framework using the REST APIs described in the following use case. Specifically, this use case shows you how to:

- Create a credential in the credential store
- View all credentials in the credential store
- Delete a credential from the credential store

**Note:** For more information about credential store management, see "Configuring the Credential Store" in Security and Administrator’s Guide for Web Services.

To manage the credential store framework using the REST API:

1. Create a credential in the credential store framework by performing the following steps:
   
a. Create a JSON document, `createcred.json`, that defines the credential that you want to create.

   The following shows an example of the request document. In this example, the name of the credential map is `default`, the credential key is `myKey`, and the username and password credentials are `myUsr` and `myPwd`, respectively.

   ```
   {
   "username" : "username",
   "credential" : "pwd",
   "key" : "mykey",
   "map" : "oracle.wsm.security"
   }
   ```

   For more information about the request attributes, see "POST Credential Method" on page 3-2.
b. Using cURL, create a credential in the credential store framework, passing
the JSON document defined in the previous step.

```bash
curl -i -X POST -u username:password --data @createcred.json -H
Content-Type:application/json
http://myhost:7001/idaas/platform/admin/v1/credential
```

The following shows an example of the response indicating the request
succeeded.

```json
{
  "STATUS": "Succeeded"
}
```

For more information, see "POST Credential Method" on page 3-2.

2. View all credentials in the credential store.

```bash
curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/credential
```

The following shows an example of the response, showing all credentials in the
credential store:

```json
{
  "CSF_MAP_NAME": "CSF_KEY_NAME",
  "default": "systemuser",
  "oracle.wsm.security": [
    "sign-csf-key",
    "jwt-sign-csf-key",
    "owsmtest.credentials",
    "basic.client.credentials",
    "weblogic-csf-key",
    "enc-csf-key",
    "mykey",
    "dummy-pwd-csf-key",
    "weblogic-kerberos-csf-key",
    "keystore-csf-key",
    "weblogic-windowsdomain-csf-key",
    "oratest-csf-key",
    "csr-csf-key",
    "invalid-csf-key",
    "ca-signed-sign-csf-key"
  ]
}
```

For more information, see "GET Credential Method" on page 3-4.

3. Delete the credential from the credential store.

```bash
curl -i -X DELETE -u username:password
http://myhost:7001/idaas/webservice/admin/v1/credential?key=mykey&map=oracle.wsm.security"
```

You must pass query parameters to define the map and key names associated with
the credential store that you want to delete. For more information, see "DELETE
Credential Method" on page 3-8.

The following shows an example of the response indicating the request succeeded.

```json
{
  "STATUS": "Succeeded"
}
```
2.2 Managing JKS Keystores Using the REST API

You can view and manage Java Keystore (JKS) certificates within the current domain using the REST APIs described in the following use case. Specifically, this use case shows you how to:

- View all aliases in the JKS keystore.
- Import a trusted certificate into the JKS keystore.
- View a trusted certificate in the JKS keystore.
- Delete a trusted certificate from the JKS keystore.


To manage JKS keystores using the REST API:

1. View all of the aliases that currently exist in the JKS keystore within the current domain:

   ```
curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/keystore
   ```

   The following shows an example of the response, showing all aliases in the JKS keystore.

   ```
   {
   "aliases":"oratest,orakey,testkey,jkstest,ms-oauthkey"
   }
   ```

   For more information, see "GET All Aliases Trusted Certificate JKS Keystore Method" on page 4-2.

2. Import the trusted certificate into the JKS keystore at the specified alias, by performing the following steps:

   a. Create a JSON document, `importjks.json`, that defines the trusted certificate to import into the JKS keystore.

      The following shows an example of the request document. In this example, the trusted certificate provided must be Base64-encoded and the component type must be set to JKS for this release.

      ```
      {
      "component":"JKS",
      "certificate":"MIIC7DCCAggAwIBAgIElhBSjALBgcgjkjOoAQDBQAwSD8KMAgGA1UEBh..."}
      ```
Managing JKS Keystores Using the REST API

2-4

For more information about the request attributes, see “POST Specified Alias Trusted Certificate JKS Keystore Method” on page 4-3.

b. Using cURL, import the trusted certificate, specifying the alias of the trusted key to be imported, mytestkey, and passing the JSON request document defined in the previous step.

curl -i -X POST -u username:password -H Content-type:application/json --data @importjks.json http://myhost:7001/idaas/platform/admin/v1/keystore/mytestkey

The following shows an example of the response indicating the request succeeded.

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

For more information, see "POST Specified Alias Trusted Certificate JKS Keystore Method" on page 4-3.

3. View the trusted certificate that you imported in step 3:

curl -i -X GET -u username:password http://myhost:7001/idaas/platform/admin/v1/keystore/mytestkey

The following shows an example of the response, showing the details for the trusted certificate.

```json
{
  "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":"Thu Jul 03 04:00:16 PDT 2014",
  "NOT_AFTER":"Wed Oct 01 04:00:16 PDT 2014",
  "SERIAL_NO":"1784168778",
  "SIGNING_ALGORITHM":"1.2.840.10040.4.3",
  "CONTENT":"-----BEGIN CERTIFICATE-----
PjEiBhQg是个二进制字符串..."}
```

For more information, see "POST Specified Alias Trusted Certificate JKS Keystore Method" on page 4-3.
2.3 Managing KSS Keystores Using the REST API

You can view and manage Keystore Service (KSS) keystores using the REST APIs described in the following use case. Specifically, this use case shows you how to:

- Create a KSS keystore
- View all KSS keystores for a stripe
- Import a trusted certificate into the KSS keystore
- View a trusted certificate in the JKS keystore
- Delete the KSS keystore

**Note:** For more information about KSS keystore management, see "Configuring the OPSS Keystore Service for Message Protection" in Security and Administrator’s Guide for Web Services.

To manage KSS keystores using the REST API:

1. Create a KSS keystore by performing the following steps:
   a. Create a JSON document, `createkss.json`, that defines the KSS keystore that you want to create.

   The following shows an example of the request document. In this example, the KSS stripe and keystore names are `myStripe` and `myKeystore`, respectively; the password for the KSS keystore is `mypwd`; and the KSS keystore created is not permission-based.
For more information about the request attributes, see "POST New KSS Keystore Method" on page 5-2.

b. Using cURL, create a KSS keystore, passing the JSON document defined in the previous step.

curl -i -X POST -u username:password -H Content-Type:application/json --data @createkss.json
http://myhost:7001/idaas/platform/admin/v1/keystoreservice

The following shows an example of the response indicating the request succeeded.

```
{
  "STATUS": "Succeeded"
}
```

For more information, see "POST New KSS Keystore Method" on page 5-2.

2. View all KSS keystores for a stripe to confirm the KSS keystore was created.

curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/keystoreservice/myStripe

The following shows an example of the response, showing all KSS keystores in the stripe:

```
{
  "keystore 1":"myKeystore"
}
```

For more information, see "GET Stripe KSS Keystores Method" on page 5-10.

3. Import a trusted certificate into the KSS keystore by performing the following steps:

a. Create a JSON document, importkss.json, that defines the details of the trusted certificate that you want to import into the KSS keystore.

The following shows an example of the request document. In this example, the KSS keystore is identified by its stripe and keystore names, myStripe and myKeystore, respectively; the KSS keystore password, mypwd, is required; the alias for the key is myAlias; the certificate is defined as a TrustedCertificate; and keystoreEntry specifies the encrypted certificate contents.

```
{
  "keyAlias" : "myAlias",
  "keystoreEntry" : "MIIC7DCCAqggAwIBAgIe1hBSjALBgcqhkj0OAwDQAwSDEKMAgGA1UEBhMBTEKMAgGA1UEChMB
  "keystore" : "myKeystore",
  "pwd" : "mypwd",
  "permission" : "false"
}
```

For more information, see "POST New KSS Keystore Method" on page 5-2.
Managing KSS Keystores Using the REST API

Use Cases for the REST API

For more information about the request attributes, see "POST Trusted Certificate KSS Keystore Method" on page 5-8.

b. Using cURL, import a trusted certificate into the KSS keystore, passing the JSON document defined in the previous step.

```
curl -i -X POST -u username:password -H Content-Type:application/json --data @importcertkss.json http://myhost:7001/idaas/platform/admin/v1/keystoreservice/certificates
```

The following shows an example of the response indicating the request succeeded.

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

For more information, see "POST Trusted Certificate KSS Keystore Method" on page 5-8.

4. View the trusted certificate that you just imported into the KSS keystore.

```
curl -i -X GET -u username:password -H keystorePassword:cHdkMQ== http://myhost:7001/idaas/platform/admin/v1/keystoreservice/certificates?stripeName=myStripe&keystoreName=myKeystore&keyAlias=myAlias&keystoreEntryType=TrustedCertificate
```

You must pass query parameters to define the stripe name, keystore name and entry type, and alias name associated with the trusted certificate you want to view.

The following shows an example of the response, showing the details of the trusted certificate.

```
{
  "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",
  "CONTENT":"-----BEGIN CERTIFICATE-----
MIIC7DCCAqqgAwIBAgIEOosLyjALBgcqhkjOOAQDBQAwS
EKMAgGA1UEBhMBcjEKMAgGA1UECBMB
cjEKMAgGA1UEBxMBcjEKMAgGA1UEChMBcjEKMAgGA1UECwM
ckJEKMAgGA1UEAxM5uAjeW0xNDA3nkJwU0QQlMTFapw0xNDEwMjw0OTQlMTFamEXxJcAJBgvNBAYTA
IxJcAIBgNVAgkTAXIcXaIbicWVpY2UwNjAwMDEyMzAyNzUxMjA9
-----END CERTIFICATE-----
```

Use Cases for the REST API 2-7
Managing Token Issuer Trust Using the REST API

You can view and manage token issuer trust using the REST APIs described in the following use case. Specifically, this use case shows you how to:

■ View all trusted issuers
■ Create a trusted issuer
■ Create a token attribute rule
■ Delete a trusted issuer
■ Create a trust document

**Note:** For more information about token issuer trust management, see "Defining Trusted Issuers and a Trusted DN List for Signing Certificates" in *Security and Administrator’s Guide for Web Services*.

To manage token issuer trust using the REST API:

1. Create a trusted issuer document.

   ```bash
curl -i -X POST -u username:password
   http://myhost:7001/idaaS/webservice/admin/v1/trustdocument?
documentName=myTrustDocument&displayName=myTrustDocument
   ``

You must pass query parameters to define the document and display names for the trusted issuer document.

---

2.4 Managing Token Issuer Trust Using the REST API

You can view and manage token issuer trust using the REST APIs described in the following use case. Specifically, this use case shows you how to:

■ View all trusted issuers
■ Create a trusted issuer
■ Create a token attribute rule
■ Delete a trusted issuer
■ Create a trust document

You must pass query parameters to define the stripe and keystore name of the KSS keystore you want to delete. For more information, see "DELETE Keystore Service KSS Keystore Method" on page 5-19.

The following shows an example of the response indicating the request succeeded.

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

---

For more information, see "GET Trusted Certificate KSS Keystore Method" on page 5-12.

5. Delete the KSS keystore.

```bash
curl -i -X DELETE -u username:password
   -H keystorePassword:CHdkMQ==
   http://myhost:7001/idaas/platform/admin/v1/keystoreservice?
   stripeName=myStripe&keystoreName=myKeystore
```

---

To manage token issuer trust using the REST API:

1. Create a trusted issuer document.

   ```bash
curl -i -X POST -u username:password
   http://myhost:7001/idaaS/webservice/admin/v1/trustdocument?
documentName=myTrustDocument&displayName=myTrustDocument
   ``

You must pass query parameters to define the document and display names for the trusted issuer document.
The following shows an example of the response indicating the request succeeded.

```
{
    "STATUS": "Succeeded",
    "Result": "New Token Issuer Trust document named "myTrustDocument" created."
}
```

For more information, see "POST TrustDocument Name Method" on page 6-2.

2. Create the trusted issuers and DN lists, by performing the following steps:

a. Create a JSON document, `createtrust.json`, that defines the trusted issuers and distinguished name (DN) lists that you want to create.

The following shows an example of the request document. In this example, the following types of trusted issuers are created: SAML holder-of-key, SAML sender vouches, and JSON Web Token (JWT). For each trusted issuer, the name and DN list is defined.

```
{
    "saml-trusted-dns":
    {
        "saml-hok-trusted-dns":
        {
            "issuer":
            {
                "-name": "www.oracle.com",
                "dn": [ "wls1", ]
            }
        },
    },
    "saml-sv-trusted-dns":
    {
        "issuer":
        {
            "-name": "www.oracle.com",
            "dn": [ "wls2", ]
        }
    },
    "jwt-trusted-issuers":
    {
        "issuer":
        {
            "-name": "www.oracle.com",
            "dn": [ "CN=orakey, OU=Orakey,O=Oracle, C=US", ]
        }
    }
}
```

For more information about the request attributes, see "POST Domain Trusted Issuers and Distinguished Name Lists Method" on page 6-3.

b. Using cURL, create the trusted issuers and DN lists, passing the JSON document defined in step 2.

```
curl -i -X POST -u username:password --data @createtrust.json -H Content-Type:application/json
http://myhost:7001/idaas/webservice/admin/v1/trust/issuers
```
The following shows an example of the response body indicating the request succeeded.

```json
{
  "STATUS": "Succeeded"
}
```

For more information, see "POST Domain Trusted Issuers and Distinguished Name Lists Method" on page 6-3.

3. Create a JSON document, createtoken.json, that defines the token attribute rules for the trusted DN lists.

The following shows an example of the request document. In this example:

- Create a separate "token-attribute-rule" entry for each trusted DN list for which you want to create a token attribute rule.
- Specify filters for the name-id and user attributes, as required.

For more information about the request attributes, see "POST Token Attribute Rule Distinguished Name Method (Domain Context)" on page 6-12.

```json
{
  "token-attribute-rules":
  {
    "token-attribute-rule":
    [
      {
        "-dn": "cn=orcladmin, o=oracle",
        "name-id":{
          "filter":
          {
            "value": [ "filter1" ]
          },
          "mapping":
          {
            "user-attribute": "val3",
            "user-mapping-attribute": "val4"
          }
        },
        "attributes":
        [
          {
            "-name": "tenant1",
            "attribute":
            {
              "filter":
              {
                "value": [ "filter1", "filter2"
              },
              "mapping":{
                "user-attribute": "val1",
                "user-mapping-attribute": "val2"
              }
            }
          }
        ]
      }
    ]
  }
}
```
4. Create the token attribute rules for the trusted DN lists, passing the JSON document defined in step 4.

   `curl -i -X POST -u username:password --data @createrule.json
   http://myhost:7001/idaas/webservice/admin/v1/trust/token`

   The following shows an example of the response body indicating the request succeeded.

   ```json
   { "STATUS": "Succeeded" }
   ```

   For more information, see “POST Token Attribute Rule Distinguished Name Method (Domain Context)” on page 6-12.

5. View the configuration details for the trusted issuer.

   `curl -i -X GET -u username:password
   http://myhost:7001/idaas/platform/admin/v1/trustdocument?"documentName=myTrustDocument"

   The following shows an example of the response body, showing the configuration details:

   ```json
   { "STATUS":"Succeeded", "Result":"List of token issuer trust documents in the Repository:
   Details of the document matching your request:
   Name : myTrustDocument	Display Name : myTrustDocument	Status : DOCUMENT_STATUS_COMMITED
   List of trusted issuers for this type:	None
   List of Token Attribute Rules:	None"
   }
   ```

   For more information, see “GET TrustDocument Method” on page 6-28.

6. Delete the trusted issuer document.

   `curl -i -X DELETE -u username:password
   http://myhost:7001/idaas/webservice/admin/v1/trustdocument?"documentName=myTrustDocument&displayName=myTrustDocument"

   You must pass query parameters to define the document and display names for the trusted issuer document that you want to delete. For more information, see “DELETE Credential Method” on page 3-8.

   The following example shows the contents of the response body.

   ```json
   { "STATUS": "Succeeded", "Result": "Token Issuer Trust document named "myTrustDocument" deleted from the repository." }
   ```
Review details about the Oracle Fusion Middleware REST API for managing credentials and keystores.

Part II contains the following chapters:

- Chapter 3, "Managing Credentials in the Credential Store"
- Chapter 4, "Managing Java Keystore Keystores"
- Chapter 5, "Viewing and Managing Keystore Service Keystores"
- Chapter 6, "Managing Token Issuer Trust Configurations"
- Appendix A, "Summary of REST APIs"
Managing Credentials in the Credential Store

Oracle Web Services Manager (WSM) uses the Credential Store Framework (CSF) to manage the credentials in a secure form. You can view and manage the credential store using a set of representational state transfer (REST) resources, as summarized below.

Before using the REST API, you need to understand how to access the REST resources and other important concepts. See "About the REST API" on page 1-1.

For more information about credential store management, see "Configuring the Credential Store" in Security and Administrator’s Guide for Web Services.

<table>
<thead>
<tr>
<th>Section</th>
<th>Method</th>
<th>Resource Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST Credential Method</td>
<td>POST</td>
<td>/idaas/platform/admin/v1/credential</td>
</tr>
<tr>
<td>GET Credential Method</td>
<td>GET</td>
<td>/idaas/platform/admin/v1/credential</td>
</tr>
<tr>
<td>PUT Credential Method</td>
<td>PUT</td>
<td>/idaas/platform/admin/v1/credential</td>
</tr>
<tr>
<td>DELETE Credential Method</td>
<td>DELETE</td>
<td>/idaas/platform/admin/v1/credential</td>
</tr>
</tbody>
</table>
POST Credential Method

Use the POST method to create a new credential in the domain credential store.

REST Request

POST /idaas/platform/admin/v1/credential

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;credential&quot;</td>
<td>Password for the credential.</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;key&quot;</td>
<td>Name of the key.</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;map&quot;</td>
<td>Name of the map (folder).</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;username&quot;</td>
<td>Username for the credential.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Response Body

| Media Types:      | application/json     |

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 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 http://myhost:7001/idaas/platform/admin/v1/credential
```

Example of Request Body

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

```
{
    "username" : "username",
    "credential" : "credential",
    "key" : "mykey",
    "map" : "oracle.wsm.security"
}
```
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 Credential Method

Use the GET method to view all credentials in the domain credential store.

REST Request

GET /idaas/platform/admin/v1/credential

Response Body

Media Types: application/json

The response body contains information about all credentials in the credential store, including:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;CSF_MAP_NAME&quot;</td>
<td>Name of the credential store map.</td>
</tr>
<tr>
<td>&quot;default&quot;</td>
<td>List of keys in the default credential map.</td>
</tr>
<tr>
<td>&quot;oracle.wsm.security&quot;</td>
<td>List of keys in the Oracle Web Services Manager (Oracle WSM) security map.</td>
</tr>
</tbody>
</table>

cURL Example

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

curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/credential

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
{
    "CSF_MAP_NAME": "CSF_KEY_NAME",
    "default": "systemuser",
    "oracle.wsm.security": [
        "sign-csf-key",
        "jwt-sign-csf-key",
        "owsmtest.credentials",
        "basic.client.credentials",
        "weblogic-csf-key",
        "enc-csf-key",
        "mykey",
        "dummy-pwd-csf-key",
        "weblogic-kerberos-csf-key",
        "keystore-csf-key",
        "weblogic-windowsdomain-csf-key",
        "oratest-csf-key",
        "csr-csf-key",
    ],
}```
`'invalid-csf-key',
'ca-signed-sign-csf-key'`
PUT Credential Method

Use the PUT method to update a credential in the domain credential store.

REST Request

PUT /idaas/platform/admin/v1/credential

Request Body

Media Types: application/json

The request body contains the details of the update request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>credential</em></td>
<td>Updated password for the key in the keystore.</td>
<td>Yes</td>
</tr>
<tr>
<td><em>key</em></td>
<td>Name of the key that you want to modify. The key must exist.</td>
<td>Yes</td>
</tr>
<tr>
<td><em>map</em></td>
<td>Name of the map (folder) that you want to modify.</td>
<td>Yes</td>
</tr>
<tr>
<td><em>username</em></td>
<td>Username for the key in the keystore.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

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><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 update a credential in the credential store by submitting a PUT request on the REST resource using cURL.

```
curl -i -X PUT -u username:password --data @updatecred.json -H 'Content-Type:application/json' http://myhost:7001/idaas/platform/admin/v1/credential
```

Example of Request Body

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

```
{
  "username" : "username",
  "credential" : "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'
}
```
DELETE Credential Method

Use the Delete method to delete a credential from the domain credential store.

REST Request

DELETE /idaas/platform/admin/v1/credential

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;key&quot;</td>
<td>Name of the key for the credential that you want to delete.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;map&quot;</td>
<td>Name of the map (folder) for the credential that you want to delete.</td>
<td>Query</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 credential from the credential store by submitting a DELETE request on the REST resource using cURL.

curl -i -X DELETE -u username:password http://myhost:7001/idaas/platform/admin/v1/credential?"key=mykey&map=oracle.wsm.security"

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"
}
```
Managing Java Keystore Keystores

You can view and manage Java Keystore (JKS) keystores within a domain using a set of representational state transfer (REST) resources, as summarized below.

Before using the REST API, you need to understand how to access the REST resources and other important concepts. See "About the REST API" on page 1-1.


<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Resource Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET All Aliases Trusted Certificate JKS Keystore Method</td>
<td>GET</td>
<td>/idaas/platform/admin/v1/keystore</td>
</tr>
<tr>
<td>POST Specified Alias Trusted Certificate JKS Keystore Method</td>
<td>POST</td>
<td>/idaas/platform/admin/v1/keystore/{alias}</td>
</tr>
<tr>
<td>POST PKCS#7 Trusted Certificate JKS Keystore Method</td>
<td>POST</td>
<td>/idaas/platform/admin/v1/keystore/pkcs7/{alias}</td>
</tr>
<tr>
<td>GET Specified Alias Trusted Certificate JKS Keystore Method</td>
<td>GET</td>
<td>/idaas/platform/admin/v1/keystore/{alias}</td>
</tr>
<tr>
<td>DELETE Trusted Certificate JKS Keystore Method</td>
<td>DELETE</td>
<td>idaaS/platform/admin/v1/keystore/{alias}</td>
</tr>
</tbody>
</table>
GET All Aliases Trusted Certificate JKS Keystore Method

Use the GET method to get all aliases for the trusted certificate entries in the JKS keystore.

REST Request

GET /idaas/platform/admin/v1/keystore

Response Body

Media Types: application/json

The response body contains the list of aliases:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;aliases&quot;</td>
<td>Comma-separated list of aliases.</td>
</tr>
</tbody>
</table>

cURL Example

The following example shows how to view all aliases for the trusted certificate entries in the JKS keystore by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/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/1.1 200 OK

Example of Response Body

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

```
{
  "aliases":"oratest,orakey,testkey,jkstest,ms-oauthkey"
}
```
POST Specified Alias Trusted Certificate JKS Keystore Method

Use the POST method to import a trusted certificate at the specified alias into the JKS keystore. The certificate must be Base64 encoded.

REST Request

POST /idaas/platform/admin/v1/keystore/{alias}

Parameters

The following table summarizes the POST request parameter.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>Alias of the trusted certificate to be imported.</td>
<td>Path</td>
</tr>
<tr>
<td></td>
<td>The alias will be created. The alias must not already exist in the JKS keystore; otherwise, the request will fail.</td>
<td></td>
</tr>
</tbody>
</table>

Request Body

Media Types: application/json

The request body contains the details of the import request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'certificate'</td>
<td>Base64-encoded certificate.</td>
</tr>
<tr>
<td>'component'</td>
<td>Component to which the certificate is imported. This value must be set to JKS.</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>'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>
<tr>
<td>'SUBJECT_DN'</td>
<td>Subject DN list that was imported.</td>
</tr>
</tbody>
</table>

cURL Example

The following example shows how to import a trusted certificate into the JKS keystore by submitting a POST request on the REST resource using cURL.
Example of Request Body
The following shows an example of the request body in JSON format.

```
{
  "component": "JKS",
  "certificate": "MIIC7DCCAqqgAwIBAgIElhBSjALBgcqhkjOQADBQAwSDEKMA4GA1UEBhMBeTEKAgGA1UECzAI
wMTzA2PedXMDE0MTAwMTZaMEgxCTAIbGNYbG9yZXkxCTAIbGNYbG9yZXkxCTAICgkUDWJwYX..."}
```

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"
}
```
POST PKCS#7 Trusted Certificate JKS Keystore Method

Use the POST method to import a PKCS#7 trusted certificate or a certificate chain associated with a private key indicated by the specified alias into the JKS keystore.

REST Request

POST /idaas/platform/admin/v1/keystore/pkcs7/{alias}

Parameters

The following table summarizes the POST request parameter.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias</td>
<td>Alias of the private key for which the trusted PKCS#7 certificate will be imported. The alias must already be in the JKS keystore.</td>
<td>Path</td>
</tr>
</tbody>
</table>

Request Body

Media Types: application/json

The request body contains the details of the import request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'certificate'</td>
<td>Base64-encoded certificate.</td>
</tr>
<tr>
<td>'component'</td>
<td>Component to which the certificate is imported. This value must be set to JKS.</td>
</tr>
<tr>
<td>'keyPassword'</td>
<td>Password for the private 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>'ERROR_CODE'</td>
<td>If 'STATUS' is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>'ERROR_MSG'</td>
<td>If 'STATUS' is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>'STATUS'</td>
<td>Status of operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
<tr>
<td>'SUBJECT_DN'</td>
<td>Subject DN list that was imported.</td>
</tr>
</tbody>
</table>

cURL Example

The following example shows how to import a trusted PKCS#7 certificate into the JKS keystore by submitting a POST request on the REST resource using cURL.
curl -i -X POST -u username:password --data @importjkscert.json -H
Content-Type:application/json
http://myhost:7001/idaas/platform/admin/v1/keystore/pkcs7/myprivatekey

Example of Request Body

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

```json
{
  "component":"JKS",
  "certificate": "MIIC7DCCAqgAwIBAgIeAlhBSjalBgcqhkjOQADBQAwSDEKMAgGA1UEBhMBeTEKAgGA1UEC
iMBeTEKAgGA1UEBxMBeTEKAgGA1UECwMBeTEKAgGA1UEAxMBeTEKAgGA1UEA4MBeTEKAgGA1UEA2MBeTEKAgGA1UEA1MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGA1UEA6MBeTEKAgGA1UEA8MBeTEKAgGA1UEAAMBeTEKAgGA1UEA2MBeTEKAgGA1UEA0MBeTEKAgGA1UEAENBeTEKAgGa
"keyPassword": "myprivatekeypwd"
}
```

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",
  "SUBJECT_DN": "CN=y,OU=y,O=y,L=y,ST=y,C=y"
}
```
GET Specified Alias Trusted Certificate JKS Keystore Method

Use to GET method to view details of the trusted certificate at the specified alias in the JKS keystore.

If the alias specifies a `keyStore.TrustedCertificateEntry`, the details of the trusted certificate are returned. If the alias specifies a `KeyStore.PrivateKeyEntry`, the first certificate in the trusted certificate chain is returned.

REST Request

GET /idaas/platform/admin/v1/keystore/{alias}

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>alias</td>
<td>Name of alias for which you want to view a trusted certificate.</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>'CONTENT'</td>
<td>Contents of the Base64-encoded certificate.</td>
</tr>
<tr>
<td>'Extensions'</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>- Value</td>
</tr>
<tr>
<td>'ISSUER_DN'</td>
<td>List of trusted distinguished names.</td>
</tr>
<tr>
<td>'NOT_AFTER'</td>
<td>Date the certificate expires.</td>
</tr>
<tr>
<td>'NOT_BEFORE'</td>
<td>Date the certificate is activated.</td>
</tr>
<tr>
<td>'SERIAL_NO'</td>
<td>Serial number of the JKS keystore.</td>
</tr>
<tr>
<td>'SIGNATURE'</td>
<td>Base64-encoded signature key.</td>
</tr>
<tr>
<td>'SIGNING_ALGORITHM'</td>
<td>Signing algorithm for the alias.</td>
</tr>
<tr>
<td>'SUBJECT_DN'</td>
<td>Subject distinguished names list.</td>
</tr>
</tbody>
</table>

cURL Example

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

curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/keystore/mytestkey
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
{
  "SUBJECT_DN": "CN=weblogic,OU=Testkey for JKS Mbean test,O=Oracle,L=testcity,ST=teststate,C=us",
  "ISSUER_DN": "CN=weblogic,OU=Testkey for JKS Mbean test,O=Oracle,L=testcity,ST=teststate,C=us",
  "NOT_BEFORE": "Tue Jun 25 02:20:38 PDT 2013",
  "NOT_AFTER": "Wed Nov 27 01:20:38 PST 2052",
  "SERIAL_NO": "1372152038",
  "SIGNING_ALGORITHM": "1.2.840.113549.1.1.5",
  "CONTENT": "-----BEGIN CERTIFICATE-----
MIIDeDCCAmCgAwIBAgIEUclg5jANBgkqhkiG9w0BAQQUFAAECMAgAwIBAgIExV
BaA2ggEYMB0GA1UdEwEB/wQFMA0GA1UEChMGT3JhY2xlMREwDwYDVQQHEwZP
cmFjbGUxIzABBgNVBAcTCHRlc3RjaXR5MQ8wDQYDVQQKEwZPcmFjbGUxIzAB
BgNVBAsTGlRlc3RrZXkgZm9yIEpLUyBNYmVhbiB0ZXN0MREwDwYDVQQDEwZ3
ZWJsb2dpYzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJtmzlqcnU+
9d4Oior0FIOfcgpI/EOflbkTicUr1Aefy18EDn1+U7hDQ+XpPzrsndjAtFbc
mxghGwueP7zIzTXUg6JGQrFAHcZC09wEvvz9v9rPchklU39hDc2hKvFk9g3b
xwUy9x53/klD895g23c296Z/CqYmOZ9G3z3hSxVv4x3J6bb6zJyfOgdSjXG
8xj21NYWe3hjGdGt4GmZzovDyJ67Xrs3m9swEIyoOvL6V00fCueYVxN1+JF
-----END CERTIFICATE-----",
  "SIGNATURE": "eAnH79sc8iMkLZRKWzh4vQ==",
  "Extensions": {
    "subjectKeyIdExtension": {
      "oid": 2.5.29.14,
      "critical": false,
      "value": "329b98f6b225e92ca52513d3bfc43ee02aa9121"
    }
  }
}
```
DELETE Trusted Certificate JKS Keystore Method

Use the Delete method to delete a trusted certificate (keyStore.TrustedCertificateEntry) with the specified alias from the JKS keystore. You cannot delete the keyStore.PrivateKeyEntry.

REST Request

DELETE /idaas/platform/admin/v1/keystore/{alias}

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>alias</td>
<td>Alias of the trusted certificate entry to be deleted.</td>
<td>Path</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 &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>'ERROR_MSG'</td>
<td>If 'STATUS' is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>'STATUS'</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 http://myhost:7001/idaas/platform/admin/v1/keystore/testalias
```

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'
}
```
You can view and manage Keystore Service (KSS) keystores using a set of representational state transfer (REST) resources, as summarized below.

Before using the REST API, you need to understand how to access the REST resources and other important concepts. See "About the REST API" on page 1-1.

For more information about KSS keystore management, see "Configuring the OPSS Keystore Service for Message Protection" in Security and Administrator’s Guide for Web Services.

<table>
<thead>
<tr>
<th>Table 5–1  KSS Keystore Management REST Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Section</strong></td>
</tr>
<tr>
<td>POST New KSS Keystore Method</td>
</tr>
<tr>
<td>POST Import KSS Keystore Method</td>
</tr>
<tr>
<td>PUT Password Update KSS Keystore Method</td>
</tr>
<tr>
<td>POST Trusted Certificate KSS Keystore Method</td>
</tr>
<tr>
<td>GET Stripe KSS Keystores Method</td>
</tr>
<tr>
<td>GET Alias KSS Keystore Method</td>
</tr>
<tr>
<td>GET Trusted Certificate KSS Keystore Method</td>
</tr>
<tr>
<td>DELETE Trusted Certificate KSS Keystore Method</td>
</tr>
<tr>
<td>POST Secret Key KSS Keystore Method</td>
</tr>
<tr>
<td>GET Secret Key Properties KSS Keystore Method</td>
</tr>
<tr>
<td>DELETE Keystore Service KSS Keystore Method</td>
</tr>
</tbody>
</table>
POST New KSS Keystore Method

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

REST Request

POST /idaas/platform/admin/v1/keystoreservice

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

Response Body

| Media Types: | application/json |

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 http://myhost:7001/idaas/platform/admin/v1/keystoreservice

Example of Request Body

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

```json
{
    "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.

```text
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

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

Response Body

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alias n*</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>ERROR_CODE*</td>
<td>If <em>STATUS</em> is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>ERROR_MSG*</td>
<td>If <em>STATUS</em> is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>STATUS*</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.

```bash
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"
http://myhost:7001/idaas/platform/admin/v1/keystoreservice/keystore
```

**Example of Response Header**

The following shows an example of the response header.

```
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 /idaas/platform/admin/v1/keystoreservice

Request Body

Media Types: application/json

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;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>
<tr>
<td>&quot;stripe&quot;</td>
<td>Name of the stripe.</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

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 http://myhost:7001/idaas/platform/admin/v1/keystoreservice

Example of Request Body

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

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

HTTP/1.1 200 OK

Example of Response Body
The following shows an example of 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 /idaas/platform/admin/v1/keystoreservice/certificates`

Request Body

| Media Types: | application/json |

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;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,</td>
</tr>
<tr>
<td></td>
<td>TrustedCertificate, or SecretKey.</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name of the KSS keystore.</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
</tr>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</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>
<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.

```bash
curl -i -X POST -u username:password --data @importcertkss.json -H Content-Type:application/json
http://myhost:7001/idaas/platform/admin/v1/keystoreservice/certificates
```

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

```json
{
    "keyAlias": "myAlias",
    "keystoreEntry": {
        "keyAlias": "myAlias",
        "keystoreEntryType": "TrustedCertificate",
        "keystoreName": "myKeystore",
        "stripeName": "myStripe",
        "keystorePassword": "myPwd"
    }  
}
```

**Example of Response Header**

The following shows an example of the response header.

```
HTTP/1.1 200 OK
```

**Example of Response Body**

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

```json
{
    "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 /idaas/platform/admin/v1/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>stripeName</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 http://myhost:7001/idaas/platform/admin/v1/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
/idaas/platform/admin/v1/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>entryType</td>
<td>Keystore type. Valid values include Certificate, TrustedCertificate, or SecretKey.</td>
<td>Path</td>
</tr>
<tr>
<td>keystoreName</td>
<td>Name of the keystore.</td>
<td>Path</td>
</tr>
<tr>
<td>stripeName</td>
<td>Name of the stripe.</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 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.


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 /idaas/platform/admin/v1/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>keyAlias</td>
<td>Alias for trusted certificate.</td>
<td>Query</td>
</tr>
<tr>
<td>keystoreEntryType</td>
<td>Type of keystore entry. Valid values include Certificate, TrustedCertificate, or CertificateChain.</td>
<td>Query</td>
</tr>
<tr>
<td>keystoreName</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>stripeName</td>
<td>Name of the stripe.</td>
<td>Query</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;CONTENT&quot;</td>
<td>Contents of the Base64-encoded certificate.</td>
</tr>
<tr>
<td>&quot;Extensions&quot;</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>&quot;ISSUER_DN&quot;</td>
<td>List of trusted distinguished names.</td>
</tr>
<tr>
<td>&quot;NOT_AFTER&quot;</td>
<td>Date the certificate expires.</td>
</tr>
<tr>
<td>&quot;NOT_BEFORE&quot;</td>
<td>Date the certificate is activated.</td>
</tr>
<tr>
<td>&quot;SERIAL_NO&quot;</td>
<td>Serial number of the JKS keystore.</td>
</tr>
<tr>
<td>&quot;SIGNATURE&quot;</td>
<td>Base64-encoded signature key.</td>
</tr>
<tr>
<td>&quot;SIGNING_ALGORITHM&quot;</td>
<td>Signing algorithm for the alias.</td>
</tr>
<tr>
<td>&quot;SUBJECT_DN&quot;</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.
GET Trusted Certificate KSS Keystore Method

```bash
curl -i -X GET -u username:password -H keystorePassword:chdkMQ==
http://myhost:7001/idaas/platform/admin/v1/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.

```json
{
  "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",
  "CONTENT": "-----BEGIN CERTIFICATE-----
MIIC7DCCAqqAwIBAgIEoosLyjALBgcqhkjOOAQDBQAwS
EKMAgGA1UEBhMBcjcjEKMAgGA1UEBIBMUDA1EdJSMBc9jEKMAgGA1UECwMc
cjEKMAgGA1UEAwmBUgjAwIBAgIMbDg0MQswCQYDVQQGEwJQYXNoZWN0
TaQ0MREwGzAOBgNVBAsTC0FSCcwc2VjcmV0IENsb3NlZCBhbiBQb3Jp
bGxkIENsb3NlZCBhbmQgTWFuZ3JvcmQgQ1ND
-----END 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 /idaas/platform/admin/v1/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>keyAlias</td>
<td>Alias for the certificate in the KSS keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>keystoreName</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>stripeName</td>
<td>Name of stripe.</td>
<td>Query</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
http://myhost:7001/idaas/platform/admin/v1/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 /idaas/platform/admin/v1/keystoreservice/secretkey

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;algorithm&quot;</td>
<td>Controls the cryptographic characteristics of the algorithms that are used when securing messages.</td>
</tr>
<tr>
<td>&quot;keyAlias&quot;</td>
<td>Alias for the secret key.</td>
</tr>
<tr>
<td>&quot;keyPassword&quot;</td>
<td>Password for the secret key.</td>
</tr>
<tr>
<td>&quot;keySize&quot;</td>
<td>Size measured in bits of the key used in cryptographic algorithm.</td>
</tr>
<tr>
<td>&quot;keystoreName&quot;</td>
<td>Name for the KSS keystore.</td>
</tr>
<tr>
<td>&quot;keystorePassword&quot;</td>
<td>Password for the KSS keystore.</td>
</tr>
<tr>
<td>&quot;stripeName&quot;</td>
<td>Name of the stripe.</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 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 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 http://myhost:7001/idaas/platform/admin/v1/keystoreservice/secretkey

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',
    '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 /idaas/platform/admin/v1/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>keyAlias</td>
<td>Alias of the secret key.</td>
<td>Query</td>
</tr>
<tr>
<td>keystoreName</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>stripeName</td>
<td>Name of the stripe.</td>
<td>Query</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.

```
curl -i -X GET -u username:password -H keystorePassword:bXlQd2Q= -H keyPassword:bXlLZXlQd2Q= http://myhost:7001/idaas/platform/admin/v1/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.

```
{
  "Property 1":"DES"
}
```
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 /idaas/platform/admin/v1/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>keystoreName</td>
<td>Name of the keystore.</td>
<td>Query</td>
</tr>
<tr>
<td>stripeName</td>
<td>Name of the stripe.</td>
<td>Query</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.

curl -i -X DELETE -u username:password -H keystorePassword:bXlQd2Q=
http://myhost:7001/idaas/platform/admin/v1/keystoreservice?
stripeName=myStripe&keystoreName=myKeystore

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
You can view and manage token issuer trust configurations using a set of representational state transfer (REST) resources, as summarized below.

Before using the REST API, you need to understand how to access the REST resources and other important concepts. See "About the REST API" on page 1-1.

For more information about token issuer trust management, see "Defining Trusted Issuers and a Trusted DN List for Signing Certificates" in Security and Administrator’s Guide for Web Services.

<table>
<thead>
<tr>
<th>Section</th>
<th>Method</th>
<th>Resource Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST TrustDocument Name Method</td>
<td>POST</td>
<td>/idaas/webservice/admin/v1/trustdocument</td>
</tr>
<tr>
<td>POST Domain Trusted Issuers and Distinguished Name Lists Method</td>
<td>POST</td>
<td>/idaas/webservice/admin/v1/trust/issuers</td>
</tr>
<tr>
<td>POST Document Trusted Issuers and Distinguished Name Lists Method</td>
<td>POST</td>
<td>/idaas/webservice/admin/v1/trust/issuers</td>
</tr>
<tr>
<td>GET All Trusted Issuer and Distinguished Name Lists Method</td>
<td>GET</td>
<td>/idaas/webservice/admin/v1/trust/issuers</td>
</tr>
<tr>
<td>GET Specified Document Trusted Issuer and Distinguished Name Lists Method</td>
<td>GET</td>
<td>/idaas/webservice/admin/v1/trust/issuers</td>
</tr>
<tr>
<td>POST Token Attribute Rule Distinguished Name Method (Domain Context)</td>
<td>POST</td>
<td>/idaas/webservice/admin/v1/trust/token</td>
</tr>
<tr>
<td>POST Token Attribute Rule Distinguished Name Method (Document Context)</td>
<td>POST</td>
<td>/idaas/webservice/admin/v1/trust/token</td>
</tr>
<tr>
<td>GET All Token Attribute Rules Method</td>
<td>GET</td>
<td>/idaas/webservice/admin/v1/trust/token</td>
</tr>
<tr>
<td>GET Specified Document Token Attribute Rules Method</td>
<td>GET</td>
<td>/idaas/webservice/admin/v1/trust/token</td>
</tr>
<tr>
<td>Import TrustDocument Name Configurations Method</td>
<td>POST</td>
<td>/idaas/webservice/admin/v1/trustdocument/import</td>
</tr>
<tr>
<td>GET TrustDocument Method</td>
<td>GET</td>
<td>/idaas/webservice/admin/v1/trustdocument</td>
</tr>
<tr>
<td>DELETE Trust Document Method</td>
<td>DELETE</td>
<td>/idaas/webservice/admin/v1/trustdocument</td>
</tr>
</tbody>
</table>
POST TrustDocument Name Method

Use the Post method to create a trusted issuer document.

REST Request

POST /idaas/webservice/admin/v1/trustdocument

Parameters

The following table summarizes the POST request parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;displayName&quot;</td>
<td>Display name for the document.</td>
<td>Query</td>
</tr>
<tr>
<td>&quot;documentName&quot;</td>
<td>Name of the document.</td>
<td>Query</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;Result&quot;</td>
<td>Details of the operation results.</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 trusted issuer document by submitting a POST request on the REST resource using cURL.

curl -i -X POST -u username:password http://myhost:7001/idaas/webservice/admin/v1/trustdocument?"documentName=myTrustDocument&displayName=myTrustDocument"

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",  "Result": "New Token Issuer Trust document named "myTrustDocument" created."}
POST Domain Trusted Issuers and Distinguished Name Lists Method

Use the POST method to create trusted issuers and distinguished name (DN) lists for signing certificates in a domain context (that is, it applies to the entire domain).

REST Request

POST /idaas/webservice/admin/v1/trust/issuers

Request Body

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>'dn'</td>
<td>List of DN values to be added to the trusted issuer. For each DN, use a string that conforms to RFC 2253, as described at the following URL: <a href="http://www.ietf.org/rfc/rfc2253.txt">http://www.ietf.org/rfc/rfc2253.txt</a></td>
<td>Yes</td>
</tr>
<tr>
<td>'issuer'</td>
<td>Groups information about a trusted issuer.</td>
<td>Yes</td>
</tr>
<tr>
<td>'-name'</td>
<td>Name of the trusted issuer. For example, <a href="http://www.yourcompany.com">www.yourcompany.com</a>. The default value for the predefined SAML client policies is <a href="http://www.oracle.com">www.oracle.com</a>.</td>
<td>Yes</td>
</tr>
<tr>
<td>'jwt-trusted-dns'</td>
<td>Groups information about JSON Web Token (JWT) trusted issuers.</td>
<td>No</td>
</tr>
<tr>
<td>'saml-hok-trusted-dns'</td>
<td>Groups information about SAML holder-of-key trusted issuers.</td>
<td>No</td>
</tr>
<tr>
<td>'saml-sv-trusted-dns'</td>
<td>Groups information about SAML sender vouches trusted issuers.</td>
<td>No</td>
</tr>
<tr>
<td>'saml-trusted-dns'</td>
<td>Groups the trusted issuers and DN lists.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Response Body

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ERROR_CODE'</td>
<td>If 'STATUS' is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>'ERROR_MSG'</td>
<td>If 'STATUS' is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>'STATUS'</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 trusted issuers and DN lists by submitting a POST request on the REST resource using cURL.

curl -i -X POST -u username:password --data @createtrust.json -H Content-Type:application/json http://myhost:7001/idaas/webservice/admin/v1/trust/issuers

Example of Request Body

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

```json
{
  "saml-trusted-dns": {
    "saml-hok-trusted-dns": {
      "issuer": [
        {
          "-name": "www.oracle.com",
          "dn": [ "wls1", ]
        }
      ]
    },
    "saml-sv-trusted-dns": {
      "issuer": [
        {
          "-name": "www.oracle.com",
          "dn": [ "wls2", ]
        }
      ]
    },
    "jwt-trusted-issuers": {
      "issuer": [
        {
          "-name": "www.oracle.com",
          "dn": [ "CN=orakey, OU=Orakey, O=Oracle, C=US", ]
        }
      ]
    }
  }
}
```

Example of Response Header

The following shows an example of the response header.

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 Document Trusted Issuers and Distinguished Name Lists Method

Use the POST method to create trusted issuers and distinguished name (DN) lists for signing certificates in a document context (that is, it applies to a specified document). The trusted issuers will be stored in the specified trusted issuers document.

REST Request

POST /idaas/webservice/admin/v1/trust/issuers/{documentName}

Parameters

The following table summarizes the POST request parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>documentName</td>
<td>Name of trusted issuer document. For information about creating a trusted issuer document, see 'POST TrustDocument Name Method' on page 6-2.</td>
<td>Query</td>
</tr>
</tbody>
</table>

Request Body

Media Types: application/json

The request body contains the details of the add request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>'dn'</td>
<td>List of DN values to be added to the trusted issuer. For each DN, use a string that conforms to RFC 2253, as described at the following URL: <a href="http://www.ietf.org/rfc/rfc2253.txt">http://www.ietf.org/rfc/rfc2253.txt</a></td>
<td>Yes</td>
</tr>
<tr>
<td>'issuer'</td>
<td>Groups information about a trusted issuer.</td>
<td>Yes</td>
</tr>
<tr>
<td>'-name'</td>
<td>Name of the trusted issuer. For example, <a href="http://www.yourcompany.com">www.yourcompany.com</a>. The default value for the predefined SAML client policies is <a href="http://www.oracle.com">www.oracle.com</a>.</td>
<td>Yes</td>
</tr>
<tr>
<td>'jwt-trusted-dns'</td>
<td>Groups information about JSON Web Token (JWT) trusted issuers.</td>
<td>No</td>
</tr>
<tr>
<td>'saml-hok-trusted-dns'</td>
<td>Groups information about SAML holder-of-key trusted issuers.</td>
<td>No</td>
</tr>
<tr>
<td>'saml-sv-trusted-dns'</td>
<td>Groups information about SAML sender vouches trusted issuers.</td>
<td>No</td>
</tr>
<tr>
<td>'saml-trusted-dns'</td>
<td>Groups the trusted issuers and DN lists.</td>
<td>Yes</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 trusted issuers and DN lists by submitting a POST request on the REST resource using cURL.

```
curl -i -X POST -u username:password --data @createtrust.json -H Content-Type:application/json http://myhost:7001/idaas/webservice/admin/v1/trust/issuers/mydocument
```

**Example of Request Body**

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

```
{
  "saml-trusted-dns":
  {
    "saml-hok-trusted-dns":
    {
      "issuer": [
        {
          "-name": "www.oracle.com",
          "dn": [ "wls1", ]
        }
      ],
    },
    "saml-sv-trusted-dns":
    {
      "issuer": [
        {
          "-name": "www.oracle.com",
          "dn": [ "wls2", ]
        }
      ],
    },
    "jwt-trusted-issuers":
    {
      "issuer": [
        {
          "-name": "www.oracle.com",
          "dn": [ "CN=orakey, OU=Orakey,O=Oracle, C=US", ]
        }
      ],
    }
  }
}
```

**Example of Response Header**

The following shows an example of the response header.

```
HTTP/1.1 200 OK
```

**Example of Response Body**

The following shows an example of the response body in JSON format.
{  
  "STATUS": "Succeeded",  
}
GET All Trusted Issuer and Distinguished Name Lists Method

Use the GET method to view a trusted issuer and its distinguished name (DN) lists for all domain documents.

REST Request

GET /idaas/webservice/admin/v1/trust/issuers

Response Body

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;dn&quot;</td>
<td>List of DN values to be added to the trusted issuer.</td>
</tr>
<tr>
<td>&quot;issuer&quot;</td>
<td>Groups information about a trusted issuer.</td>
</tr>
<tr>
<td>&quot;-name&quot;</td>
<td>Name of the trusted issuer.</td>
</tr>
<tr>
<td>&quot;jwt-trusted-dns&quot;</td>
<td>Groups information about JSON Web Token (JWT) trusted issuers.</td>
</tr>
<tr>
<td>&quot;saml-hok-trusted-dns&quot;</td>
<td>Groups information about SAML holder-of-key trusted issuers.</td>
</tr>
<tr>
<td>&quot;saml-sv-trusted-dns&quot;</td>
<td>Groups information about SAML sender vouches trusted issuers.</td>
</tr>
<tr>
<td>&quot;saml-trusted-dns&quot;</td>
<td>Groups the DN lists.</td>
</tr>
</tbody>
</table>

cURL Example

The following example shows how to view a trusted issuer and its DN lists by submitting a GET request on the REST resource using cURL.

```
curl -i-X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/trust/issuers
```

Example of Response Header

The following shows an example of the response header.

```
HTTP/1.1 200 OK
```

Example of Response Body

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

```json
{
  "saml-trusted-dns":
  {
    "saml-hok-trusted-dns":
    {
      "issuer": [
        {
          "-name": "www.oracle.com",
          "dn": [ "wls1", ]
        }
      ]
    }
  }
}
```
```
}"
],
"saml-sv-trusted-dns":
{
  "issuer": [
    {
      "-name": "www.oracle.com",
      "dn": [ "wls2", ]
    }
  ]
},
"jwt-trusted-issuers":
{
  "issuer": [
    {
      "-name": "www.oracle.com",
      "dn": [ "CN=orakey, OU=Orakey, O=Oracle, C=US", ]
    }
  ]
}
}
```
GET Specified Document Trusted Issuer and Distinguished Name Lists Method

Use the GET method to view a trusted issuer and its distinguished name (DN) lists based on the document name provided.

REST Request

GET /idaas/webservice/admin/v1/trust/issuers/{documentName}

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>documentName</td>
<td>Name of document for which you want to view issuer and DN lists.</td>
<td>Path</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

The response body contains information about the trusted issuer and DN lists, including:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;dn&quot;</td>
<td>List of DN values to be added to the trusted issuer.</td>
</tr>
<tr>
<td>&quot;issuer&quot;</td>
<td>Groups information about a trusted issuer.</td>
</tr>
<tr>
<td>&quot;-name&quot;</td>
<td>Name of the trusted issuer.</td>
</tr>
<tr>
<td>&quot;jwt-trusted-dns&quot;</td>
<td>Groups information about JSON Web Token (JWT) trusted issuers.</td>
</tr>
<tr>
<td>&quot;saml-hok-trusted-dns&quot;</td>
<td>Groups information about SAML holder-of-key trusted issuers.</td>
</tr>
<tr>
<td>&quot;saml-sv-trusted-dns&quot;</td>
<td>Groups information about SAML sender vouches trusted issuers.</td>
</tr>
<tr>
<td>&quot;saml-trusted-dns&quot;</td>
<td>Groups the DN lists.</td>
</tr>
</tbody>
</table>

cURL Example

The following example shows how to view a trusted issuer and its DN lists by submitting a GET request on the REST resource using cURL.

curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/trust/issuers/mydocument

Example of Response Header

The following shows an example of the response header.

HTTP/1.1 200 OK

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

```
{
    "saml-trusted-dns": {
        "saml-hok-trusted-dns": {
            "issuer": [
                {
                    "-name": "www.oracle.com",
                    "dn": [ "wls1", ]
                }
            ]
        },
        "saml-sv-trusted-dns": {
            "issuer": [
                {
                    "-name": "www.oracle.com",
                    "dn": [ "wls2", ]
                }
            ]
        },
        "jwt-trusted-issuers": {
            "issuer": [
                {
                    "-name": "www.oracle.com",
                    "dn": [ "CN=orakey, OU=Orakey, O=Oracle, C=US", ]
                }
            ]
        }
    }
}
```
POST Token Attribute Rule Distinguished Name Method (Domain Context)

Use the POST method to create a token attribute rule for a trusted distinguished name (DN) for a domain context (that is, it applies to the entire domain). This operation can be performed by the REST service or client. Only token attribute mapping is supported on the client side.

REST Request

POST /idaas/webservice/admin/v1/trust/token

Request Body

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes</td>
<td>Groups the constraints filter and mapping attributes for trusted users.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This attribute is not required on the client side.</td>
</tr>
<tr>
<td>-dn</td>
<td>On the service side, set this value to a trusted DN for which you are configuring an attribute rule. Use a string that conforms to RFC 2253, as described at the following URL: <a href="http://www.ietf.org/rfc/rfc2253.txt">http://www.ietf.org/rfc/rfc2253.txt</a></td>
</tr>
<tr>
<td></td>
<td>On the client side, set this value to a URL of the domain hosting the targeted services using the following format: http(s)://host or http(s)://host/root. For example, if you set this value to <a href="https://messaging.us2.com/">https://messaging.us2.com/</a>, then the attribute rule applies to all service invocations with the service URL of the form <a href="https://messaging.us2.com/">https://messaging.us2.com/</a>&lt;path&gt;.</td>
</tr>
<tr>
<td>filter</td>
<td>Defines the constraint values for trusted users and attributes.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>mapping</td>
<td>Defines the mapping attributes for trusted users.</td>
</tr>
<tr>
<td>-name</td>
<td>Name of the attribute rule.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>name-id</td>
<td>Defines the users that are accepted for the trusted DN.</td>
</tr>
<tr>
<td>token-attribute-rule</td>
<td>Groups information about a single token attribute rule.</td>
</tr>
<tr>
<td>token-attribute-rules</td>
<td>Groups information about all token attribute rules.</td>
</tr>
<tr>
<td>user-attribute</td>
<td>Defines the user attribute that the trusted DN can assert.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>user-mapping-attribute</td>
<td>Defines the user mapping attribute that the trusted DN can assert.</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><code>*ERROR_CODE*</code></td>
<td>If <code>*STATUS*</code> is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td><code>*ERROR_MSG*</code></td>
<td>If <code>*STATUS*</code> is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td><code>*STATUS*</code></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 token attribute rule for a trusted DN by submitting a POST request on the REST resource using cURL.

```bash
curl -i -X POST -u username:password --data @createrule.json http://myhost:7001/idaas/webservice/admin/v1/trust/token
```

**Example of Request Body - Service Side**

The following shows an example of the request body in JSON format for creating a token attribute rule for a trusted DN on the service side.

```json
{
    "token-attribute-rules":
    {
        "token-attribute-rule":
        [
            {
                "-dn": "cn=orcladmin,o=oracle",
                "name-id":{
                    "filter":{
                        "value": ["filter1"]
                    },
                    "mapping":{
                        "user-attribute": "val3",
                        "user-mapping-attribute": "val4"
                    }
                },
                "attributes":
                [
                    {
                        "-name": "tenant1",
                        "attribute":
                    }
                ]
            }
        ]
    }
}
```
Example of Request Body - Client Side

The following shows an example of the request body in JSON format for creating a token attribute rule on the client side.

```json
{
    "token-attribute-rules":
    {
        "token-attribute-rule":
        [
            {
                "dn": "https://messaging.us2.com/",
                "name-id": {
                    "mapping": {
                        "user-mapping-attribute": "mail"
                    }
                }
            },
            {
                "dn": "https://messaging.us2.com/mysvcInstance1-acme/",
                "name-id": {
                    "mapping": {
                        "user-mapping-attribute": "uid"
                    }
                }
            }
        ]
    }
}
```

Example of Response Header

The following shows an example of the response header.

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 Token Attribute Rule Distinguished Name Method (Document Context)

Use the POST method to create a token attribute rule for a trusted distinguished name (DN) for a document context (that is, it applies to a specified document). This operation can be performed by the REST service or client. Only token attribute mapping is supported on the client side.

REST Request

POST /idaas/webservice/admin/v1/trust/token/{documentName}

Parameters

The following table summarizes the POST request parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>documentName</td>
<td>Name of document for which you want to create a token attribute rule.</td>
<td>Path</td>
</tr>
</tbody>
</table>

Request Body

Media Types: application/json

The request body contains the details of the add request:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;attributes&quot;</td>
<td>Groups the constraints filter and mapping attributes for trusted users.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This attribute is not required on the client side.</td>
</tr>
<tr>
<td>&quot;-dn&quot;</td>
<td>On the service side, set this value to a trusted DN for which you are configuring an attribute rule. Use a string that conforms to RFC 2253, as described at the following URL: <a href="http://www.ietf.org/rfc/rfc2253.txt">http://www.ietf.org/rfc/rfc2253.txt</a>. On the client side, set this value to a URL of the domain hosting the targeted services using the following format: http(s)://host or http(s)://host/root. For example, if you set this value to <a href="https://messaging.us2.com/">https://messaging.us2.com/</a>, then the attribute rule applies to all service invocations with the service URL of the form <a href="https://messaging.us2.com/">https://messaging.us2.com/</a>&lt;path&gt;</td>
</tr>
<tr>
<td>&quot;filter&quot;</td>
<td>Defines the constraint values for trusted users and attributes.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>&quot;mapping&quot;</td>
<td>Defines the mapping attributes for trusted users.</td>
</tr>
<tr>
<td>&quot;-name&quot;</td>
<td>Name of the attribute rule.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>&quot;name-id&quot;</td>
<td>Defines the users that are accepted for the trusted DN.</td>
</tr>
<tr>
<td>&quot;token-attribute-rule&quot;</td>
<td>Groups information about a single token attribute rule.</td>
</tr>
<tr>
<td>&quot;tokn-attribute-rules&quot;</td>
<td>Groups information about all token attribute rules.</td>
</tr>
</tbody>
</table>
Response Body

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'user-attribute'</td>
<td>Defines the user attribute that the trusted DN can assert.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>'user-mapping-attribute'</td>
<td>Defines the user mapping attribute that the trusted DN can assert.</td>
</tr>
<tr>
<td>'value'</td>
<td>Defines values for the constraint filter attribute. This value can be a</td>
</tr>
<tr>
<td></td>
<td>full name or name pattern with a wildcard character ('*'), such as:</td>
</tr>
<tr>
<td></td>
<td>'yourTrusted*'. Multiple values must be separated by a comma.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: This attribute is not applicable on the client side.</td>
</tr>
</tbody>
</table>

**cURL Example**

The following example shows how to create a token attribute rule for a trusted DN by submitting a POST request on the REST resource using cURL.

```bash
curl -i -X POST -u username:password --data @createrule.json http://myhost:7001/idaas/webservice/admin/v1/trust/token/mydocument
```

**Example of Request Body - Service Side**

The following shows an example of the request body in JSON format for creating a token attribute rule for a trusted DN on the service side.

```json
{
  "token-attribute-rules":
  {
    "token-attribute-rule":
    {
      "dn": "cn=orcladmin,o=oracle",
      "name-id":{
        "filter":{
          "value": [ "filter1" ]
        },
        "mapping":{
          "user-attribute": "val3",
          "user-mapping-attribute": "val4"
        }
      }
    }
  }
}
```
Example of Request Body - Client Side

The following shows an example of the request body in JSON format for creating a token attribute rule on the client side.

```json
{
    "token-attribute-rules": {
        "token-attribute-rule": {
            "token-attribute-rule": {
                "dn": "https://messaging.us2.com/",
                "name-id": {
                    "mapping": {
                        "user-mapping-attribute": "mail"
                    }
                }
            },
            "token-attribute-rule": {
                "dn": "https://messaging.us2.com/mysvcInstance1-acme/",
                "name-id": {
                    "mapping": {
                        "user-mapping-attribute": "uid"
                    }
                }
            }
        }
    }
}
```

Example of Response Header
The following shows an example of the response header.

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 All Token Attribute Rules Method

Use the GET method to view all token attribute rules for a domain context (applies to entire domain). This operation can be performed by the REST service or client. Only token attribute mapping is supported on the client side.

REST Request

GET /idaas/webservice/admin/v1/trust/token

Response Body

| Media Types: | application/json |

The response body contains information about all token attribute rules, including:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;attributes&quot;</td>
<td>Groups the constraints filter and mapping attributes for trusted users.</td>
</tr>
<tr>
<td></td>
<td>Note: This attribute is not required on the client side.</td>
</tr>
<tr>
<td>&quot;-dn&quot;</td>
<td>On the service side, trusted DN for which you are configuring an attribute rule. The string conforms to RFC 2253, as described at the following URL: <a href="http://www.ietf.org/rfc/rfc2253.txt">http://www.ietf.org/rfc/rfc2253.txt</a></td>
</tr>
<tr>
<td></td>
<td>On the client side, URL specified using the following format: http(s)://host or http(s)://host/root</td>
</tr>
<tr>
<td>&quot;filter&quot;</td>
<td>Defines the filter values for trusted users and attributes. You can enter a complete name or a name pattern with a wildcard character (<em>), such as yourTrusted</em>. If you specify multiple attribute filters, each filter should be separated by a comma.</td>
</tr>
<tr>
<td>&quot;mapping&quot;</td>
<td>Defines the mapping attributes for trusted users.</td>
</tr>
<tr>
<td>&quot;-name&quot;</td>
<td>Name of the attribute rule.</td>
</tr>
<tr>
<td>&quot;-note&quot;</td>
<td>This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>&quot;name-id&quot;</td>
<td>Defines the users that are accepted for the trusted DN.</td>
</tr>
<tr>
<td>&quot;token-attribute-rule&quot;</td>
<td>Groups information about a single token attribute rule.</td>
</tr>
<tr>
<td>&quot;token-attribute-rules&quot;</td>
<td>Groups information about all token attribute rules.</td>
</tr>
<tr>
<td>&quot;user-attribute&quot;</td>
<td>Defines the user attribute that the trusted DN can assert.</td>
</tr>
<tr>
<td>&quot;user-mapping-attribute&quot;</td>
<td>Defines the user mapping attribute that the trusted DN can assert.</td>
</tr>
<tr>
<td>&quot;value&quot;</td>
<td>Defines values for the constraint filter attribute. This value can be a full name or name pattern with a wildcard character (<em>), such as ‘yourTrusted</em>’. Multiple values must be separated by a comma.</td>
</tr>
</tbody>
</table>
cURL Example

The following example shows how to view all token attribute rules by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/trust/token
```

Example of Response Header

The following shows an example of the response header.

```
HTTP/1.1 200 OK
```

Example of Response Body—Service Side

The following shows an example of the response body in JSON format for viewing a token attribute rule on the service side.

```
{
    "token-attribute-rules":
    {
        "token-attribute-rule":
        [
            {
                "dn": "cn=orcladmin,o=oracle",
                "attributes":
                [
                    {
                        "name": "tenant1",
                        "attribute":
                        {
                            "filter":
                            {
                                "value": [
                                    "filter1",
                                    "filter2"
                                ]
                            },
                            "mapping":{
                                "user-attribute": "val1",
                                "user-mapping-attribute": "val2"
                            }
                        }
                    }
                ],
                "name-id":{
                    "filter":
                    {
                        "value":[
                            "filter1"
                        ],
                    },
                    "mapping":{
                        "user-attribute": "val3",
                        "user-mapping-attribute": "val4"
                    }
                }
            }
        ]
    }
}
```

Example of Response Body - Client Side
The following shows an example of the response body in JSON format for viewing a token attribute rule on the client side.

```
{
  "token-attribute-rules":
  {
    "token-attribute-rule":
    [
      {
        "-dn": "https://messaging.us2.com/",
        "name-id":{
          "mapping":
          {
            "user-mapping-attribute":"mail"
          }
        },
      },
    ]
  }
}
```
GET Specified Document Token Attribute Rules Method

Use the GET method to view token attribute rules for a specified document. This operation can be performed by the REST service or client. Only token attribute mapping is supported on the client side.

REST Request

GET /idaas/webservice/admin/v1/trust/token/{documentName}

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>documentName</td>
<td>Name of document for which you want to view token attribute rules.</td>
<td>Path</td>
</tr>
</tbody>
</table>

Response Body

Media Types: application/json

The response body contains information about all token attribute rules for the document, including:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'attributes'</td>
<td>Groups the constraints filter and mapping attributes for trusted users.</td>
</tr>
<tr>
<td></td>
<td>Note: This attribute is not required on the client side.</td>
</tr>
<tr>
<td>'-dn'</td>
<td>On the service side, trusted DN for which you are configuring an attribute rule. The string conforms to RFC 2253, as described at the following URL: <a href="http://www.ietf.org/rfc/rfc2253.txt">http://www.ietf.org/rfc/rfc2253.txt</a></td>
</tr>
<tr>
<td></td>
<td>On the client side, URL specified using the following format: http(s)://host or http(s)://host/root</td>
</tr>
<tr>
<td>'filter'</td>
<td>Defines the filter values for trusted users and attributes.</td>
</tr>
<tr>
<td></td>
<td>You can enter a complete name or a name pattern with a wildcard character (<em>), such as yourTrusted</em>. If you specify multiple attribute filters, each filter should be separated by a comma.</td>
</tr>
<tr>
<td>'mapping'</td>
<td>Defines the mapping attributes for trusted users.</td>
</tr>
<tr>
<td></td>
<td>Note: This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>'-name'</td>
<td>Name of the attribute rule.</td>
</tr>
<tr>
<td></td>
<td>Note: This attribute is not applicable on the client side.</td>
</tr>
<tr>
<td>'name-id'</td>
<td>Defines the users that are accepted for the trusted DN.</td>
</tr>
<tr>
<td>'token-attribute-rule'</td>
<td>Defines information about a single token attribute rule.</td>
</tr>
<tr>
<td>'tokn-attribute-rules'</td>
<td>Defines information about all token attribute rules.</td>
</tr>
</tbody>
</table>
The following example shows how to view all token attribute rules by submitting a GET request on the REST resource using cURL.

```
curl -i -X GET -u username:password
http://myhost:7001/idaas/platform/admin/v1/trust/token/mydocument
```

**Example of Response Header**

The following shows an example of the response header.

```
HTTP/1.1 200 OK
```

**Example of Response Body—Service Side**

The following shows an example of the response body in JSON format for viewing a token attribute rule on the service side.

```
{
  "token-attribute-rules":
  {
    "token-attribute-rule":
    {
      "-dn": 'cn=orcladmin,o=oracle',
      "attributes":
      {
        "-name": 'tenant1',
        "attribute":
        {
          "filter":
          {
            "value": [
              "filter1",
              "filter2"
            ],
          },
          "mapping":{
            "user-attribute": "val1",
            "user-mapping-attribute": "val2"
          }
        }
      },
      "name-id":{
        "filter":
        {
          "value": [ "filter1" ]
        },
```
Example of Response Body - Client Side

The following shows an example of the response body in JSON format for viewing a token attribute rule on the client side.

```json
{
  "token-attribute-rules":
  {
    "token-attribute-rule":
    [
      {
        "dn": "https://messaging.us2.com/",
        "name-id":{
          "mapping":
          {
            "user-mapping-attribute":"mail"
          }
        }
      },
      {
        "dn": "https://messaging.us2.com/mysvcInstance1-acme/",
        "name-id":{
          "mapping":
          {
            "user-mapping-attribute":"uid"
          }
        }
      }
    ]
  }
}
```
Import TrustDocument Name Configurations Method

Use the POST method to import trusted issuer configurations, including issuer names, distinguished name (DN) lists, and token attribute rules.

REST Request

POST /idaas/webservice/admin/v1/trustdocument/import

Request Body

The request body contains the details of the import request, in XML format. You must create a trusted issuers document, as described in "POST TrustDocument Name Method" on page 6-2, and pass it using the oratrust:name element. For example:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<oratrust:TokenIssuerTrust
xmlns:oratrust="http://xmlns.oracle.com/wsm/security/trust"
oratrust:name="trustdocument">
  <oratrust:Issuers>
    <oratrust:Issuer oratrust:name="www.oracle.com" oratrust:tokentype="saml.hok">
      <oratrust:TrustedKeys/>
    </oratrust:Issuer>
    <oratrust:Issuer oratrust:name="www.oracle.com" oratrust:tokentype="saml.sv">
      <oratrust:TrustedKeys/>
    </oratrust:Issuer>
    <oratrust:Issuer oratrust:name="www.oracle.com" oratrust:tokentype="jwt">
      <oratrust:TrustedKeys/>
    </oratrust:Issuer>
  </oratrust:Issuers>
  <oratrust:TokenAttributeRules/>
</oratrust:TokenIssuerTrust>
```

Response Body

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

<table>
<thead>
<tr>
<th>Element</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;Result&quot;</td>
<td>Details of the operation results.</td>
</tr>
<tr>
<td>&quot;STATUS&quot;</td>
<td>Status of the operation. For example, &quot;Succeeded&quot; or &quot;Failed&quot;.</td>
</tr>
</tbody>
</table>
cURL Example

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

```bash
curl -i -X POST -u username:password --data @import.xml -H Content-Type:application/xml -H Accept:application/json http://myhost:7001/idaas/platform/admin/v1/trustdocument/import
```

Example of Request Body

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

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<oratrust:TokenIssuerTrust
xmlns:oratrust="http://xmlns.oracle.com/wsm/security/trust" oratrust:name="test">
  <oratrust:Issuers>
    <oratrust:Issuer oratrust:name="www.oracle.com" oratrust:tokentype="saml.hok">
      <oratrust:TrustedKeys/>
    </oratrust:Issuer>
  </oratrust:Issuers>
  <oratrust:Issuers>
    <oratrust:Issuer oratrust:name="www.oracle.com" oratrust:tokentype="saml.sv">
      <oratrust:TrustedKeys/>
    </oratrust:Issuer>
  </oratrust:Issuers>
  <oratrust:Issuers>
    <oratrust:Issuer oratrust:name="www.oracle.com" oratrust:tokentype="jwt">
      <oratrust:TrustedKeys/>
    </oratrust:Issuer>
  </oratrust:Issuers>
  <oratrust:TokenAttributeRules/>
</oratrust:TokenIssuerTrust>
```
GET TrustDocument Method

Use the GET method to view configuration details for the trusted issuer document.

REST Request

GET /idaas/webservice/admin/v1/trustdocument

Parameters

The following table summarizes the POST request parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;documentName&quot;</td>
<td>Name of the document.</td>
<td>Query</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;Result&quot;</td>
<td>Details of the operation results.</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 view all token attribute rules by submitting a GET request on the REST resource using cURL.

curl -i -X GET -u username:password http://myhost:7001/idaas/platform/admin/v1/trustdocument?"documentName=myTrustDocument"

Example of Response Header

The following shows an example of the response header.

HTTP/1.1 200 OK

Example of Response Body

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

```
{
    "STATUS":"Succeeded",
    "Result":"List of token issuer trust documents in the Repository:
Details of the document matching your request:
Name         : myTrustDocument	Display Name : myTrustDocument	Status       : DOCUMENT_STATUS_COMMITED
List of trusted issuers for this type: None
List of Token Attribute Rules: None"
}
```
DELETE Trust Document Method

Use the Delete method to delete a trusted issuer document.

REST Request

DELETE /idaas/webservice/admin/v1/trustdocument

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>'displayName'</td>
<td>Display name for the document.</td>
<td>Query</td>
</tr>
<tr>
<td>'documentName'</td>
<td>Name of trusted issuer document.</td>
<td>Query</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>'ERROR_CODE'</td>
<td>If 'STATUS' is set to &quot;Failed&quot;, provides the error code.</td>
</tr>
<tr>
<td>'ERROR_MSG'</td>
<td>If 'STATUS' is set to &quot;Failed&quot;, provides the contents of the error message.</td>
</tr>
<tr>
<td>'Result'</td>
<td>Details of the operation results.</td>
</tr>
<tr>
<td>'STATUS'</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 SAML issuer trust document by submitting a DELETE request on the REST resource using cURL.

curl -i -X DELETE -u username:password
http://myhost:7001/idaas/webservice/admin/v1/trustdocument?"documentName=myTrustDocument&displayName=myTrustDocument"

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',
    'Result': 'Token Issuer Trust document named "myTrustDocument" deleted from the repository.'
}
The credential and keystore management REST API provides a powerful set of resources that you can use to manage web service security, including credential stores, keystores, and trust stores.

Before using the REST API, you need to understand how to access the REST resources and other important concepts. See "About the REST API" on page 1-1.

The following table summarizes the REST resource paths, alphabetically by resource path.

<table>
<thead>
<tr>
<th>REST Resource</th>
<th>Method</th>
<th>More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>/idaas/platform/admin/v1/credential</td>
<td>GET</td>
<td>GET Credential Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/credential</td>
<td>DELETE</td>
<td>DELETE Credential Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/credential</td>
<td>POST</td>
<td>POST Credential Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/credential</td>
<td>PUT</td>
<td>PUT Credential Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystore</td>
<td>GET</td>
<td>GET All Aliases Trusted Certificate JKS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystore/{alias}</td>
<td>GET</td>
<td>GET Specified Alias Trusted Certificate JKS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystore/{alias}</td>
<td>DELETE</td>
<td>DELETE Trusted Certificate JKS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystore/{alias}</td>
<td>POST</td>
<td>POST Specified Alias Trusted Certificate JKS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystore/pkcs7/{alias}</td>
<td>POST</td>
<td>GET Specified Alias Trusted Certificate JKS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice</td>
<td>DELETE</td>
<td>DELETE Keystore Service KSS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice</td>
<td>POST</td>
<td>POST New KSS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice</td>
<td>PUT</td>
<td>PUT Password Update KSS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice/alias/{stripeName}://{keystoreName}/{entryType}</td>
<td>GET</td>
<td>GET Alias KSS Keystore Method</td>
</tr>
<tr>
<td>REST Resource</td>
<td>Method</td>
<td>More Information</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>--------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice/certificates</td>
<td>GET</td>
<td>GET Trusted Certificate KSS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice/certificates</td>
<td>DELETE</td>
<td>DELETE Trusted Certificate KSS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice/certificates</td>
<td>POST</td>
<td>POST Trusted Certificate KSS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice/keystore</td>
<td>POST</td>
<td>POST Import KSS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice/secretkey</td>
<td>GET</td>
<td>GET Secret Key Properties KSS Keystore Method</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice/secretkey</td>
<td>POST</td>
<td>POST Secret Key KSS Keystore</td>
</tr>
<tr>
<td>/idaas/platform/admin/v1/keystoreservice/{stripeName}</td>
<td>GET</td>
<td>GET Stripe KSS Keystores Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trust/issuers</td>
<td>GET</td>
<td>GET All Trusted Issuer and Distinguished Name Lists Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trust/issuers/{documentName}</td>
<td>GET</td>
<td>GET Specified Document Trusted Issuer and Distinguished Name Lists Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trust/issuers</td>
<td>POST</td>
<td>POST Domain Trusted Issuer and Distinguished Name Lists Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trust/issuers/{documentName}</td>
<td>POST</td>
<td>POST Document Trusted Issuer and Distinguished Name Lists Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trust/token</td>
<td>GET</td>
<td>GET All Token Attribute Rules Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trust/token/{documentName}</td>
<td>GET</td>
<td>GET Specified Document Token Attribute Rules Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trust/token</td>
<td>POST</td>
<td>POST Token Attribute Rule Distinguished Name Method (Domain Context)</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trust/token/{documentName}</td>
<td>POST</td>
<td>POST Token Attribute Rule Distinguished Name Method (Document Context)</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trustdocument</td>
<td>GET</td>
<td>GET TrustDocument Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trustdocument</td>
<td>DELETE</td>
<td>DELETE Trust Document Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trustdocument</td>
<td>POST</td>
<td>POST TrustDocument Name Method</td>
</tr>
<tr>
<td>/idaas/webservice/admin/v1/trustdocument/import</td>
<td>POST</td>
<td>Import TrustDocument Name Configurations Method</td>
</tr>
</tbody>
</table>