Oracle® Fusion Middleware

REST API for Managing Credentials and Keystores with Oracle Web Services Manager

12*c* 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.

Oracle Fusion Middleware REST API for Managing Credentials and Keystores with Oracle Web Services Manager, 12*c* Release 12.2.1

E64994-01

Primary Author: Oracle Corporation

Contributing Author: Vinaye Misra

Contents

Preface

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:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New In This Guide

This section summarizes the new features and significant product changes for Oracle Web Services Manager in Oracle Fusion Middleware 12*c* (12.2.1).

New and Changed Features for 12*c* (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:

1 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:

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.

- *contextpath*—Context path for the REST resource. This value can be set to `platform` for resources that apply across the domain (for example, keystore and credential management resources), or `webservice` for resources that apply to a specific web service (for example, trust management resources).
- *resource*—Relative path that defines the REST resource. For more information, see "REST API Reference." To access the Web Application Definition Language (WADL) document, specify `application.wadl`.

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.

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

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 keystorePassword:cHdkMQ== -H
keyPassword:bXlQd2Qy
http://myhost:7001/idaas/platform/admin/v1/keystoreservice/certificates?"stripeNam
e=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:

HTTP Status Code	Description
200 OK	The request was successfully completed. A 200 status is returned for successful `GET` or `POST` method.
201 Created	The request has been fulfilled and resulted in a new resource being created. The response includes a Location header containing the canonical URI for the newly created resource. A 201 status is returned from a synchronous resource creation or an asynchronous resource creation that completed before the response was returned.

HTTP Status Code	Description
202 Accepted	The request has been accepted for processing, but the processing has not been completed. The request may or may not eventually be acted upon, as it may be disallowed at the time processing actually takes place. When specifying an asynchronous (`__detached=true`) resource creation (for example, when deploying an application), or update (for example, when redeploying an application), a 202 is returned if the operation is still in progress. If `__detached=false`, a 202 may be returned if the underlying operation does not complete in a reasonable amount of time. The response contains a Location header of a job resource that the client should poll to determine when the job has finished. Also, returns an entity that contains the current state of the job
400 Bad Request	The request could not be processed because it contains missing or invalid information (such as, a validation error on an input field, a missing required value, and so on).
401 Unauthorized	The request is not authorized. The authentication credentials included with this request are missing or invalid.
403 Forbidden	The user cannot be authenticated. The user does not have authorization to perform this request.
404 Not Found	The request includes a resource URI that does not exist.
405 Method Not Allowed	The HTTP verb specified in the request (`DELETE`, `GET`, `POST`, `PUT`) is not supported for this request URI.
406 Not Acceptable	The resource identified by this request is not capable of generating a representation corresponding to one of the media types in the Accept header of the request. For example, the client's Accept header request XML be returned, but the resource can only return JSON.
415 Not Acceptable	The client's ContentType header is not correct (for example, the client attempts to send the request in XML, but the resource can only accept JSON).
500 Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
503 Service Unavailable	The server is unable to handle the request due to temporary overloading or maintenance of the server. The Oracle WSM REST web application is not currently running.

2 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.

   ```
   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.

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

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

2. View all credentials in the credential store.

   ```
   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:

   ```
   {
       "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.

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

   ```
   {
       "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.

Note: For information about JKS keystore management, see "Configuring Keystores for Message Protection" in *Security and Administrator's Guide for Web Services*.

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":
      "MIIC7DCCAqqgAwIBAgIEalhBSjALBgcqhkjOOAQDBQAwSDEKMAgGA1UEBhMBeTEKMAgGA1UECB
      MB\neTEKMAgGA1UEBxMBeTEKMAgGA1UEChMBeTEKMAgGA1UECxMBeTEKMAgGA1UEAxMBeTAeFw0
      xNDA3\nMDMxMTAwMTZaFw0xNDEwMDExMTAwMTZaMEgxCjAIBgNVBAYTAXkxCjAIBgNVBAgTAXkx
      CjAIBgNV\nBAcTAXkxCjAIBgNVBAoTAXkxCjAIBgNVBAsTAXkxCjAIBgNVBAMTAXkwggG3MIIBL
      AYHKoZIzjgE\nATCCAR8CgYEA/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUf
      tZPY1Y+r/F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKY
      VDwT7g/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu/yIgMZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL
      8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu/o66oL5V0wLPQeCZ1FZV4661FlP5nEHEIGAt
      EkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qF
      ```

```
GQiaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7/s9JKgOBhAACgYBrvzkjozmv6t6T0GNJES1R3ypRsBs
8VLX2g3GotHd7Kht/TCj4HikelZDd\nuL0t96R5Q4A3srOgSIZ+0INRs1ER8y1Q37LyJNfyqYn5
KqLBlN9bhSYAfcuIpjwIXGVfLQGdByD7\ntr4PSvZQx18K6p68HUCh+jXQT9+7n3ZUIBzH5aMhM
B8wHQYDVR0OBBYEFPdMpcEBbYSCYMdJiE4r\ncQxf7Me4MAsGByqGSM44BAMFAAMvADAsAhQH/G
1ixrEaWAG3lGWafkHgXxnzhwIUW5eSctgmaQBj\nvKaY0E6fYJzcp5c="
}
```

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.

```
{
    "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.

```
{
    "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-----\
nMIIC7DCCAqqgAwIBAgIEalhBSjALBgcqhkjOOAQDBQAw
SDEKMAgGA1UEBhMBeTEKMAgGA1UECBMB\neTEKMAgGA1UEBxMBeTEKMAgGA1UEChMBeTEKMAgGA1UEC
x
MBeTEKMAgGA1UEAxMBeTAeFw0xNDA3\nMDMxMTAwMTZaFw0xNDEwMDExMTAwMTZaMEgxCjAIBgNVBAY
T
AXkxCjAIBgNVBAgTAXkxCjAIBgNV\nBAcTAXkxCjAIBgNVBAoTAXkxCjAIBgNVBAsTAXkxCjAIBgNVB
A
MTAXkwggG3MIIBLAYHKoZIzjgE\nATCCAR8CgYEA\/X9TgR11EilS30qcLuzk5\/YRt1I870QAwx4\/
g
LZRJmlFXUAiUftZPY1Y+r\/F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5\/oBHsQIsJPu6nX\/rf
G
G\/g7V+fGqKYVDwT7g\/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu\/yIgMZndFIAccCFQCXYFCPFSMLzL
K
SuYKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu\/o66oL5V0wLPQeCZ1FZV4661FlP5nE
H
EIGAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+q
F
GQiaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7\/s9JKgOBhAACgYBrvzkjozmv6t6T0GNJES1R3ypRsBs8VL
X
2g3GotHd7Kht\/TCj4HikelZDd\nuL0t96R5Q4A3srOgSIZ+0INRs1ER8y1Q37LyJNfyqYn5KqLBlN9
b
hSYAfcuIpjwIXGVfLQGdByD7\ntr4PSvZQx18K6p68HUCh+jXQT9+7n3ZUIBzH5aMhMB8wHQYDVR0OB
B
YEFPdMpcEBbYSCYMdJiE4r\ncQxf7Me4MAsGByqGSM44BAMFAAMvADAsAhQH\/G1ixrEaWAG3lGWafk
H
gXxnzhwIUW5eSctgmaQBj\nvKaY0E6fYJzcp5c=\n-----END CERTIFICATE-----",
    "SIGNATURE": "7JmdaAc+5T+spDFFo9gsRA==",
    "Extensions": "{subjectKeyIDExtension {oid = 2.5.29.14, critical = false,
value = f74ca5c1016d848260c749884e2b710c5fecc7b8}}"
}
```

For more information, see "GET Specified Alias Trusted Certificate JKS Keystore Method" on page 4-7.

4. Delete the trusted certificate from the JKS keystore.

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

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

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

For more information, see "DELETE Trusted Certificate JKS Keystore Method" on page 4-9.

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.

```
{
    "stripe" : "myStripe",
    "keystore" : "myKeystore",
    "pwd" : "mypwd",
    "permission" : "false"
}
```

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":
"MIIC7DCCAqqgAwIBAgIEalhBSjALBgcqhkjOOAQDBQAwSDEKMAgGA1UEBhMBeTEKMAgGA1UECB
MB\neTEKMAgGA1UEBxMBeTEKMAgGA1UEChMBeTEKMAgGA1UECxMBeTEKMAgGA1UEAxMBeTAeFw0
xNDA3\nMDMxMTAwMTZaFw0xNDEwMDExMTAwMTZaMEgxCjAIBgNVBAYTAXkxCjAIBgNVBAgTAXkx
CjAIBgNV\nBAcTAXkxCjAIBgNVBAoTAXkxCjAIBgNVBAsTAXkxCjAIBgNVBAMTAXkwggG3MIIBL
AYHKoZIzjgE\nATCCAR8CgYEA/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUf
tZPY1Y+r/F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKY
VDwT7g/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu/yIgMZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL
8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu/o66oL5V0wLPQeCZ1FZV4661FlP5nEHEIGAt
EkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qF
GQiaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7/s9JKgOBhAACgYBrvzkjozmv6t6T0GNJES1R3ypRsBs
8VLX2g3GotHd7Kht/TCj4HikelZDd\nuL0t96R5Q4A3srOgSIZ+0INRs1ER8y1Q37LyJNfyqYn5
KqLBlN9bhSYAfcuIpjwIXGVfLQGdByD7\ntr4PSvZQx18K6p68HUCh+jXQT9+7n3ZUIBzH5aMhM
B8wHQYDVR0OBBYEFPdMpcEBbYSCYMdJiE4r\ncQxf7Me4MAsGByqGSM44BAMFAAMvADAsAhQH/G
1ixrEaWAG3lGWafkHgXxnzhwIUW5eSctgmaQBj\nvKaY0E6fYJzcp5c=",
    "keystoreEntryType" : "TrustedCertificate",
    "keystoreName" : "myKeystore",
    "stripeName" : "myStripe",
    "keystorePassword" : "myPwd"
}
```

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?"stripe
Name=myStripe&keystoreName=myKeystore&keyAlias=myAlias&keystoreEntryType=Truste
dCertificate"
```

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-----
\nMIIC7DCCAqqgAwIBAgIEOosLyjALBgcqhkjOOAQDBQAwS
EKMAgGA1UEBhMBcjEKMAgGA1UECBMB\ncjEKMAgGA1UEBxMBcjEKMAgGA1UEChMBcjEKMAgGA1UECxM
cjEKMAgGA1UEAxMBUjAeFw0xNDA3\nMjUwOTQ1MTFaFw0xNDEwMjMwOTQ1MTFaMEgxCjAIBgNVBAYTA
IxCjAIBgNVBAgTAXIxCjAIBgNV\nBAcTAXIxCjAIBgNVBAoTAXIxCjAIBgNVBAsTAXIxCjAIBgNVBAM
AVIwggG3MIIBLAYHKoZIzjgE\nATCCAR8CgYEA\/X9TgR11EilS30qcLuzk5\/YRt1I870QAwx4\/gL
RJmlFXUAiUftZPY1Y+r\/F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5\/oBHsQIsJPu6nX\/rfGG
/g7V+fGqKYVDwT7g\/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu\/yIgMZndFIAccCFQCXYFCPFSMLzLKS
YKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu\/o66oL5V0wLPQeCZ1FZV4661FlP5nEHE
GAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qFG
iaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7\/s9JKgOBhAACgYAjhpZybXj6rlXDow8srnSFE9dZJJpCKaQV
ACagQogePV+xlqPClDOoiQJ\nuvuUGHerDrThC1\/Wq5Uj1+TnkSKTy0qYxmQoq56xALa47np9TKtqt
4Vy8eUUorakG4lrjNt\/EgR\nfO675n+qINkKXKpcxaCicupRCYPkPXlnT4mtyKMhMB8wHQYDVR0OBB
EFDKbmPa2Il6SylJRPTv8\nQ+4CqpEhMAsGByqGSM44BAMFAAMvADAsAhQbkmlaUG5QDR5mXUiYC74p
\/FBOwIUGx5lc5Y01ppo\nvK3UgL7M8E3eOfc=\n-----END CERTIFICATE-----",
    "SIGNATURE":FEZN2l4SPFEK5jt2QZRb5Q==",
    "Extensions":"{subjectKeyIDExtension {oid = 2.5.29.14 critical = false,
value = 329b98f6b6225e92ca52513d3bfc43ee02aa9121}}"
}
```

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

5. Delete the KSS keystore.

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

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
```

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

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.

```
curl -i -X POST -u username:password
http://myhost:7001/idaas/webservice/admin/v1/trustdocument?"documentName=myTrus
tDocument&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.

```
{
    "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.

```
{
    "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.

   ```
   {
       "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=myTrustD
   ocument"
   ```

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

   ```
   {
       "STATUS":"Succeeded",
       "Result":"List of token issuer trust documents in the Repository:\nDetails
   of the document matching your request:\nName        : myTrustDocument\tDisplay
   Name : myTrustDocument\tStatus       : DOCUMENT_STATUS_COMMITED \nList of
   trusted issuers for this type:\tNone\nList of Token Attribute Rules\tNone"
   }
   ```

   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=myTrus
   tDocument&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.

   ```
   {
       "STATUS": "Succeeded",
      "Result": "Token Issuer Trust document named "myTrustDocument" deleted from
   the repository."
   }
   ```

Part II

REST API Reference

Review details about the Oracle Fusion Middleware REST API for managing credentials and keystores.

Part II contains the following chapters:

3 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*.

Section	Method	Resource Path
POST Credential Method	`POST`	`/idaas/platform/admin/v1/credential`
GET Credential Method	`GET`	`/idaas/platform/admin/v1/credential`
PUT Credential Method	`PUT`	`/idaas/platform/admin/v1/credential`
DELETE Credential Method	`DELETE`	`/idaas/platform/admin/v1/credential`

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:

Attribute	Description	Required
`"credential"`	Password for the credential.	Yes
`"key"`	Name of the key.	Yes
`"map"`	Name of the map (folder).	Yes
`"username"`	Username for the credential.	Yes

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to create a 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.

```
{
    "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:

Attribute	Description
`"CSF_MAP_NAME"`	Name of the credential store map.
`"default"`	List of keys in the default credential map.
`"oracle.wsm.security"`	List of keys in the Oracle Web Services Manager (Oracle WSM) security credential map.

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.

```
{
    "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:

Attribute	Description	Required
`"credential"`	Updated password for the key in the keystore.	Yes
`"key"`	Name of the key that you want to modify. The key must exist.	Yes
`"map"`	Name of the map (folder) that you want to modify.	Yes
`"username"`	Username for the key in the keystore.	Yes

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to 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/patform/admin/v1/credential
```

Example of Request Body

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

```
{
    "username" : "username",
    "credential" : "myNewPwd",
    "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.

```
{
    "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.

Name	Description	Type
`"key"`	Name of the key for the credential that you want to delete.	Query
`"map"`	Name of the map (folder) for the credential that you want to delete.	Query

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to delete a 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.se
curity"
```

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"
}
```

4 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.

For information about JKS keystore management, see "Configuring Keystores for Message Protection" in *Security and Administrator's Guide for Web Services*.

Task	Method	Resource Path
GET All Aliases Trusted Certificate JKS Keystore Method	`GET`	`/idaas/platform/admin/v1/keystore`
POST Specified Alias Trusted Certificate JKS Keystore Method	`POST`	`/idaas/platform/admin/v1/keystore/{alias}`
POST PKCS#7 Trusted Certificate JKS Keystore Method	`POST`	`/idaas/platform/admin/v1/keystore/pkcs7/{alias}`
GET Specified Alias Trusted Certificate JKS Keystore Method	`GET`	`/idaas/platform/admin/v1/keystore/{alias}`
DELETE Trusted Certificate JKS Keystore Method	`DELETE`	`idaas/platform/admin/v1/keystore/{alias}`

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:

Attribute	Description
`"aliases"`	Comma-separated list of aliases.

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.

Name	Description	Type
`alias`	Alias of the trusted certificate to be imported. The alias will be created. The alias must not already exist in the JKS keystore; otherwise, the request will fail.	Path

Request Body

Media Types:	`application/json`

The request body contains the details of the import request:

Attribute	Description
`"certificate"`	Base64-encoded certificate.
`"component"`	Component to which the certificate is imported. This value must be set to `JKS`.

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to import a trusted 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/mytestkey
```

Example of Request Body

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

```
{
   "component":"JKS",
  "certificate":
"MIIC7DCCAqqgAwIBAgIEalhBSjALBgcqhkjOOAQDBQAwSDEKMAgGA1UEBhMBeTEKMAgGA1UECBMB\neTE
KMAgGA1UEBxMBeTEKMAgGA1UEChMBeTEKMAgGA1UECxMBeTEKMAgGA1UEAxMBeTAeFw0xNDA3\nMDMxMTA
wMTZaFw0xNDEwMDExMTAwMTZaMEgxCjAIBgNVBAYTAXkxCjAIBgNVBAgTAXkxCjAIBgNV\nBAcTAXkxCjA
IBgNVBAoTAXkxCjAIBgNVBAsTAXkxCjAIBgNVBAMTAXkwggG3MIIBLAYHKoZIzjgE\nATCCAR8CgYEA/X9
TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow\n9subVWzXgTuAHTRv8mZ
gt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu/yIgM
ZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu/o66oL5V0w
LPQeCZ1FZV4661FlP5nEHEIGAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE
1YvH7YnoBJDvMpPG+qFGQiaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7/s9JKgOBhAACgYBrvzkjozmv6t6T0GN
JES1R3ypRsBs8VLX2g3GotHd7Kht/TCj4HikelZDd\nuL0t96R5Q4A3srOgSIZ+0INRs1ER8y1Q37LyJNf
yqYn5KqLBlN9bhSYAfcuIpjwIXGVfLQGdByD7\ntr4PSvZQx18K6p68HUCh+jXQT9+7n3ZUIBzH5aMhMB8
wHQYDVR0OBBYEFPdMpcEBbYSCYMdJiE4r\ncQxf7Me4MAsGByqGSM44BAMFAAMvADAsAhQH/G1ixrEaWAG
3lGWafkHgXxnzhwIUW5eSctgmaQBj\nvKaY0E6fYJzcp5c="
}
```

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.

Name	Description	Type
`alias`	Alias of the private key for which the trusted PKCS#7 certificate will be imported. The alias must already in the JKS keystore.	Path

Request Body

Media Types:	`application/json`

The request body contains the details of the import request:

Attribute	Description
`"certificate"`	Base64-encoded certificate.
`"component"`	Component to which the certificate is imported. This value must be set to `JKS`.
`"keyPassword"`	Password for the private key.

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to 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.

```
{
   "component":"JKS",
   "certificate":
"MIIC7DCCAqqgAwIBAgIEalhBSjALBgcqhkjOOAQDBQAwSDEKMAgGA1UEBhMBeTEKMAgGA1UECBMB\neTE
KMAgGA1UEBxMBeTEKMAgGA1UEChMBeTEKMAgGA1UECxMBeTEKMAgGA1UEAxMBeTAeFw0xNDA3\nMDMxMTA
wMTZaFw0xNDEwMDExMTAwMTZaMEgxCjAIBgNVBAYTAXkxCjAIBgNVBAgTAXkxCjAIBgNV\nBAcTAXkxCjA
IBgNVBAoTAXkxCjAIBgNVBAsTAXkxCjAIBgNVBAMTAXkwggG3MIIBLAYHKoZIzjgE\nATCCAR8CgYEA/X9
TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow\n9subVWzXgTuAHTRv8mZ
gt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu/yIgM
ZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu/o66oL5V0w
LPQeCZ1FZV4661FlP5nEHEIGAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE
1YvH7YnoBJDvMpPG+qFGQiaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7/s9JKgOBhAACgYBrvzkjozmv6t6T0GN
JES1R3ypRsBs8VLX2g3GotHd7Kht/TCj4HikelZDd\nuL0t96R5Q4A3srOgSIZ+0INRs1ER8y1Q37LyJNf
yqYn5KqLBlN9bhSYAfcuIpjwIXGVfLQGdByD7\ntr4PSvZQx18K6p68HUCh+jXQT9+7n3ZUIBzH5aMhMB8
wHQYDVR0OBBYEFPdMpcEBbYSCYMdJiE4r\ncQxf7Me4MAsGByqGSM44BAMFAAMvADAsAhQH/G1ixrEaWAG
3lGWafkHgXxnzhwIUW5eSctgmaQBj\nvKaY0E6fYJzcp5c=",
    "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.

```
{
    "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.

Name	Description	Type
`alias`	Name of alias for which you want to view a trusted certificate.	Path

Response Body

Media Types:	`application/json`

The response body contains information about the certificate, including:

Attribute	Description
`"CONTENT"`	Contents of the Base64-encoded certificate.
`"Extensions"`	Optional extensions that are used to issue a certificate for a specific purpose. Each extension includes the following: ■ Object identifier (oid) that uniquely identifies it ■ Flag indicating whether the extension is critical ■ Value
`"ISSUER_DN"`	List of trusted distinguished names.
`"NOT_AFTER"`	Date the certificate expires.
`"NOT_BEFORE"`	Date the certificate is activated.
`"SERIAL_NO"`	Serial number of the JKS keystore.
`"SIGNATURE"`	Base64-encoded signature key.
`"SIGNING_ALGORITHM"`	Signing algorithm for the alias.
`"SUBJECT_DN"`	Subject distinguished names list.

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.

```
{
   "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-----\nMIIDeDCCAmCgAwI
BAgIEUclg5jANBgkqhkiG9w0BAQUFADB9MQswCQYDVQQGEwJ1czESMBAGA1UE\nCBMJdGVzdHN0YXRlM
REwDwYDVQQHEwh0ZXN0Y2l0eTEPMA0GA1UEChMGT3JhY2xlMSMwIQYDVQQL\nExpUZXN0a2V5IGZvciB
KS1MgTWJlYW4gdGVzdDERMA8GA1UEAxMId2VibG9naWMwIBcNMTMwNjI1\nMDkyMDM4WhgPMjA1MjExM
jcwOTIwMzhaMH0xCzAJBgNVBAYTAnVzMRIwEAYDVQQIEwl0ZXN0c3Rh\ndGUxETAPBgNVBAcTCHRlc3R
jaXR5MQ8wDQYDVQQKEwZPcmFjbGUxIzAhBgNVBAsTGlRlc3RrZXkg\nZm9yIEpLUyBNYmVhbiB0ZXN0M
REwDwYDVQQDEwh3ZWJsb2dpYzCCASIwDQYJKoZIhvcNAQEBBQAD\nggEPADCCAQoCggEBAJtmzlqcnU+
9d4OIor0FIOfcgpI\/EOflbkTicUjPr1AefYl8EDnl+U7hlDQ+\nPzrsndjAtFbcmxghGuw+P7\/ztIX
BBqIViLFW7wEBMdnGcO6Oc9swDca5vIofwNtor2hGI\/mIUPNx\nd9ExE2JOuqJmgr5RPyThv6mmxrVU
WJGCuHg4leQvSOOXxZFRWKHHWFv8lWwaqdY3haYHVD2DlNwS\nEPWqVAPZD6Kcv58l9ucHxAER5n5+wJ
PHH7kkGJL2gv2LIUMhwy3rlv2Fbhy7\/MTCeXYkUno5CXH9\n+nnAdWZ\/MzuVxXdzEZv72kmW\/oHnX
jSZtEdAwdQJAETz9Cxqwt9VtzsCAwEAATANBgkqhkiG9w0B\nAQUFAAOCAQEAG2\/kH7IlgFw3MAekgl
oOgwLgl87OVtlAySORxg2YNw9Z4GYQ2bRIL5lxp4kbMYic\nhB1SjR7aPXV0Jufw8EkBZMwDbLf053d6
oPEGWF7e6roCcHlY\/mBFd7BQFHW0vlBAZN9e1HkavWNE\n4k3qmjgct5BegMi9jhGrSws5aZ33qyrWc
r8zlZ3dhu52z4uGRG0UVeRnBemdPIk++6obiRErU3+v\nlI\/JYsQJmDrQwZlWGjznkXnQw5toJQuWFd
oE2TUPF1r3KTZiJ+TyVh64wtbnUVptxr1lFjtSfqPq\n0nzVlZlyXTi\/Rv7X+ODkRp29Hozs95c9HA9
3vnCYRaneNin7Kw==\n-----END CERTIFICATE-----",
   "SIGNATURE":"eAnH79sc8iMkLZRKWzh4vQ==",
   "Extensions":"{subjectKeyIDExtension {oid = 2.5.29.14 critical = false, value =
329b98f6b6225e92ca52513d3bfc43ee02aa9121}}"
}
```

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.

Name	Description	Type
`alias`	Alias of the trusted certificate entry to be deleted.	Path

Response Body

Media Types:	`application/json`

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

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

cURL Example

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

```
curl -i -X DELETE -u username:password
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"
}
```

5

Viewing and Managing Keystore Service Keystores

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 5–1 KSS Keystore Management REST Resources

Section	Method	Resource Path
POST New KSS Keystore Method	`POST`	`/idaas/platform/admin/v1/keystoreservice`
POST Import KSS Keystore Method	`POST`	`/idaas/platform/admin/v1/keystoreservice/keystore`
PUT Password Update KSS Keystore Method	`PUT`	`/idaas/platform/admin/v1/keystoreservice`
POST Trusted Certificate KSS Keystore Method	`POST`	`/idaas/platform/admin/v1/keystoreservice/certificates`
GET Stripe KSS Keystores Method	`GET`	`/idaas/platform/admin/v1/keystoreservice/{stripeName}`
GET Alias KSS Keystore Method	`GET`	`/idaas/platform/admin/v1/keystoreservice/alias/{stripeName}/{keystoreName}/{entryType}`
GET Trusted Certificate KSS Keystore Method	`GET`	`/idaas/platform/admin/v1/keystoreservice/certificates`
DELETE Trusted Certificate KSS Keystore Method	`DELETE`	`/idaas/platform/admin/v1/keystoreservice/certificates`
POST Secret Key KSS Keystore	`POST`	`/idaas/platform/admin/v1/keystoreservice/secretkey`
GET Secret Key Properties KSS Keystore Method	`GET`	`/idaas/platform/admin/v1/keystoreservice/secretkey`
DELETE Keystore Service KSS Keystore Method	`DELETE`	`/idaas/platform/admin/v1/keystoreservice`

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:

Attribute	Description
`"keystore"`	Name for the KSS keystore.
`"permission"`	Boolean value that specifies whether to create a permission-based keystore.
`"pwd"`	Password for the KSS keystore.
`"stripe"`	Name of the stripe to contain the KSS keystore.

Response Body

Media Types:	`application/json`

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

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

cURL Example

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

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

Example of Request Body

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

```
{
    "stripe" : "myStripe",
    "keystore" : "myKeystore",
    "pwd" : "myPwd",
    "permission" : "false"
}
```

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

Example of Response Header

The following shows an example of the response header.

```
HTTP/1.1 201 Created
```

Example of Response Body

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

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

POST Import KSS Keystore Method

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

REST Request

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

Request Body

Media Types:	`multipart/form-data`

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

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

Response Body

Media Types:	`application/json`

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

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

cURL Example

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

```
curl -i -X POST -u username:password -H Content-Type:multipart/form-data --form
"stripeName=myStripe" --form "keystoreFile=@clientkeystore" --form
"keystoreName=myKeystore" --form "keystorePassword=myPwd" --form
"keystoreType=JKS" --form "keyAliases=client" --form "keyPasswords=myPwd2" --form
"permission=false"
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.

```
{
    "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:

Attribute	Description
`"keystore"`	Name of the KSS keystore.
`"newpass"`	New password for the keystore.
`"oldpass"`	Old password for the keystore.
`"stripe"`	Name of the stripe.

Response Body

Media Types:	`application/json`

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

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

cURL Example

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

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

Example of Request Body

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

```
{
    "stripe" : "myStripe",
    "keystore" : "mykssstore",
    "oldpass" : "myPwd",
    "newpass" : "myNewPwd"
}
```

Example of Response Header

The following shows an example of the response header.

```
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:

Attribute	Description
`"keyAlias"`	Alias for the trusted certificate.
`"keystoreEntry"`	Base64-encoded certificate.
`"keystoreEntryType"`	Keystore entry type. Valid values include: `Certificate`, `TrustedCertificate`, or `SecretKey`.
`"keystoreName"`	Name of the KSS keystore.
`"keystorePassword"`	Password for the KSS keystore.
`"stripeName"`	Name of the stripe.

Response Body

Media Types:	`application/json`

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

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

cURL Example

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

```
curl -i -X POST -u username:password --data @importcertkss.json -H
Content-Type:application/json
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.

```
{
    "keyAlias" : "myAlias",
    "keystoreEntry":
"MIIC7DCCAqqgAwIBAgIEalhBSjALBgcqhkjOOAQDBQAwSDEKMAgGA1UEBhMBeTEKMAgGA1UECBMB\neTE
KMAgGA1UEBxMBeTEKMAgGA1UEChMBeTEKMAgGA1UECxMBeTEKMAgGA1UEAxMBeTAeFw0xNDA3\nMDMxMTA
wMTZaFw0xNDEwMDExMTAwMTZaMEgxCjAIBgNVBAYTAXkxCjAIBgNVBAgTAXkxCjAIBgNV\nBAcTAXkxCjA
IBgNVBAoTAXkxCjAIBgNVBAsTAXkxCjAIBgNVBAMTAXkwggG3MIIBLAYHKoZIzjgE\nATCCAR8CgYEA/X9
TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow\n9subVWzXgTuAHTRv8mZ
gt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu/yIgM
ZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu/o66oL5V0w
LPQeCZ1FZV4661FlP5nEHEIGAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE
1YvH7YnoBJDvMpPG+qFGQiaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7/s9JKgOBhAACgYBrvzkjozmv6t6T0GN
JES1R3ypRsBs8VLX2g3GotHd7Kht/TCj4HikelZDd\nuL0t96R5Q4A3srOgSIZ+0INRs1ER8y1Q37LyJNf
yqYn5KqLBlN9bhSYAfcuIpjwIXGVfLQGdByD7\ntr4PSvZQx18K6p68HUCh+jXQT9+7n3ZUIBzH5aMhMB8
wHQYDVR0OBBYEFPdMpcEBbYSCYMdJiE4r\ncQxf7Me4MAsGByqGSM44BAMFAAMvADAsAhQH/G1ixrEaWAG
3lGWafkHgXxnzhwIUW5eSctgmaQBj\nvKaY0E6fYJzcp5c=",
    "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.

```
{
    "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.

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

Response Body

Media Types:	`application/json`

The response body contains information about the certificate, including:

Attribute	Description
`"keystore n"`	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.

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.

Name	Description	Type
`entryType`	Keystore type. Valid values include `Certificate`, `TrustedCertificate`, or `SecretKey`.	Path
`keystoreName`	Name of the keystore.	Path
`stripeName`	Name of the stripe.	Path

Response Body

Media Types:	`application/json`

The response body contains information about the certificate, including:

Attribute	Description
`"keystore` *n*`"`	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.

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/alias/myStripe/myKeysto
re/TrustedCertificate
```

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":"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.

Name	Description	Type
`keyAlias`	Alias for trusted certificate.	Query
`keystoreEntryType`	Type of keystore entry. Valid values include `Certificate`, `TrustedCertificate`, or `CertificateChain`.	Query
`keystoreName`	Name of the keystore.	Query
`stripeName`	Name of the stripe.	Query

Response Body

Media Types:	`application/json`

The response body contains information about the certificate, including:

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

cURL Example

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

```
curl -i -X GET -u username:password -H keystorePassword:cHdkMQ==
http://myhost:7001/idaas/platform/admin/v1/keystoreservice/certificates?"stripeNam
e=myStripe&keystoreName=myKeystore&keyAlias=client&keystoreEntryType=Certificate"
```

Example of Response Header

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

```
HTTP/1.1 200 OK
```

Example of Response Body

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

```
{
    "SUBJECT_DN":"CN=y,OU=y,O=y,L=y,ST=y,C=y",
    "ISSUER_DN":"CN=y,OU=y,O=y,L=y,ST=y,C=y",
    "NOT_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-----
\nMIIC7DCCAqqgAwIBAgIEOosLyjALBgcqhkjOOAQDBQAwS
EKMAgGA1UEBhMBcjEKMAgGA1UECBMB\ncjEKMAgGA1UEBxMBcjEKMAgGA1UEChMBcjEKMAgGA1UECxM
cjEKMAgGA1UEAxMBUjAeFw0xNDA3\nMjUwOTQ1MTFaFw0xNDEwMjMwOTQ1MTFaMEgxCjAIBgNVBAYTA
IxCjAIBgNVBAgTAXIxCjAIBgNV\nBAcTAXIxCjAIBgNVBAoTAXIxCjAIBgNVBAsTAXIxCjAIBgNVBAM
AVIwggG3MIIBLAYHKoZIzjgE\nATCCAR8CgYEA\/X9TgR11EilS30qcLuzk5\/YRt1I870QAwx4\/gL
RJmlFXUAiUftZPY1Y+r\/F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5\/oBHsQIsJPu6nX\/rfGG
/g7V+fGqKYVDwT7g\/bTxR7DAjVU\nE1oWkTL2dfOuK2HXKu\/yIgMZndFIAccCFQCXYFCPFSMLzLKS
YKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu\/o66oL5V0wLPQeCZ1FZV4661FlP5nEHE
GAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qFG
iaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7\/s9JKgOBhAACgYAjhpZybXj6rlXDow8srnSFE9dZJJpCKaQV
ACagQogePV+xlqPClDOoiQJ\nuvuUGHerDrThC1\/Wq5Uj1+TnkSKTy0qYxmQoq56xALa47np9TKtqt
4Vy8eUUorakG4lrjNt\/EgR\nfO675n+qINkKXKpcxaCicupRCYPkPXlnT4mtyKMhMB8wHQYDVR0OBB
EFDKbmPa2Il6SylJRPTv8\nQ+4CqpEhMAsGByqGSM44BAMFAAMvADAsAhQbkmlaUG5QDR5mXUiYC74p
\/FBOwIUGx5lc5Y01ppo\nvK3UgL7M8E3eOfc=\n-----END CERTIFICATE-----",
    "SIGNATURE":FEZN2l4SPFEK5jt2QZRb5Q==",
    "Extensions":"{subjectKeyIDExtension {oid = 2.5.29.14 critical = false, value
= 329b98f6b6225e92ca52513d3bfc43ee02aa9121}}"
}
```

DELETE Trusted Certificate KSS Keystore Method

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

REST Request

```
DELETE /idaas/platform/admin/v1/keystoreservice/certificates
```

Parameters

The following table summarizes the DELETE request parameters.

Name	Description	Type
`keyAlias`	Alias for the certificate in the KSS keystore.	Query
`keystoreName`	Name of the keystore.	Query
`stripeName`	Name of stripe.	Query

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to 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?"stripeNam
e=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

Media Types:	`application/json`

The request body contains the details of the create request:

Attribute	Description
`"algorithm"`	Controls the cryptographic characteristics of the algorithms that are used when securing messages.
`"keyAlias"`	Alias for the secret key.
`"keyPassword"`	Password for the secret key.
`"keySize"`	Size measured in bits of the of the key used in cryptographic algorithm.
`"keystoreName"`	Name for the KSS keystore.
`"keystorePassword"`	Password for the KSS keystore.
`"stripeName"`	Name of the stripe.

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to create a 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.

```
{
    "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.

```
{
    "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.

Name	Description	Type
`keyAlias`	Alias of the secret key.	Query
`keystoreName`	Name of the keystore.	Query
`stripeName`	Name of the stripe.	Query

Response Body

Media Types:	`application/json`

The response body contains information about the certificate, including:

Attribute	Description
`"Property n"`	List of secret key properties, where *n* serves as an index that starts at `1` and is incremented by `1` for each additional property.

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=m
yStripe&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.

Name	Description	Type
`keystoreName`	Name of the keystore.	Query
`stripeName`	Name of the stripe.	Query

Response Body

Media Types:	`application/json`

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

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

cURL Example

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

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

6 Managing Token Issuer Trust Configurations

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*.

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

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.

Name	Description	Type
`"displayName"`	Display name for the document.	Query
`"documentName"`	Name of the document.	Query

Response Body

Media Types:	`application/json`

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

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

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=myTrustDo
cument&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

Media Types:	`application/json`

The request body contains the details of the add request:

Attribute	Description	Required
`"dn"`	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: `http://www.ietf.org/rfc/rfc2253.txt`	Yes
`"issuer"`	Groups information about a trusted issuer.	Yes
`"-name"`	Name of the trusted issuer. For example, `www.yourcompany.com`. The default value for the predefined SAML client policies is `www.oracle.com`.	Yes
`"jwt-trusted-dns"`	Groups information about JSON Web Token (JWT) trusted issuers.	No
`"saml-hok-trusted-dns"`	Groups information about SAML holder-of-key trusted issuers.	No
`"saml-sv-trusted-dns"`	Groups information about SAML sender vouches trusted issuers.	No
`"saml-trusted-dns"`	Groups the trusted issuers and DN lists.	Yes

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to create a 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.

```
{
    "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",
}
```

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.

Name	Description	Type
`documentName`	Name of trusted issuer document. For information about creating a trusted issuer document, see "POST TrustDocument Name Method" on page 6-2.	Query

Request Body

Media Types:	`application/json`

The request body contains the details of the add request:

Attribute	Description	Required
`"dn"`	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: `http://www.ietf.org/rfc/rfc2253.txt`	Yes
`"issuer"`	Groups information about a trusted issuer.	Yes
`"-name"`	Name of the trusted issuer. For example, `www.yourcompany.com`. The default value for the predefined SAML client policies is `www.oracle.com`.	Yes
`"jwt-trusted-dns"`	Groups information about JSON Web Token (JWT) trusted issuers.	No
`"saml-hok-trusted-dns"`	Groups information about SAML holder-of-key trusted issuers.	No
`"saml-sv-trusted-dns"`	Groups information about SAML sender vouches trusted issuers.	No
`"saml-trusted-dns"`	Groups the trusted issuers and DN lists.	Yes

Response Body

Media Types:	`application/json`

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

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

cURL Example

The following example shows how to create 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

Media Types:	`application/json`

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

Attribute	Description
`"dn"`	List of DN values to be added to the trusted issuer.
`"issuer"`	Groups information about a trusted issuer.
`"-name"`	Name of the trusted issuer.
`"jwt-trusted-dns"`	Groups information about JSON Web Token (JWT) trusted issuers.
`"saml-hok-trusted-dns"`	Groups information about SAML holder-of-key trusted issuers.
`"saml-sv-trusted-dns"`	Groups information about SAML sender vouches trusted issuers.
`"saml-trusted-dns"`	Groups the DN lists.

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.

```
{
    "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.

Name	Description	Type
`documentName`	Name of document for which you want to view issuer and DN lists.	Path

Response Body

Media Types:	`application/json`

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

Attribute	Description
`"dn"`	List of DN values to be added to the trusted issuer.
`"issuer"`	Groups information about a trusted issuer.
`"-name"`	Name of the trusted issuer.
`"jwt-trusted-dns"`	Groups information about JSON Web Token (JWT) trusted issuers.
`"saml-hok-trusted-dns"`	Groups information about SAML holder-of-key trusted issuers.
`"saml-sv-trusted-dns"`	Groups information about SAML sender vouches trusted issuers.
`"saml-trusted-dns"`	Groups the DN lists.

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

Media Types:	`application/json`

The request body contains the details of the add request:

Attribute	Description
`"attributes"`	Groups the constraints filter and mapping attributes for trusted users. **Note**: This attribute is not required on the client side.
`"-dn"`	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: `http://www.ietf.org/rfc/rfc2253.txt` 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 `https://messaging.us2.com/`, then the attribute rule applies to all service invocations with the service URL of the form `https://messaging.us2.com/`*`<path>`*
`"filter"`	Defines the constraint values for trusted users and attributes. **Note**: This attribute is not applicable on the client side.
`"mapping"`	Defines the mapping attributes for trusted users.
`"-name"`	Name of the attribute rule. **Note**: This attribute is not applicable on the client side.
`"name-id"`	Defines the users that are accepted for the trusted DN.
`"token-attribute-rule"`	Groups information about a single token attribute rule.
`"tokn-attribute-rules"`	Groups information about all token attribute rules.
`"user-attribute"`	Defines the user attribute that the trusted DN can assert. **Note**: This attribute is not applicable on the client side.
`"user-mapping-attribute"`	Defines the user mapping attribute that the trusted DN can assert.

Attribute	Description
"value"	Defines values for the constraint filter attribute. This value can be a full name or name pattern with a wildcard character (*), such as "yourTrusted*". Multiple values must be separated by a comma. **Note**: This attribute is not applicable on the client side.

Response Body

Media Types:	application/json

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

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

cURL Example

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

```
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.

```
{
    "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"
                            }
                        }
                    }
                ]
            }
        ]
    }
}
```

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.

```
{
    "token-attribute-rules":
    {
        "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.

```
{
    "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.

Name	Description	Type
`documentName`	Name of document for which you want to create a token attribute rule.	Path

Request Body

Media Types:	`application/json`

The request body contains the details of the add request:

Attribute	Description
`"attributes"`	Groups the constraints filter and mapping attributes for trusted users. **Note**: This attribute is not required on the client side.
`"-dn"`	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: `http://www.ietf.org/rfc/rfc2253.txt` 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 `https://messaging.us2.com/`, then the attribute rule applies to all service invocations with the service URL of the form `https://messaging.us2.com/`*`<path>`*
`"filter"`	Defines the constraint values for trusted users and attributes. **Note**: This attribute is not applicable on the client side.
`"mapping"`	Defines the mapping attributes for trusted users.
`"-name"`	Name of the attribute rule. **Note**: This attribute is not applicable on the client side.
`"name-id"`	Defines the users that are accepted for the trusted DN.
`"token-attribute-rule"`	Groups information about a single token attribute rule.
`"tokn-attribute-rules"`	Groups information about all token attribute rules.

Attribute	Description
`"user-attribute"`	Defines the user attribute that the trusted DN can assert. **Note**: This attribute is not applicable on the client side.
`"user-mapping-attribute"`	Defines the user mapping attribute that the trusted DN can assert.
`"value"`	Defines values for the constraint filter attribute. This value can be a full name or name pattern with a wildcard character (*), such as `"yourTrusted*"`. Multiple values must be separated by a comma. **Note**: This attribute is not applicable on the client side.

Response Body

Media Types:	`application/json`

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

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

cURL Example

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

```
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.

```
{
    "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"
                            }
                        }
                    }
                ]
            }
        ]
    }
}
```

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.

```
{
    "token-attribute-rules":
    {
        "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.

```
{
    "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:

Attribute	Description
`"attributes"`	Groups the constraints filter and mapping attributes for trusted users. **Note**: This attribute is not required on the client side.
`"-dn"`	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: `http://www.ietf.org/rfc/rfc2253.txt` On the client side, URL specified using the following format: `http(s)://`*`host`* or `http(s)://`*`host/root`*
`"filter"`	Defines the filter values for trusted users and attributes. You can enter a complete name or a name pattern with a wildcard character (*), such as `yourTrusted*`. If you specify multiple attribute filters, each filter should be separated by a comma.
`"mapping"`	Defines the mapping attributes for trusted users. **Note**: This attribute is not applicable on the client side.
`"-name"`	Name of the attribute rule. **Note**: This attribute is not applicable on the client side.
`"name-id"`	Defines the users that are accepted for the trusted DN.
`"token-attribute-rule"`	Groups information about a single token attribute rule.
`"tokn-attribute-rules"`	Groups information about all token attribute rules.
`"user-attribute"`	Defines the user attribute that the trusted DN can assert. **Note**: This attribute is not applicable on the client side.
`"user-mapping-attribute"`	Defines the user mapping attribute that the trusted DN can assert.
`"value"`	Defines values for the constraint filter attribute. This value can be a full name or name pattern with a wildcard character (*), such as `"yourTrusted*"`. Multiple values must be separated by a comma.

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

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.

Name	Description	Type
`documentName`	Name of document for which you want to view token attribute rules.	Path

Response Body

Media Types:	`application/json`

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

Attribute	Description
`"attributes"`	Groups the constraints filter and mapping attributes for trusted users. **Note**: This attribute is not required on the client side.
`"-dn"`	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: `http://www.ietf.org/rfc/rfc2253.txt` On the client side, URL specified using the following format: `http(s)://`*`host`* or `http(s)://`*`host/root`*
`"filter"`	Defines the filter values for trusted users and attributes. You can enter a complete name or a name pattern with a wildcard character (*), such as `yourTrusted*`. If you specify multiple attribute filters, each filter should be separated by a comma.
`"mapping"`	Defines the mapping attributes for trusted users. **Note**: This attribute is not applicable on the client side.
`"-name"`	Name of the attribute rule. **Note**: This attribute is not applicable on the client side.
`"name-id"`	Defines the users that are accepted for the trusted DN.
`"token-attribute-rule"`	Groups information about a single token attribute rule.
`"tokn-attribute-rules"`	Groups information about all token attribute rules.

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

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/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" ]
                    },
                    "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"
                    }
                },
            }
        ]
        "token-attribute-rule":
        [
            {
                "-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

Media Types:	`application/xml`

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 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

Media Types:	`application/json`

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

Element	Description
`"ERROR_CODE"`	If `"STATUS"` is set to `"Failed"`, provides the error code.
`"ERROR_MSG"`	If `"STATUS"` is set to `"Failed"`, provides the contents of the error message.
`"Result"`	Details of the operation results.
`"STATUS"`	Status of operation. For example, `"Succeeded"` or `"Failed"`.

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.

```
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 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: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>
```

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.

Name	Description	Type
`"documentName"`	Name of the document.	Query

Response Body

Media Types:	`application/json`

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

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

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=myTrustDocu
ment"
```

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:\nDetails of
the document matching your request:\nName         : myTrustDocument\tDisplay Name
: myTrustDocument\tStatus       : DOCUMENT_STATUS_COMMITED \nList of trusted
issuers for this type:\tNone\nList of Token Attribute Rules\tNone"
}
```

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.

Name	Description	Type
`"displayName"`	Display name for the document.	Query
`"documentName"`	Name of trusted issuer document.	Query

Response Body

Media Types:	`application/json`

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

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

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=myTrustDo
cument&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."
}
```

A Summary of REST APIs

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.

REST Resource	Method	More Information
`/idaas/platform/admin/v1/credential`	GET	GET Credential Method
`/idaas/platform/admin/v1/credential`	DELETE	DELETE Credential Method
`/idaas/platform/admin/v1/credential`	POST	POST Credential Method
`/idaas/platform/admin/v1/credential`	PUT	PUT Credential Method
`/idaas/platform/admin/v1/keystore`	GET	GET All Aliases Trusted Certificate JKS Keystore Method
`/idaas/platform/admin/v1/keystore/{alias}`	GET	GET Specified Alias Trusted Certificate JKS Keystore Method
`/idaas/platform/admin/v1/keystore/{alias}`	DELETE	DELETE Trusted Certificate JKS Keystore Method
`/idaas/platform/admin/v1/keystore/{alias}`	POST	POST Specified Alias Trusted Certificate JKS Keystore Method
`/idaas/platform/admin/v1/keystore/pkcs7/{alias}`	POST	GET Specified Alias Trusted Certificate JKS Keystore Method
`/idaas/platform/admin/v1/keystoreservice`	DELETE	DELETE Keystore Service KSS Keystore Method
`/idaas/platform/admin/v1/keystoreservice`	POST	POST New KSS Keystore Method
`/idaas/platform/admin/v1/keystoreservice`	PUT	PUT Password Update KSS Keystore Method
`/idaas/platform/admin/v1/keystoreservice/alias/{stripeName}/{keystoreName}/{entryType}`	GET	GET Alias KSS Keystore Method

REST Resource	Method	More Information
/idaas/platform/admin/v1/keystoreservice/certificates	GET	GET Trusted Certificate KSS Keystore Method
/idaas/platform/admin/v1/keystoreservice/certificates	DELETE	DELETE Trusted Certificate KSS Keystore Method
/idaas/platform/admin/v1/keystoreservice/certificates	POST	POST Trusted Certificate KSS Keystore Method
/idaas/platform/admin/v1/keystoreservice/keystore	POST	POST Import KSS Keystore Method
/idaas/platform/admin/v1/keystoreservice/secretkey	GET	GET Secret Key Properties KSS Keystore Method
/idaas/platform/admin/v1/keystoreservice/secretkey	POST	POST Secret Key KSS Keystore
/idaas/platform/admin/v1/keystoreservice/{stripeName}	GET	GET Stripe KSS Keystores Method
/idaas/webservice/admin/v1/trust/issuers	GET	GET All Trusted Issuer and Distinguished Name Lists Method
/idaas/webservice/admin/v1/trust/issuers/{documentName}	GET	GET Specified Document Trusted Issuer and Distinguished Name Lists Method
/idaas/webservice/admin/v1/trust/issuers	POST	POST Domain Trusted Issuers and Distinguished Name Lists Method
/idaas/webservice/admin/v1/trust/issuers/{documentName}	POST	POST Document Trusted Issuers and Distinguished Name Lists Method
/idaas/webservice/admin/v1/trust/token	GET	GET All Token Attribute Rules Method
/idaas/webservice/admin/v1/trust/token/{documentName}	GET	GET Specified Document Token Attribute Rules Method
/idaas/webservice/admin/v1/trust/token	POST	POST Token Attribute Rule Distinguished Name Method (Domain Context)
/idaas/webservice/admin/v1/trust/token/{documentName}	POST	POST Token Attribute Rule Distinguished Name Method (Document Context)
/idaas/webservice/admin/v1/trustdocument	GET	GET TrustDocument Method
/idaas/webservice/admin/v1/trustdocument	DELETE	DELETE Trust Document Method
/idaas/webservice/admin/v1/trustdocument	POST	POST TrustDocument Name Method
/idaas/webservice/admin/v1/trustdocument/import	POST	Import TrustDocument Name Configurations Method