

Oracle® Fusion Middleware

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

11g Release 1 (11.1.1.7)

E65317-01

July 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, 11g Release 1 (11.1.1.7)

E65317-01

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface	v
----------------------	---

Part I Get Started

1 About the REST API

1.1	Introducing the REST API	1-1
1.2	URL Structure	1-2
1.3	Supported REST Methods	1-2
1.4	Authentication	1-2
1.5	Status Codes	1-2
1.6	cURL Access	1-3

2 Use Cases

2.1	Managing the Credential Store Framework Using the REST API	2-1
2.2	Managing JKS Keystores Using the REST API	2-3
2.3	Managing KSS Keystores Using the REST API	2-5

Part II REST API Reference

3 Credential Store Management

	Create a New Credential in the Credential Store	3-2
	View All Credentials in the Credential Store	3-4
	Update a Credential in the Credential Store	3-6
	Delete a Credential from the Credential Store	3-8

4 JKS Keystore Management

	View All Aliases in the JKS Keystore	4-2
	Import a Trusted Certificate into the JKS Keystore	4-3
	Import a Trusted PKCS#7 Certificate into the JKS Keystore	4-5
	View a Trusted Certificate in the JKS Keystore	4-7
	Delete a Trusted Certificate from the JKS Keystore	4-9

5 KSS Keystore Management

Create a KSS Keystore	5-2
Import a KSS Keystore	5-4
Update the Password for a KSS Keystore	5-6
Import a Trusted Certificate into a KSS Keystore	5-8
View All KSS Keystores for a Stripe.....	5-10
View the Alias for the KSS Keystore	5-11
View a Trusted Certificate in the KSS Keystore	5-13
Delete a Certificate from a KSS Keystore	5-15
Create a Secret Key for a KSS Keystore	5-17
View Secret Key Properties for a KSS Keystore	5-19
Delete a KSS Keystore	5-21

6 Token Issuer Trust Management

Create Trusted Issuers and DN Lists	6-2
View a Trusted Issuer and DN Lists	6-5
Create a Token Attribute Rule for a Trusted DN	6-7
View All Token Attribute Rules	6-11

A Summary of REST APIs, Alphabetical by Resource Path

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.
<i>italic</i>	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.

Part I

Get Started

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

Part I contains the following chapters:

- [Chapter 1, "About the REST API"](#)
- [Chapter 2, "Use Cases"](#)

About the REST API

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

Topics:

- [Introducing the REST API](#)
- [URL Structure](#)
- [Supported REST Methods](#)
- [Authentication](#)
- [Status Codes](#)
- [cURL Access](#)

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 client applications such as:

- Web browsers
- cURL
- GNU Wget
- Postman

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

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 services (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 Supported REST Methods

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 Authentication

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-encoded keystore and key passwords, respectively, in custom headers. For example:

```
curl -i -X DELETE -u weblogic:welcome1 -H keystorePassword:cHdkMQ== -H
keyPassword:bx1Qd2Qy
http://myhost:7001/idaas/platform/admin/v1/keystoreservice/certificates?stripeName=
myStripe&keystoreName=myKeystore&keyAlias=myAlias"
```

1.5 Status Codes

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

HTTP Status Code	Description
200 OK	The request was successfully completed. A 200 status is returned for successful GET or POST method.
201 Created	The request has been fulfilled and resulted in a new resource being created. The response includes a Location header containing the canonical URI for the newly created resource. A 201 status is returned from a synchronous resource creation or an asynchronous resource creation that completed before the response was returned.
202 Accepted	The request has been accepted for processing, but the processing has not been completed. The request may or may not eventually be acted upon, as it may be disallowed at the time processing actually takes place. When specifying an asynchronous (<code>__detached=true</code>) 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 <code>__detached=false</code> , 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.

1.6 cURL Access

The examples within this document use cURL to demonstrate how to access the Oracle Java Cloud Service REST resources.

In the examples, one or more of the following options is used to direct the execution of cURL

cURL Option	Description
<code>--data @file.json</code>	Identifies the request document, in JSON format, on the local machine
<code>--form "name=value"</code>	Identifies form data.

cURL Option	Description
-H	Header that defines one or both of the following: <ul style="list-style-type: none">■ Content type of the request document■ Base64-encrypted keystore or key password.
-i	Displays response header information.
-u username:password	Specifies the user name and password for the WebLogic administrator for authentication.
-X	Indicates the type of request (for example, GET, POST, and so on). For more information about the supported REST methods, see " Supported REST Methods " on page 1-2.

For example:

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

For information about downloading and using cURL, see <http://curl.haxx.se>.

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](#)

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 "[Create a New Credential in the Credential Store](#)" 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
```

For more information about the cURL command-line options, see "[cURL Access](#)" on page 1-3.

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

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

For more information, see "[Create a New Credential in the Credential Store](#)" 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
```

For more information about the cURL command-line options, see "[cURL Access](#)" on page 1-3.

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",
    "owstest.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 "[View All Credentials in the Credential Store](#)" 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 a Credential from the Credential Store](#)" on page 3-8.

For more information about the cURL command-line options, see "[cURL Access](#)" on page 1-3.

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
```

For more information about the cURL command-line options, see "[cURL Access](#)" on page 1-3.

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 "[View All Aliases in the JKS Keystore](#)" 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.


```

"CONTENT": "-----BEGIN CERTIFICATE-----\n
nMIIC7DCCAgqgAwIBAgIEalhBSjALBgcqhkJ00AQDBQAw\n
SDEKMAgGA1UEBhMBeTEKMAgGA1UECBMB\neTEKMAgGA1UEBxMBeTEKMAgGA1UEChMBeTEKMAgGA1UEC\n
x\n
MBeTEKMAgGA1UEAxMBeTAeFw0xNDA3\nnMDMxMTAwMTZaFw0xNDEwMDEwMTAwMTZaMEgxCjAIBgNVBAY\n
T\n
AXkxCjAIBgNVBAGTAXkxCjAIBgNV\nnBACtAXkxCjAIBgNVBAoTAXkxCjAIBgNVBAsTAXkxCjAIBgNVB\n
A\n
MTAXkwggG3MIIBLAYHKoZiZjgE\nnATCCAR8CgYEA\n/X9TgR11Ei1S30qcLuzk5\n/YRt1I870QAwX4\n/\n
g\n
LZRJmlFXUaiUftZPY1Y+r\n/F9bow\nn9subVWzXgTuAHTRv8mZgt2uZUKWkn5\n/oBHsQIsJPu6nX\n/rf\n
G\n
G\n/g7V+fGqKYVDwT7g\n/bTxR7DAjVU\nnE1oWkTL2dfOuK2HXKu\n/yIgmZndFIAccCFQCXYFCPFSMLzL\n
K\n
SuYKi64QL8Fgc9QKBgQD34aCF1ps9\nn3su8q1w2uFe5eZSvu\n/o66oL5V0wLPQeCZ1FZV4661F1P5nE\n
H\n
EIGATekWcSPoTCgWE7fPCTKMyKbh\nnPBZ6i1R8jsjgo64eK70mdZFuo38L+ie1YvH7YnoBJDvMpg+q\n
F\n
GQiaid3+Fa5Z8GkotmXoB7VSVk\nnAUw7\n/s9JKgOBhAACgYBrvzkjozmV6t6T0GNJES1R3ypRsBs8VL\n
X\n
2g3GotHd7Kht\n/TCj4HikelzDd\nnuL0t96R5Q4A3srOgSIZ+0INRs1ER8y1Q37LyJNfyqYn5KqLB1N9\n
b\n
hSYAfcuIppjwIXGvFLQGdByD7\nntr4PSvZQx18K6p68HUCH+jXQT9+7n3ZUIBzH5aMhMB8wHQYDVR0OB\n
B\n
YEFpDmpeEBbYSCYmJiE4r\nncQxf7Me4MAsGByqGSM44BAMFAAMvADAsAhQH\n/G1ixrEaWAG1lGwafk\n
H\n
gXxnzhwIUW5eSctgmaQBJ\nnvKaY0E6fYJzcp5c=\nn-----END CERTIFICATE-----",
  "SIGNATURE": "7JmdaAc+5T+spDFFo9gsRA==",
  "Extensions": "{subjectKeyIDExtension {oid = 2.5.29.14, critical = false,
value = f74ca5c1016d848260c749884e2b710c5fecc7b8}}"}
}

```

For more information, see ["View a Trusted Certificate in the JKS Keystore"](#) 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

```

For more information about the cURL command-line options, see ["cURL Access"](#) on page 1-3.

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

```

{
  "STATUS": "Succeeded"
}

```

For more information, see ["Delete a Trusted Certificate from the JKS Keystore"](#) 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 "[Create a KSS Keystore](#)" 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
```

For more information about the cURL command-line options, see "[cURL Access](#)" on page 1-3.

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

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

For more information, see "[Create a KSS Keystore](#)" 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
```

For more information about the cURL command-line options, see "[cURL Access](#)" on page 1-3.

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

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

}

For more information, see ["View All KSS Keystores for a Stripe"](#) 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":
  "MIIC7DCCAqggAwIBAgIEalhbSjAlBgcqhkJ0OAQDBQAwSDEKMAgGA1UEBhMBEgGA1UECB
  MB\neTEKMAgGA1UEBxMBEgTEKMAgGA1UEChMBEgTEKMAgGA1UECxBEgTEKMAgGA1UEA
  xNDA3\NMDMxMTAwMTZaFw0xNDEwMDEwMTZaMEgxCjAIBgNVBAYTAxkxCjAIBgNVBA
  GTAxkxCjAIBgNV\nBACkxkxCjAIBgNVBAoTAxkxCjAIBgNVBAsTAxkxCjAIBgNVBA
  MTAxkxwG3MIIBL
  AYHKOZIZjgE\nATCCAR8CgYEA/X9Tgr11EilS30qcLuzk5/YRt1I870QAwX4/gLZR
  JmlFXUAiUf
  tZPY1Y+r/F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHSQIsJPu6nX/rfGG
  /g7V+fgqKY
  VDwT7g/bTxR7DAjVU\nEloWkTL2dfOuK2HXKu/yIgmZndFIaccCFQCYFCPFSMLzL
  KSuYKi64QL
  8Fgc9QKbgQD34aCF1ps9\n3su8qlw2uFe5eZsvu/o66oL5V0wLPQeCZ1FZV4661
  F1P5nEHEIGAT
  EkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK70mdZFuo38L+iE1YvH7Yno
  BJDvMppG+qF
  GQiaid3+Fa5Z8GkotmXoB7VSVk\nAUw7/s9JKgOBhAACgYBrvzkjozmV6t6T0GN
  JES1R3ypRsBs
  8VLX2g3GotHd7Kht/TCj4HikelZDd\nuL0t96R5Q4A3srOgSIZ+0INRs1ER8y1Q
  37LyJNfyqYn5
  KqLBlN9bhSYAfcuIppjwIXGVfLQGdByD7\ntr4PSvZQx18K6p68HUCH+jXQT9+7
  n3ZUIBzH5aMhM
  B8wHQYDVR0OBByEFpMpcEBbYSCYMDJiE4r\ncQxf7Me4MAsGByqGSM44BAMFAAM
  vADAsAhQH/G
  1ixrEaWAG3lGWafkHgXxnzhwIUW5eSctgmaQBj\nvKaY0E6fYJzcp5c=",
  "keystoreEntryType" : "TrustedCertificate",
  "keystoreName" : "myKeystore",
  "stripeName" : "myStripe",
  "keystorePassword" : "myPwd"
}
```

For more information about the request attributes, see ["Import a Trusted Certificate into a KSS Keystore"](#) 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
```

For more information about the cURL command-line options, see ["cURL Access"](#) on page 1-3.

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 ["Import a Trusted Certificate into a KSS Keystore"](#) on page 5-8.

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

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

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

For more information about the cURL command-line options, see ["cURL Access"](#) on page 1-3.

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-----
\nMIIC7DCCAqggAwIBAgIEOosLyjALBgqhkJ00AQDBQAwS
EKMAgGA1UEBhMBCjEKMAgGA1UECBMB\ncjEKMAgGA1UEBxMBCjEKMAgGA1UEChMBCjEKMAgGA1UECXM
cjEKMAgGA1UEAxMBUjAeFw0xNDA3\nMjUwOTQ1MTFhFw0xNDEwMjUwOTQ1MTFhMEgxCjAIBgNVBAYTA
IxMjUwOTQ1MTFhFw0xNDEwMjUwOTQ1MTFhMEgxCjAIBgNVBAsTAXIxCjAIBgNVBAM
AVIwggG3MIIBLAYHKOZIZjgE\nATCCAR8CgYEA\X9TgR11EilS30qcLuzk5\YRt1I870QAwX4\gLR
Um1FXUAiUftZPY1Y+r\F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5\oBhsQIsJPu6nX\rfGG
/g7V+fGqKYVDwT7g\bTxR7DAjVU\nE1oWkTL2df0uK2HXKu\yIgmZndFIaccCFQCXYFCPFSSMLzLKS
Yki64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu\o66oL5V0wLPQeCz1FZV4661F1P5nEHE
GAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpg+qFG
iaid3+Fa5Z8GkotmXoB7VSVk\nAUw7\s9JKgOBhAACgYAjhpybXj6rlXDow8srnSFE9dzJpCKaQV
ACagQogePV+xlqPClDooiQJ\nuvuUGHerDrThC1\Wq5Uj1+TnkSKTy0qYxmQoq56xALa47np9TKtqt
4Vy8eUUorakG4lrjNt\EgR\nf0675n+qINkKXKpcxaCicupRCYPkPXLnt4mtyKMhMB8wHQYDVR0OBB
EFDkmbPa2I16SylJRPTv8\nQ+4CqpEhMAsGByqGSM44BAMFAAMvADAsAhQbkmlaUG5QDR5mXUiy7C4p
\FBOWIUGx51c5Y01ppo\nvK3Ug7M8E3e0fc=\n-----END CERTIFICATE-----",
  "SIGNATURE" : "FEZN214SPFEK5jt2QZrb5Q==",
  "Extensions" : "{subjectKeyIDExtension {oid = 2.5.29.14 critical = false,
value = 329b98f6b6225e92ca52513d3bfc43ee02aa9121}}"}
}
```

For more information, see ["View a Trusted Certificate in the KSS Keystore"](#) on page 5-13.

- 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"
```

For more information about the cURL command-line options, see ["cURL Access"](#) on page 1-3.

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 a KSS Keystore"](#) on page 5-21.

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

HTTP/1.1 204 No Content

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:

- [Chapter 3, "Credential Store Management"](#)
- [Chapter 4, "JKS Keystore Management"](#)
- [Chapter 5, "KSS Keystore Management"](#)
- [Chapter 6, "Token Issuer Trust Management"](#)
- [Appendix A, "Summary of REST APIs, Alphabetical by Resource Path"](#)

Credential Store Management

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
Create a New Credential in the Credential Store	POST	/idaas/platform/admin/v1/credential
View All Credentials in the Credential Store	GET	/idaas/platform/admin/v1/credential
Update a Credential in the Credential Store	PUT	/idaas/platform/admin/v1/credential
Delete a Credential from the Credential Store	DELETE	/idaas/platform/admin/v1/credential

Create a New Credential in the Credential Store

Creates 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 ["Status Codes."](#)

```
HTTP/1.1 200 OK
```

Example of Response Body

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

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

View All Credentials in the Credential Store

Returns 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 ["Status Codes."](#)

```
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",  
    "owstest.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"  
  ]  
}
```

Update a Credential in the Credential Store

Updates 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 ["Status Codes."](#)

```
HTTP/1.1 200 OK
```

Example of Response Body

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

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

Delete a Credential from the Credential Store

Deletes 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 ["Status Codes."](#)

```
HTTP/1.1 200 OK
```

Example of Response Body

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

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


}

JKS Keystore Management

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
View All Aliases in the JKS Keystore	GET	/idaas/platform/admin/v1/keystore
Import a Trusted Certificate into the JKS Keystore	POST	/idaas/platform/admin/v1/keystore/{alias}
Import a Trusted PKCS#7 Certificate into the JKS Keystore	POST	/idaas/platform/admin/v1/keystore/pkcs7/{alias}
View a Trusted Certificate in the JKS Keystore	GET	/idaas/platform/admin/v1/keystore/{alias}
Delete a Trusted Certificate from the JKS Keystore	DELETE	idaas/platform/admin/v1/keystore/{alias}

View All Aliases in the JKS Keystore

Returns 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 ["Status Codes."](#)

```
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"  
}
```

Import a Trusted Certificate into the JKS Keystore

Imports 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. For more information, see ["cURL Access"](#) on page 1-3.

Import a Trusted PKCS#7 Certificate into the JKS Keystore

Imports 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. For more information, see ["cURL Access"](#) on page 1-3.

View a Trusted Certificate in the JKS Keystore

Returns 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: <ul style="list-style-type: none"> 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 "[Status Codes](#)."

```
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
BAgIEUc1g5jANBgkqhkiG9w0BAQUFADB9MQswCQYDVQQGEwJ1czESMBAGA1UE\nnCBMJdGVzdHN0YXR1M
REwDwYDVQQHEwh0ZXN0Y210eTEPMA0GA1UEChMGT3JhY2x1MSMwIQYDVQQQL\nExpUZXR0a2V5IGZvcib
KS1MgTWJlYW4gdGVzdDERMA8GA1UEAxMI d2VibG9naWwMwIBcNMTMwNjI1\nnMDkyMDM0WWhgPMjA1MjExM
jcwOTIwMzhaMH0xCzAJBgNVBAYTAnVzMRlWEAYDVQQIEw10ZXN0c3Rh\nndGUxETAPBgNVBACtCHRlc3R
jaXR5MQ8wDQYDVQQKEwZPcmFjbGUxIzAhBgNVBAsTGlRlc3RrZXkg\nnZm9yIEpLUyBNYmVhbiB0ZXN0M
REwDwYDVQQDEwh3ZWJsb2dpYzCCASlWdQYJKoZIhvcNAQEBBQAD\nnggEPADCCAQoCggEBAJtmzlcuU+
9d40Ior0FIOfcgPI\ /EOflbkTicUjPr1AefYl8EDnl+U7h1DQ+\n pZrsndjAtFbcmxghGuw+P7\ /ztIX
BBqIViLFW7wEBMdnGcO6Oc9swDca5vIofwNtor2hGI\ /mIUPNx\nd9ExE2J0uqJmgr5RPyThv6mmxrVU
WJGCUHg4leQvSOOXzZFRWKHHWFv81WwaqdY3haYHVD2D1NwS\nnEPWqVAPZD6Kcv5819ucHxAER5n5+wJ
PHH7kkGJL2gv2LIUMhwy3rlv2Fbhy7\ /MTCeXYkUno5CXH9\nn+nnAdWZ\ /MzuVxXdzEZv72kmW\ /oHnX
jSztEdAwdQJAETz9Cqxwt9VtzcAwEAATANBgkqhkiG9w0B\nnAQUFAAOCAQEAG2\ /kH7IlgFw3MAekgl
oOgwLgl870Vt1AySORxg2YNw9Z4GYQ2bRIL5lxp4kbMYic\nnhB1sJR7aPXV0Jufw8EkBZMwDbLf053d6
oPEGWF7e6roCch1Y\ /mBFd7BQFHW0v1BAZN9e1HkavWNE\nn4k3qmjgct5BegMi9jhGrSws5aZ33qyrWc
r8z1Z3dhu52z4uGRG0UveRnBemdPIk++6obiRErU3+v\nn1I\ /JYsQJmDrQwZ1WGjznkXnQw5toJQuWfd
oE2TUPF1r3KTziJ+TyVh64wtbnUVptxr11fjtSfqPq\nn0nzV1Z1yXTi\ /Rv7X+ODkRp29Hozs95c9HA9
3vnCYRaneNin7Kw==\n-----END CERTIFICATE-----",
  "SIGNATURE": "eAnH79sc8imkLZRKwzh4vQ==",
  "Extensions": "{subjectKeyIDExtension {oid = 2.5.29.14 critical = false, value =
329b98f6b6225e92ca52513d3bfc43ee02aa9121}}"}
}
```

Delete a Trusted Certificate from the JKS Keystore

Deletes 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 ["Status Codes."](#)

```
HTTP/1.1 200 OK
```

Example of Response Body

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

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


KSS Keystore Management

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
Create a KSS Keystore	POST	/idaas/platform/admin/v1/keystoreservice
Import a KSS Keystore	POST	/idaas/platform/admin/v1/keystoreservice/keystore
Update the Password for a KSS Keystore	PUT	/idaas/platform/admin/v1/keystoreservice
Import a Trusted Certificate into a KSS Keystore	POST	/idaas/platform/admin/v1/keystoreservice/certificates
View All KSS Keystores for a Stripe	GET	/idaas/platform/admin/v1/keystoreservice/{stripeName}
View the Alias for the KSS Keystore	GET	/idaas/platform/admin/v1/keystoreservice/alias/{stripeName}/{keystoreName}/{entryType}
View a Trusted Certificate in the KSS Keystore	GET	/idaas/platform/admin/v1/keystoreservice/certificates
Delete a Certificate from a KSS Keystore	DELETE	/idaas/platform/admin/v1/keystoreservice/certificates
Create a Secret Key for a KSS Keystore	POST	/idaas/platform/admin/v1/keystoreservice/secretkey
View Secret Key Properties for a KSS Keystore	GET	/idaas/platform/admin/v1/keystoreservice/secretkey
Delete a KSS Keystore	DELETE	/idaas/platform/admin/v1/keystoreservice

Create a KSS Keystore

Creates 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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" : "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"  
}
```

Import a KSS Keystore

Imports a Keystore Service (KSS) keystore from a KSS 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 <code>keystoreFile</code> .
"keyPasswords"	Comma-separated list of passwords for the keys to be imported from the <code>keystoreFile</code> .
"keystoreFile"	Name of a valid local KSS keystore file
"keystoreName"	Name for the KSS 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 <code>JKS</code> .
"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 <i>n</i> "	List of keystores in the stripe, where <i>n</i> 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. For more information, see ["cURL Access"](#) on

page 1-3.

```
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"
}
```

Update the Password for a KSS Keystore

Updates 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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"  
}
```

Import a Trusted Certificate into a KSS Keystore

Imports 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. For more information, see ["cURL Access"](#) on page 1-3.

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


View All KSS Keystores for a Stripe

Returns 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 <i>n</i> "	List of keystores in the stripe, where <i>n</i> 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. For more information, see "[cURL Access](#)" on page 1-3.

```
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 "[Status Codes](#)."

```
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"  
}
```

View the Alias for the KSS Keystore

Returns 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 <i>n</i> "	List of keystore aliases in the stripe where <i>n</i> 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. For more information, see ["cURL Access"](#) on page 1-3.

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

Example of Response Header

The following shows an example of the response header. For more about the HTTP status codes, see ["Status Codes."](#)

```
HTTP/1.1 200 OK
```

Example of Response Body

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

```
{
```

```
"keystore 1": "myAlias",  
}
```


View a Trusted Certificate in the KSS Keystore

Returns a 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: <ul style="list-style-type: none"> 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. For more information, see ["cURL](#)

[Access](#)" on page 1-3.

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

Example of Response Header

The following shows an example of the response header. For more about the HTTP status codes, see ["Status Codes."](#)

```
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-----
\nMIIC7DCCAqggAwIBAgIEOosLyjALBgcqhkj0OAQDBQAwS
EKMAgGA1UEBhMbcjEKMAgGA1UECBMB\ncjEKMAgGA1UEBxMbcjEKMAgGA1UEChMbcjEKMAgGA1UECXM
cjEKMAgGA1UEAxMBUjAeFw0xNDA3\nMjUwOTQ1MTFaFw0xNDEwMjUwOTQ1MTFaMEgxCjAIBGNVBAYTA
IxCjAIBGNVBAGTAXIxCjAIBGNV\nnBACtAXIxCjAIBGNVBAoTAXIxCjAIBGNVBAsTAXIxCjAIBGNVBAM
AVIwggG3MIIBLAYHKOziZjgE\nATCCAR8CgYEA\X9Tgr11EilS30qcLuzk5\YRt1I870QAwx4\g/L
RjmlFXUaiUftZPY1Y+r\F9bow\n9subVWzXgTuAHTRv8mZgt2uZUKWkn5\oBhsQIsJPu6nX\rfGG
/g7V+fGqKYVDwT7g\bTxR7DAjvU\nE1oWkTL2dfOuK2HXKu\yIgmZndFIaccFQCXYFCPPSMLzLKS
YKi64QL8Fgc9QKBgQD34aCF1ps9\n3su8q1w2uFe5eZSvu\o66oL5V0wLPQeCZ1FZV4661F1P5nEHE
GAtEkWcSPoTCgWE7fPCTKMyKbh\nPBZ6i1R8jsjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qFG
iaiD3+Fa5Z8GkotmXoB7VSVk\nAUw7\9sJKgOBhAACgYAjhpZybXj6r1XDow8srnSFE9dzJJpCKaQV
ACagQogePV+xlqPClDOoiQJ\nnuvuUGHerDrThC1\Wq5Uj1+TnkSKTy0qYxmQoq56xALa47np9TKtqt
4Vy8eUorakG4lrjNt\EgR\nf0675n+qINKXXKpcxaCicupRCYPkPXlnT4mtYKMhMB8wHQYDVR0OBB
EFDKbmPa2I16SylJRPTv8\nnQ+4CqpEhMasGByqGSM44BAMFAAMvADAsAhQbkmlaUG5QDR5mXUiYC74p
\FBOWIUGx51c5Y01ppo\nnvK3Ug7M8E3eOfc=\n-----END CERTIFICATE-----",
  "SIGNATURE": "FEZN214SPFEK5jt2QZRb5Q==",
  "Extensions": "{subjectKeyIDExtension {oid = 2.5.29.14 critical = false, value
= 329b98f6b6225e92ca52513d3bfc43ee02aa9121}}}"
}
```

Delete a Certificate from a KSS Keystore

Deletes 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. For more information, see ["cURL Access"](#) on page 1-3.

```
curl -i -X DELETE -u username:password -H keystorePassword:cHdkMQ== -H
keyPassword:bXlQd2Qy
http://myhost:7001/idaas/platform/admin/v1/keystoreservice/certificates?stripeName=myStripe&keystoreName=myKeystore&keyAlias=myAlias "
```

Example of Response Header

The following shows an example of the response header. For more about the HTTP status codes, see ["Status Codes."](#)

```
HTTP/1.1 200 OK
```

Example of Response Body

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

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

Create a Secret Key for a KSS Keystore

Creates 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 ["Status Codes."](#)

```
HTTP/1.1 200 OK
```

Example of Response Body

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

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

View Secret Key Properties for a KSS Keystore

Returns 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 <i>n</i> "	List of secret key properties, where <i>n</i> 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. For more information, see "[cURL Access](#)" on page 1-3.

```
curl -i -X GET -u username:password -H keystorePassword:bx1Qd2Q= -H
keyPassword:bx1LZX1Qd2Q=
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 "[Status Codes](#)."

```
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 a KSS Keystore

Deletes 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. For more information, see ["cURL Access"](#) on page 1-3.

```
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 ["Status Codes."](#)

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

Token Issuer Trust Management

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
Create Trusted Issuers and DN Lists	POST	/idaas/webservice/admin/v1/trust/issuers
View a Trusted Issuer and DN Lists	GET	/idaas/webservice/admin/v1/trust/issuers
Create a Token Attribute Rule for a Trusted DN	POST	/idaas/webservice/admin/v1/trust/token
View All Token Attribute Rules	GET	/idaas/webservice/admin/v1/trust/token

Create Trusted Issuers and DN Lists

Create trusted issuers and distinguished name (DN) lists for signing certificates. The trusted issuers will be stored in the specified trusted issuers document.

REST Request

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

Parameters

The following table summarizes the POST request parameters.

Name	Description	Type
documentName	Name of trusted issuer document.	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, <code>www.yourcompany.com</code> . The default value for the predefined SAML client policies is <code>www.oracle.com</code> .	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. For more information, see ["cURL Access"](#) on page 1-3.

```
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",  
}
```

View a Trusted Issuer and DN Lists

Returns a trusted issuer and its distinguished name (DN) lists based on the document name provided.

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. For more information, see ["cURL Access"](#) on page 1-3.

```
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", ]
      }
    ]
  }
}
```


Create a Token Attribute Rule for a Trusted DN

Creates a token attribute rule for a trusted distinguished name (DN). 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: <code>http(s)://host</code> or <code>http(s)://host/root</code> . For example, if you set this value to <code>https://messaging.us2.com/</code> , then the attribute rule applies to all service invocations with the service URL of the form <code>https://messaging.us2.com/<path></code>
"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. For more information, see "[cURL Access](#)" on page 1-3.

```
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"  
}
```

View All Token Attribute Rules

Returns all token attribute rules for a trust 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

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: <code>http(s)://host</code> or <code>http(s)://host/root</code>
"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 <code>yourTrusted*</code> . 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 <code>"yourTrusted"</code> . 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. For more information, see "[cURL Access](#)" on page 1-3.

```
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"
          }
        },
      }
    ]
  }
}
```


A

Summary of REST APIs, Alphabetical by Resource Path

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	View All Credentials in the Credential Store
/idaas/platform/admin/v1/credential	DELETE	Delete a Credential from the Credential Store
/idaas/platform/admin/v1/credential	POST	Create a New Credential in the Credential Store
/idaas/platform/admin/v1/credential	PUT	Update a Credential in the Credential Store
/idaas/platform/admin/v1/keystore	GET	View All Aliases in the JKS Keystore
/idaas/platform/admin/v1/keystore/{alias}	GET	View a Trusted Certificate in the JKS Keystore
/idaas/platform/admin/v1/keystore/{alias}	DELETE	Delete a Trusted Certificate from the JKS Keystore
/idaas/platform/admin/v1/keystore/{alias}	POST	Import a Trusted Certificate into the JKS Keystore
/idaas/platform/admin/v1/keystore/pkcs7/{alias}	POST	View a Trusted Certificate in the JKS Keystore
/idaas/platform/admin/v1/keystoreservice	DELETE	Delete a KSS Keystore
/idaas/platform/admin/v1/keystoreservice	POST	Create a KSS Keystore
/idaas/platform/admin/v1/keystoreservice	PUT	Update the Password for a KSS Keystore
/idaas/platform/admin/v1/keystoreservice/alias/{stripeName}/{keystoreName}/{entryType}	GET	View the Alias for the KSS Keystore

REST Resource	Method	More Information
/idaas/platform/admin/v1/keystoreservice/certificates	GET	View a Trusted Certificate in the KSS Keystore
/idaas/platform/admin/v1/keystoreservice/certificates	DELETE	Delete a Certificate from a KSS Keystore
/idaas/platform/admin/v1/keystoreservice/certificates	POST	Import a Trusted Certificate into a KSS Keystore
/idaas/platform/admin/v1/keystoreservice/keystore	POST	Import a KSS Keystore
/idaas/platform/admin/v1/keystoreservice/secretkey	GET	View Secret Key Properties for a KSS Keystore
/idaas/platform/admin/v1/keystoreservice/secretkey	POST	Create a Secret Key for a KSS Keystore
/idaas/platform/admin/v1/keystoreservice/{stripeName}	GET	View All KSS Keystores for a Stripe
/idaas/webservice/admin/v1/trust/issuers	GET	View a Trusted Issuer and DN Lists
/idaas/webservice/admin/v1/trust/issuers	POST	Create Trusted Issuers and DN Lists
/idaas/webservice/admin/v1/trust/token	GET	View All Token Attribute Rules
/idaas/webservice/admin/v1/trust/token	POST	Create a Token Attribute Rule for a Trusted DN