Support for REST API in Cisco Prime Access Registrar

This appendix provides information about the REpresentational State Transfer (REST) APIs supported in Cisco Prime Access Registrar. The purpose of this appendix is to provide a developer, system or network administrator, or system integrator with basic guidelines for using the outlined REST APIs within the Prime Access Registrar deployment.

This chapter contains the following sections:

REST API Framework

REST is a resource-based architectural style to create web services. A resource is an object, which could be a user, address, and so on. Each resource is identified by a Unique Resource Identifier (URI) and is manipulated by representations that pass back and forth between client and server. Representations can be in the form of XML, JSON, Plain, TEXT, or HTML. However, Prime Access Registrar supports only the JSON format.

Table D-1 lists the common operations supported in Prime Access Registrar for REST APIs.

Table D-1 Common Operations Used in REST APIs

Method	Crud Operation
ADD	Create a resource
GET	Read and retrieve a representation of a resource
EDIT	Update an existing representation
DELETE	Delete a resource

This topic contains the following sections:

REST API Services

You can use any client for creating the APIs and must pass the following information as inputs for the APIs:

- Content-Type—application/json
- username—username to access the service
- password—password to access the service

Table D-1 lists the REST APIs used in Prime Access Registrar.

Table D-2 REST API Services

Object -> Type ID	ADD	EDIT	GET	DELETE

Script -> 1	`http://<hostname>:8080/RESTAPI/service/addobject`
	
	Pass the object and the object name as data.
	
	Example:
	
	`{"Script":{"Name":"test,..}}`
		`http://<hostname>:8080/RESTAPI/service/editobject?typeid=1&name=<object name>`	`http://<hosktname>:8080/RESTAPI/service/getobject?typeid=1&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=1&name=<object name>`

Client -> 2	`http://<hostname>:8080/RESTAPI/service/addobject` Pass the object and the object name as data.	`http://<hostname>:8080/RESTAPI/service/editobject?typeid=2&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/getobject?typeid=2&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=2&name=<object name>`
Service -> 3	`http://<hostname>:8080/RESTAPI/service/addobject` Pass the object and the object name as data.	`http://<hostname>:8080/RESTAPI/service/editobject?typeid=3name=<object name>`	`http://<hostname>:8080/RESTAPI/service/getobject?typeid=3&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=3&name=<object name>`
Policy -> 4	`http://<hostname>:8080/RESTAPI/service/addobject` Pass the object and the object name as data.	`http://<hostname>:8080/RESTAPI/service/editobject?typeid=4&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/getobject?typeid=4&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=4&name=<object name>`
ResourceManager -> 5	`http://<hostname>:8080/RESTAPI/service/addobject` Pass the object and the object name as data.	`http://<hostname>:8080/RESTAPI/service/editobject?typeid=5&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/getobject?typeid=5&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=5&name=<object name>`

Table D-2 REST API Services (continued)

Object -> Type ID	ADD	EDIT	GET	DELETE
Administrator -> 6	http://<hostname>:8080/RESTAPI/service/addobject Pass the object and the object name as data.	http://<hostname>:8080/RESTAPI/service/editobject?typeid=6&name=<object name>	http://<hostname>:8080/RESTAPI/service/getobject?typeid=6&name=<object name>	http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=6&name=<object name>
RemoteServer -> 7	http://<hostname>:8080/RESTAPI/service/addobject Pass the object and the object name as data.	http://<hostname>:8080/RESTAPI/service/editobject?typeid=7&name=<object name>	http://<hostname>:8080/RESTAPI/service/getobject?typeid=7&name=<object name>	http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=7&name=<object name>
UserGroup -> 8	http://<hostname>:8080/RESTAPI/service/addobject Pass the object and the object name as data.	http://<hostname>:8080/RESTAPI/service/editobject?typeid=8&name=<object name>	http://<hostname>:8080/RESTAPI/service/getobject?typeid=8&name=<object name>	http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=8&name=<object name>
Profile -> 9	http://<hostname>:8080/RESTAPI/service/addobject Pass the object and the object name as data.	http://<hostname>:8080/RESTAPI/service/editobject?typeid=9&name=<object name>	http://<hostname>:8080/RESTAPI/service/getobject?typeid=9&name=<object name>	http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=9&name=<object name>
Replication -> 10	http://<hostname>:8080/RESTAPI/service/addobject Pass the object and the object name as data.	http://<hostname>:8080/RESTAPI/service/editobject?typeid=10&name=<name of Repmember>	http://<hostname>:8080/RESTAPI/service/getobject?typeid=10&name=<name of Repmember>	http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=10&name=Replication/Rep+Members/<name of Repmember>
Rule -> 11	http://<hostname>:8080/RESTAPI/service/addobject Pass the object and the object name as data.	http://<hostname>:8080/RESTAPI/service/editobject?typeid=11&name=<object name>	http://<hostname>:8080/RESTAPI/service/getobject?typeid=11&name=<object name>	http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=11&name=<object name>
SessionManager -> 12	http://<hostname>:8080/RESTAPI/service/addobject Pass the object and the object name as data.	http://<hostname>:8080/RESTAPI/service/editobject?typeid=12&name=<object name>	http://<hostname>:8080/RESTAPI/service/getobject?typeid=12&name=<object name>	http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=12&name=<object name>

Table D-2 REST API Services (continued)

Object -> Type ID	ADD	EDIT	GET	DELETE
Snmp -> 13	`http://<hostname>:8080/RESTAPI/service/addobject` Pass the object and the object name as data.	—	—	—
RemoteODBCSession Server -> 14	`http://<hostname>:8080/RESTAPI/service/addobject` Pass the object and the object name as data.	—	—	—
UserList -> 16	`http://<hostname>:8080/RESTAPI/service/addobject` Pass the object and the object name as data.	`http://<hostname>:8080/RESTAPI/service/editobject?typeid=16&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/getobject?typeid=16&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=16&name=<object name>`
EncryptedIMSI-PrivateKeys -> 18	`http://<hostname>:8080/RESTAPI/service/addobject`	`http://<hostname>:8080/RESTAPI/service/editobject?typeid=18&name=<name of key>`	`http://<hostname>:8080/RESTAPI/service/getobject?typeid=18&name=<name of key>`	`http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=18&name=EncryptedIMSI-PrivateKeys/keys/<name of key>`
ODBCDataSource -> 44	`http://<hostname>:8080/RESTAPI/service/addobject` Pass the object and the object name as data.	`http://<hostname>:8080/RESTAPI/service/editobject?typeid=44&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/getobject?typeid=44&name=<object name>`	`http://<hostname>:8080/RESTAPI/service/deleteobject?typeid=44&name=<object name>`

Table D-3 provides a list of other REST APIs used in Prime Access Registrar.

Table D-3 Other APIs used in Prime Access Registrar

Operation	URL
Tacacs Statistics	http://<hostname>:8080/RESTAPI/service/Tacacsstatistics
Diameterstatistics	http://<hostname>:8080/RESTAPI/service/Diameterstatistics
Statistics	http://<hostname>:8080/RESTAPI/service/Statistics
RemoteServerStats	http://<hostname>:8080/RESTAPI/service/RemoteServerStats
DiameterPeerStats	http://<hostname>:8080/RESTAPI/service/DiameterPeerStats
DiaRemoteServerStats	http://<hostname>:8080/RESTAPI/service/DiaRemoteServerStats
ClientStats	http://<hostname>:8080/RESTAPI/service/ClientStats

Table D-3 Other APIs used in Prime Access Registrar (continued)

Operation	URL
Reload	http://<hostname>:8080/RESTAPI/service/Reload
AddUser	http://<hostname>:8080/RESTAPI/service/adduser?UserListName=<nameofUsernameList>
GetUser	http://<hostname>:8080/RESTAPI/service/getuser?name=<nameof user>&UserListName=<nameofuserlist>
EditUser	http://<hostname>:8080/RESTAPI/service/edituser?name=< nameof user >&UserListName=<nameofuserlist>
DeleteUser	http://<hostname>:8080/RESTAPI/service/deleteuser?name=< nameof user>&UserListName=<nameofuserlist>

Example for adding a user using REST interface:

```
curl http://hostname:8080/RESTAPI/service/adduser?UserListName=new -H "Content-Type:
application/json" -H "Authorization: Basic YWRtaW46YWljdXNlcg==" --data
"{\"User\":{\"Name\":\"TestUser\",\"Description\":\"\",\"Password\":\"testuser\",\"Enabled
\"=\"TRUE\",\"AllowNullPassword\":\"FALSE\",\"Attributes\":{\"User-Name\":\"joe\",\"Nas-Po
rt\":\"3\"}}"
```

Always the JSON input must start with name of the objects while editing sub objects. A sample is given below:

```
curl -k -X PUT -H "Authorization: Basic YWRtaW46YWljdXNlcg==" -H "Content-Type:
application/json" 'https://<hostname>:8443/RESTAPI/service/editobject?typeid=18&name=key1'
-data
"{\"AllowedKeyIdentifiers\":\"hello\",\"keys\":[{\"Name\":\"key1\",\"identifier\":\"100\",
\"PrivateKey\":\"test456\"}]}"
```

Note REST interface can also be accessed using HTTPS through the 8443 port.

CoA and PoD REST APIs

The Change of Authorization (CoA) and Packet of Disconnect (PoD) API calls allow you to send session reauthentication and session disconnect commands for a specified session.

You can use any client for creating the APIs and must pass the following information as inputs for the APIs:

- URL—URL to access the PoD/CoA service. Example:
  - For PoD—http://<hostname>:8080/RESTAPI/service/PoD
  - For CoA—http://<hostname>:8080/RESTAPI/service/CoA
- Content-Type—application/json
- username—username to access the service
- password—password to access the service
- data—API body with syntax as listed in Table D-4

Examples

The following example shows a sample PoD API written using cURL client:

```
http://ar-lnx-vm054:8080/RESTAPI/service/PoD -H "Content-Type: application/json" -H
"username:admin -H "password:aicuser --data
"{"parameter":"S21","value":"","type":"with-id"}"
```

Prime Access Registrar supports basic authentication with Base64 encoding support for username and password.

A sample header on encryption is provided in the example below:

```
-H "Authorization: Basic YWRtaW46YWljdXNlcg=="
```

The following example shows a sample CoA API:

```
curl http://10.197.95.187:8080/RESTAPI/service/CoA -H "Authorization: Basic
YWRtaW46YWljdXNlcg==" -H "Content-Type: alication/json" --data
"{\"parameter\":\"bob\",\"value\":\"\",\"type\":\"with-user\"}"
```

Note REST interface can also be accessed using HTTPS through the 8443 port.

We can also send CoA using **with-profile** option along with the existing parameters using REST API.

The parameters supported for REST API for CoA with-profile option are: **with-id**, **with-user**, **with-key**, **with-nas**, **with-ip-address**, **with-ipx-network**, **with-age**, **with-usr-vpn**, **with-attribute**, **with-Home-Agent**, and **with-IP-Subnet**.

Example:

```
curl  'http://10.197.95.162:8080/RESTAPI/service/CoA' -H "Authorization: Basic
YWRtaW46YWljdXNlcg==" -H "Content-Type: application/json"  --data
"{\"parameter\":\"bob1\",\"value\":\"bob1\",\"type\":\"with-user\",\"profileType\":\"with-
profile\",\"profileValue\":\"cap\"}"
```

Table D-4 Parameter and Data Syntax for APIs

Parameter	Data Syntax/Example
with-profile	"{"profileType":"with-profile","profileValue":"cap"}"
with-id	"{"parameter":"S21","value":"","type":"with-id"}"
with-user	"{"parameter":"bob","value":"","type":"with-user"}"
with-key	"{"parameter":"bob","value":"","type":"with-key"}"
with-nas	"{"parameter":"localhost","value":"","type":"with-nas"}"
with-ip-address	"{"parameter":"192.168.0.4","value":"","type”:”with-ip-address"}"
with-ipx-network	"{"parameter":"0x6","value":"","type":"with-ipx-network"}"
with-age	"{"parameter":"1S","value":"","type":"with-age"}"
with-usr-vpn	"{"parameter":"1","value":"","type":"with-usr-vpn"}"
with-attribute	"{"parameter":"Framed-IP-Address","value":"192.168.0.1","type":"with-attrib ute"}"
with-Home-Agent	"{"parameter":"","value":"","type":"with-Home-Agent"}"
with-IP-Subnet	"{"parameter":"","value":"","type":" with-IP-Subnet "}"

Prime Access Registrar supports **send-CoA** using CLI interface as well. For configuring **send-CoA** using CLI, see the "query-sessions" section in the "Setting the Cisco Prime Access Registrar Configurable Option" chapter of the *Cisco Prime Access Registrar 9.2 Administrator Guide*.

REST API Support for Query and Release Sessions

The REST interface allows you to perform the following:

- Query the server about the currently active user sessions
- Release the currently active user sessions

You can request information about those sessions that match a specified filter type, which could be one of the following:

- with-id
- with-user
- with-key
- with-nas
- with-ip-address
- with-ipx-network
- with-age
- with-usr-vpn
- with-attribute
- with-Home-Agent
- with-IP-Subnet

Table D-5 lists the details of REST APIs for query and release session services.

Table D-5 REST APIs for Query and Release Sessions

Service	URL	Inputs	Sample API

Query Session	`http://<hostname>:8080/RESTAPI/service/querySessions?path=/r&filterType=with-user&filterValue=bob`	• Content-Type—application/json
		• username—username to access the services
		• password—password to access the service
		• filterType—as listed above
			`curl -H "username:admin" -H "password:aicuser" 'http://ar-lnx-vm038:8080/RESTAPI/service/querySessions?path=/r&filterType=with-user&filterValue=bob'`
			`{"session-mgr-1":{"S3":{"Username":"bob","Key":"localhost:1","Nas":"localhost","IP":"192.168.0.0","IPX":"0x1","GSL":"1","USL":"1","User VPN":"1","Nas-port":"1","Time":"00:01:17","User-Name":"bob"}}}`
			`[root@ar-lnx-vm049 ~]#`

Release Session	`http://<hostname>:8080/RESTAPI/service/releasesessions?path=/r&filterType=with-user&filterValue=bob`		`curl -X GET -H "username:admin" -H "password:aicuser" -H "Content-Type: application/json" 'http://ar-lnx-vm041:8080/RESTAPI/service/releasesessions?path=/r/SessionManagers/&filterType=with-user&filterValue=bob'` `Released 1 session(s) Successfully in /Radius/SessionManagers`

Support for RADIUS to JSON and JSON to RADIUS Translation

Prime Access Registrar allows you to translate incoming radius requests to JSON format and vice versa.

The REST interface is extended to accommodate this functionality. This translation is supported for the following scenarios:

- Authorization
- Accounting (Start/Interim-Update/Stop)
- Change of Authorization/Packet of Disconnect (CoA/PoD)
- Session manager

Note This translation is not supported for authentication.

The following are CLI configurations to support this feature:

```
--> ls -R /r/services/restproxy

[ restproxy ]
Name = restproxy
Description =
Type = rest
IncomingScript~ =
OutgoingScript~ =
OutagePolicy~ = RejectAll
OutageScript~ = myscript
MultipleServersPolicy = Failover
RemoteServers/
1. restRM


--> ls -R

[ //localhost/Radius/RemoteServers/rest ]
Name = rest
Description =
Protocol = rest
ReactivateTimerInterval = 300000
Timeout = 5000
MaxTimeOuts = 3
RESTSourceConnections = 16
RequestURL = http://10.81.78.143:8080/eapauth/IMSI/CISCO/NASId/NASIP/Port/authorization
HTTPVersion = HTTP2
UserName = eapAuth32TMUS
Password = <encrypted>
KeepAliveTimerInterval = 0
RequestToJSONRequestMappings/
RequestToQueryMappings/
CISCO = Cisco-AVPair
IMSI = User-Name
NASId = NAS-Identifier
NASIP = NAS-IP-Address
Port = NAS-Port
```

CSRF Token Implementation using REST

Prime Access Registrar supports Cross-Site Request Forgery (CSRF) check for enhanced security. A CSRF token is introduced to handle a CSRF request. This is an optional feature and is backward compatible.

To use the CSRF token:

1. Enable CSRF Token in RestCSRF.properties under /cisco-ar/apache-tomcat-9.0.31/webapps/RESTAPI/WEB-INF/classes/RestCSRF.properties

   Set the value to **YES** as shown below. Default is NO.

   ```
   CSRF-TOKENS=YES
   ```

2. Set the timer for the CSRF token in RestCSRF.properties. The token expires based on the timer value. Default time value is 5 mins.
3. Generate CSRF tokens based on the authentication parameters. See the sample command below:

   ```
   curl
   [http://%3cIpaddress:port%3e/RESTAPI/service/getlogin]http://<Ipaddress:port>/RESTAPI/
   service/getlogin -H "Authorization:Basic < Authentication paramert>"

   Output:

   {CSRF tokens :< csrf-token >}
   ```

4. You can perform curl operations using the CSRF token. See the sample command below:

   ```
   curl  'http:// ://<Ipaddress:port>/RESTAPI/service/addobject' -H "csrf-token: <
   csrf-token >" -H "Content-Type: application/json" --data
   "{\"Service\":{\"Name\":\"null\",\"Description\":\"\",\"Type\":\"null\",\"IncomingScri
   pt\":\"\",\"OutgoingScript\":\"\"}"
   ```