Oracle® Communications Convergent Charging Controller

REST Technical Guide

Release 15.2
G48280-01
January 2026

Oracle Communications Convergent Charging Controller REST Technical Guide, Release 15.2

G48280-01

Contents

About This Content

This document includes all the information required to install, configure and administer the RESTClient and RESTServer application.

Audience

This document is intended for system administrators and persons installing, configuring and administering the RESTClient and RESTServer application. However, sections of the document may be useful to anyone requiring an introduction to the application.

A solid understanding of UNIX and a familiarity with IN concepts are an essential prerequisite for safely using the information contained in this technical guide. Attempting to install, remove, configure or otherwise alter the described system without the appropriate background skills, could cause damage to the system; including temporary or permanent incorrect operation, loss of service, and may render your system beyond recovery.

Although it is not a prerequisite to using this guide, familiarity with the target platform would be an advantage.

This manual describes system tasks that should only be carried out by suitably trained operators.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document.

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

Part I
REST Client

This part provides the REST Client related information. It contains the following chapters:

1

System Overview

This chapter provides a high-level overview of the application. It explains the basic functionality of the system and lists the main components.

It is not intended to advise on any specific Oracle Communications Convergent Charging Controller network or service implications of the product.

What is RESTClient?

The RESTClient (REST) interface is used to send REST requests from Convergent Charging Controller to REST server endpoints.

Request Processing

RESTClient accepts the requests in xml format on a specific port that is configured and translates the xml request to JSON format. RESTClient then forwards the JSON request to REST server, which is accepted at the REST server endpoint.

Response Handling

RESTClient encapsulates response received from the REST server, transforms it to xml, and sends it back to the requesting process (DAP) with a response code of 200. The actual result code or error code is encoded in the response xml.

Features

- RESTClient supports requests to Balance Transfer and Apply Loan endpoints in the Oracle Communications Billing and Revenue Management (BRM) REST server.
- RESTClient also supports generic request types, which can be forwarded towards any REST server endpoints with required design time customizations.

2
Configuration

This chapter explains how to configure the Oracle Communications Convergent Charging Controller application.

RESTClient Configuration

RESTClient reads its configuration from the **config.json** file. The **config.json** file is located in the **/IN/service_packages/REST/etc** directory.

RESTClient config.json Section

To organize the configuration data within the config file, some sections are nested within other sections. Configuration details are opened and closed using { }.

- Groups of parameters are enclosed with curly brackets – { }
- Comments are prefaced with a "//" at the beginning of the line

Editing the File

Open the configuration file on your system using a standard text editor. Do not use text editors, such as Microsoft Word, that attach control characters. These can be, for example, Microsoft DOS or Windows line termination characters (for example, ^M), which are not visible to the user, at the end of each row. This causes file errors when the application tries to read the configuration file.

Always keep a backup of your file before making any changes to it. This ensures you have a working copy to which you can return.

Loading Config Changes

If you change the configuration file, you must restart the service to enable the new options to take effect.

Wallet Management for oAuth

You need to create an Oracle wallet to store and manage OAM server clientId and clientSecrets, after installing the REST client (and before triggering any Auth requests). Wallets are created using **/u01/app/oracle/product/12.2.0/bin/mkstore** tool.

Perform the following steps:

1. Create an empty Oracle wallet.
2. Store the credentials of OAM server for Auth requests.

Example

```
mkstore -wrl /IN/service_packages/REST/etc/wallet -createCredential REST
username password
```

```
mkstore -wrl /IN/service_packages/REST/etc/wallet -createCredential
connect_string username password
```

> **Note**
>
> connect_string should be different for different credentials.

Configuration

1. Configure location of the wallet in **config.json** file. For example, **/IN/service_packages/REST/etc/wallet**.
2. Generate the base64 encoded value of the wallet password and configure it in **config.json** file. For example, to generate the base64 encoded wallet password, run the following command:

   ```
   echo -n "wallet_password" | base64
   ```

3. Configure the clientId in **config.json** file.

SSL Configuration

When endpoint contains https, SSL configuration is used for certification validation. SSL configuration is taken from default java configuration. Default truststore is present in **java/bin** directory.

To import the certificate, run the **keytool** command as an administrator or root user. For example:

```
keytool -importcert -alias cert_alias_name -file  ./ssl_cert.pem -
keystore /usr/java/jdk1.8.0_261/jre/lib/security/cacerts
```

3
Background Processes

This chapter explains the process which runs automatically as part of the Oracle Communications Convergent Charging Controller application. This process is started automatically by the system services (**/IN/bin/OUI_systemctl.sh**) in the SLC node.

RESTClient

Purpose

The RESTClient (REST) interface is used to trigger REST requests towards REST server endpoints.

Startup

This task is started by the system services, by the following line in the service files:

/IN/service_packages/REST/bin/RestClientStartup.sh config.json

Configuration

The high-level structure of the REST client is shown below:

```
{
  "maxthreadcount": 1000,
  "port": 4050,
  "webroot": "/tmp",
  "walletlocation": "Location",
  "walletkey": "key",

  "restendpoint": {
    "BalanceTransfer" : {
      "endpoint" : "BalanceTransferEndpoint",
      "servergroupid" : "BRM",
      "resourceid": 840,
      "chargesource": false,
      "chargedestination": false
    },
    "ApplyLoan" : {
      "endpoint" : "ApplyLoanEndpoint",
      "serviceType" : "/service",
      "servergroupid" : "BillingCare",
      "resourceid": 840
    },
    "GenericRequest" : {
      "endpoint" : "GenericRequestEndpoint",
      "type" : "POST",
      "servergroupid" : "BillingCare"
    }
  },
  "serverlist": {
```

```
    "BRM": {
      "tokenendpoint":"BRMOAuthEndpoint",
      "clientid":"username"
    },
    "BillingCare": {
      "tokenendpoint":"BCOAuthEndpoint",
      "clientid":"username"
    }
  }
}
```

Parameters

Parameters of the REST client are listed below.

maxthreadcount

Syntax:	`maxthreadcount: "value"`
Description:	Maximum number of thread the java process can have.
Type:	Integer
Optionality:	Optional
Allowed:	NA
Default:	1000
Example:	`maxthreadcount: "1000"`

port

Syntax:	`port: "value"`
Description:	Port on which REST client is listening to DAP requests.
Type:	Integer
Optionality:	Optional
Allowed:	NA
Default:	4050
Example:	`port: "4050"`

webroot

Syntax:	`webroot: "value"`
Description:	Path to which client will be listening. When “/” is specified, the client will be listening to “http://localhost:port/”. In API, “/” should be present at the end in the DAP configuration screen.
Type:	String
Optionality:	Optional
Allowed::	NA
Default:	"/"
Example:	`webroot: "/tmp"`

walletlocation

Syntax:	`walletlocation: "value"`
Description:	Location where the wallet is created, which stores the REST endpoint username and password.
Type:	String
Optionality:	Optional
Allowed:	NA
Default:	"/IN/service_packages/REST/etc/wallet"
Example:	`walletlocation: "/IN/service_packages/REST/etc/wallet"`

walletkey

Syntax:	`walletkey= "value"`
Description:	Base64 encrypted password for the wallet.
Type:	String
Optionality:	Mandatory
Allowed:	NA
Default:	“”
Example:	`walletkey: "key”`

restendpoint

Syntax:	`"value" : {`
	`"endpoint" : "value",`
	`"type" : "value",`
	`"servergroupid" : "value"`
	`}`

Description:	Group of REST operations.
Type:	JSON
Optionality:	Optional
Allowed:	NA
Default:	NA

Example:	`{ "BalanceTransfer" : {`
	`"endpoint" : "BalanceTransferEndpoint"`
	`"servergroupid" : "BillingCare"`
	`}`
	`"ApplyLoan" : {`
	`"endpoint" : "ApplyLoanEndpoint",`
	`"servergroupid" : "BRM"`
	`}`
	`}`

endpoint

Syntax:	`endpoint: "value"`
Description:	Endpoint to which the request is triggered.
Type:	String
Optionality:	Optional
Allowed:	NA
Default:	“”
Example:	`endpoint: "http:localhost:restopertion/"`

type

Syntax:	`type: "value"`
Description:	The type of request. This applies to GenericRequest handler only.
Type:	String
Optionality:	Optional
Allowed:	The allowed values are: • **GET**: Request will be triggered without body. • **POST**: XML Request body will be converted to JSON body and triggered. • **PUT**: XML Request body will be converted to JSON body and triggered.
Default:	POST
Example:	`type: "POST"`

servergroupid

Syntax:	`servergroupid: "value"`
Description:	ID of the oauth credential present in serverlist.
Type:	String
Optionality:	Optional
Allowed:	Servergroupid should be present in the `serverlist`, so that corresponding oAuth endpoint and clientid is used to generate the token.
Default:	“”
Example:	`servergroupid: "BRM"`

resourceid

Syntax:	`resourceid: "value"`
Description:	Resourceid used for requesting **BalanceTransfer** and **ApplyLoan**.
Type:	Integer
Optionality:	Optional
Allowed:	Only used in **BalanceTransfer** and **ApplyLoan**.
Default:	840
Example:	`resourceid: "840"`

chargesource

Syntax:	`chargesource: "Boolean"`

Description:	This is a boolean flag to indicate whether the source account to be charged for balance transfer. The <chargeSource> xml input field will hold this value. If xml does not carry this information, then a default value is taken from the **config.json** file.
Type:	Boolean
Optionality:	Optional
Allowed:	Only used in BalanceTransfer.
Default:	false
Example:	`chargesource: "false"`

chargedestination

Syntax:	`chargedestination: "Boolean"`
Description:	This is a boolean flag to indicate whether the target/destination account to be charged for balance transfer. The <chargeDestination> xml input field will hold this value. If xml does not carry this information, then a default value is taken from the **config.json** file.
Type:	Integer
Optionality:	Optional
Allowed:	Only used in BalanceTransfer.
Default:	false
Example:	`chargedestination: "false"`

serviceType

Syntax:	`serviceType : "value"`
Description:	This is the MSISDN Service type for which apply loan details are requested. The <serviceType> xml input field will hold this value. If xml does not carry this information, then a default value is taken from the **config.json** file
Type:	String
Optionality:	Optional
Allowed:	Only used in ApplyLoan.
Default:	/service
Example:	`serviceType : "/service"`

serverlist

Syntax:	`"value": {`
	`"tokenendpoint":"value",`
	`"clientid":"value"`
	`}`

Description:	Group of OAuth credentials.
Type:	JSON
Optionality:	Mandatory
Allowed:	NA
Default:	NA

Example:	<pre>"BRM": {
	"tokenendpoint":"OAuthendpoint",
	"clientid":"username"
	}</pre>

tokenendpoint

Syntax:	`tokenendpoint: "value"`
Description:	Endpoint to which token request will be triggered.
Type:	String
Optionality:	Mandatory
Allowed:	NA
Default:	“”
Example:	`tokenendpoint: "OAuthendpoint"`

clientid

Syntax:	`clientid: "value"`
Description:	Username used for OAuth token request. Corresponding password should be added in the wallet.
Type:	String
Optionality:	Mandatory
Allowed:	NA
Default:	NA
Example:	`clientid: "username"`

4

About Installation

This chapter provides information about the installed components for the Oracle Communications Convergent Charging Controller application described in this guide. It also lists the files installed by the application that you can check for, to ensure that the application is installed successfully.

Installation Overview

For information about the following requirements and tasks, see *Installation Guide*:

- Convergent Charging Controller system requirements
- Pre-installation tasks
- Installing and removing Convergent Charging Controller packages

RESTClient Package

Installation of Oracle Communications Convergent Charging Controller RESTClient includes **restScp** package on SLC.

Checking the Installation

Refer to the following checklist to ensure that RESTClient is installed correctly.

Checklist - SLC

Follow the steps in this checklist to ensure RESTClient is installed correctly on an SLC machine.

1. Log in to the SLC machine as root.
2. Check that the following directory structure exists, with subdirectories:
   /IN/service_packages/REST
3. Check that directories contain subdirectories and that all are owned by:
   smf_oper user (group esg)

Process list - SLC

If the application is running correctly, **RESTClient.jar** process should be running on each SLC, started during OUI_systemctl startup.

RESTClient Directories and Files

The RESTClient installation on SLC creates the following directories:

- **/IN/services_packages/REST/bin**
- **/IN/services_packages/REST/lib**
- **/IN/services_packages/REST/etc**

- **/IN/services_packages/REST/tmp**

Installing RESTClient installs the following interface:

/IN/services_packages/REST/bin/RESTClient.jar

Installing RESTClient installs the following configuration file:

/IN/services_packages/REST/etc/config.json.example

> **Note**
>
> You need to create the **config.json** file with actual values required during runtime.

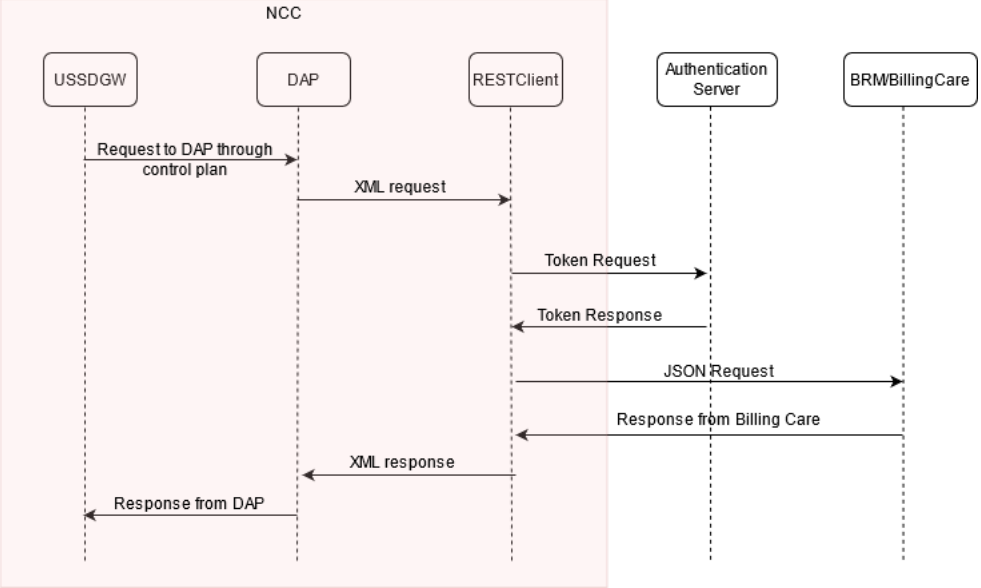
5
RESTClient Call Flows

This chapter provides a sample REST request flow.

Request Flow

Figure 5-1 shows the request and response exchange between USSDGW, DAP, RESTClient, and REST server. The flow depicts a simple scenario where RESTClient receives an xml request from DAP. RESTClient converts the incoming xml request from DAP to a JSON request and that JSON request is sent to the REST server (BRM). RESTClient then receives a response from the REST server. The response is sent back to DAP and a message will be sent to the subscriber based on the response from the REST server.

Figure 5-1 REST Request Flow



6
Supported Request Type

RESTClient supports three types of requests:

- BalanceTransfer
- ApplyLoan
- GenericRequest

> **Note**
>
> If the location fetch fails during BalanceTransfer or ApplyLoan APIs, Convergent Charging Controller does not send any location attributes to BRM. In such cases, taxes are applied based on how it is configured in the BRM system.

BalanceTransfer

BalanceTransfer request is triggered when the following parameter is present in the input xml request:

```
<operation>BalanceTransfer</operation>
```

> **Note**
>
> - Configure **BalanceTransfer** under **restendpoint** in **config.json** file.
> - The output request will be always POST.

Input Request

Table 6-1 describes the parameters accepted in the XML `<requestDetails>` tag.

Table 6-1 Input Request Parameters

Notification Type	Mandatory/ Optional	Description
sourceMSISDN	Mandatory	Source MSISDN for BalanceTransfer.
destinationMSISDN	Mandatory	Destination MSISDN for BalanceTransfer.
transferAmount	Mandatory	Amount to be transferred from source to destination.
transferAmountType	Optional	ResourceId of the amount passed. If not present in the input request, default value of **resourceid** from the config file is used.

Table 6-1 (Cont.) Input Request Parameters

Notification Type	Mandatory/ Optional	Description
chargeSource	Optional	Boolean flag to indicate whether the source account will be charged. If not present in the input request, default value of **chargesource** from the config file is used.
chargeDestination	Optional	Boolean flag to indicate whether the target or destination account will be charged. If not present in the input request, default value of **chargedestination** from the config file is used.
sourceZoneMapTarget	Optional	Cellid for source MSISDN. If not present in the input request, location will not be sent to the REST server.
targetZoneMapTarget	Optional	Cellid for destination MSISDN. If not present in the input request, location will not be sent to the REST server.

Sample XML Input Request

Following is a sample XML input request sent out from DAP module.

```
<operation>BalanceTransfer</operation>
<requestDetails>
<sourceMSISDN>90989098</sourceMSISDN>
<destinationMSISDN>89878987</destinationMSISDN>
<transferAmount>10</transferAmount>
<transferAmountType>840</transferAmountType>
<chargeSource>true</chargeSource>
<chargeDestination>false</chargeDestination>
<sourceZoneMapTarget>52021005DC03EA</sourceZoneMapTarget>
<targetZoneMapTarget>52121005DC03EA</targetZoneMapTarget>
</requestDetails>
```

REST Request Parameters

Table 6-2 lists the parameters sent to the REST server.

Table 6-2 REST Request Parameters

Notification Type	Mapping From Input Request
sourceRef.id	From **sourceMSISDN**.
sourceRef.type	Always set as **service**.
targetRef.id	From **destinationMSISDN**.
targetRef.type	Always set as **service**.
transferAmount	From **transferAmount**.
transferAmountType	From **transferAmountType**. If not present in the input request, **restendpoint.BalanceTransfer.resourceid** from the config file is used.

Table 6-2 (Cont.) REST Request Parameters

Notification Type	Mapping From Input Request
chargeSource	From **chargeSource**. If not present in the input request, **restendpoint.BalanceTransfer.chargesource** from the config file is used.
chargeDestination	From **chargeDestination**. If not present in the input request, **restendpoint.BalanceTransfer.chargedestination** from the config file is used.
description	Same as operation name.
sourceLocation. zoneMapTarget	From **sourceZoneMapTarget**.
targetLocation. zoneMapTarget	From **targetZoneMapTarget**.

Sample REST API Request

Following is a sample REST API request sent out from RESTClient towards REST endpoint.

```
{
  "description" : "BalanceTransfer",
  "sourceRef" : {
    "id" : "90989098",
    "type" : "service"
  },
  "targetRef" : {
    "id" : "89878987",
    "type" : "service"
  },
  "transferAmount" : 10,
  "transferAmountType" : 840,
  "chargeSource" : true,
  "chargeDestination" : false,
  "sourceLocation" : {
    "zoneMapTarget" : "52021005DC03EA"
  },
  "targetLocation" : {
    "zoneMapTarget" : "52121005DC03EA"
  }
}
```

REST API Response

Tags present in the response will not be validated. JSON message will be directly converted to XML.

Sample REST API Response

Following is the snippet of response received from the REST endpoint.

```
{
    "extension": null,
    "id": "0.0.0.1+-event-audit-transfer_balance+335711686285720131",
    "uri": "http://hostname:port/bcws/webresources/v1.0/billunits/
```

```
balancegroups/transferbalance/0.0.0.1+-event-audit-
transfer_balance+335711686285720131",
    "transferAmount": 10,
    "transferAmountType": 840,
    "sourceRef": {
        "id": "90989098",
        "type": "service"
    },
    "targetRef": {
        "id": "89878987",
        "type": "service"
    },
    "sourceBucket": [
        {
            "validFrom": 1647993600000,
            "validTo": 0,
            "currentBalance": 2147485311.57
        }
    ],
    "targetBucket": [
        {
            "validFrom": 1647993600000,
            "validTo": 0,
            "currentBalance": -2147483808.93
        },
        {
            "validFrom": 1632230093000,
            "validTo": 1703164493000,
            "currentBalance": 0
        }
    ],
    "sourceTransferFee": {
        "amount": 3.00,
        "feeTax": 0.30,
        "resourceId": 840
    },
    "targetTransferFee": {
        "amount": 3.00,
        "feeTax": 0.60,
        "resourceId": 840
    }
}
```

where, *hostname* and *port* is the hostname and port of the machine where REST server is running.

XML Response

XML response will have extra tag `<rest_result_code>` containing the status code from REST response header. The status code from RESTClient to DAP will always be 200. `<rest_status_code>` can be used for checking the status code from REST server.

Sample XML Response

Following is the XML response received by DAP for a particular request.

```
<targetRef>
    <id>89878987</id>
    <type>service</type>
</targetRef>
<extension>null</extension>
<targetBucket>
    <currentBalance>-2.14748380893E9</currentBalance>
    <validFrom>1647993600000</validFrom>
    <validTo>0</validTo>
</targetBucket>
<targetBucket>
    <currentBalance>0</currentBalance>
    <validFrom>1632230093000</validFrom>
    <validTo>1703164493000</validTo>
</targetBucket>
<targetTransferFee>
    <amount>3.0</amount>
    <resourceId>840</resourceId>
    <feeTax>0.6</feeTax>
</targetTransferFee>
<transferAmount>10</transferAmount>
<sourceTransferFee>
    <amount>3.0</amount>
    <resourceId>840</resourceId>
    <feeTax>0.3</feeTax>
</sourceTransferFee>
<transferAmountType>840</transferAmountType>
<id>0.0.0.1+-event-audit-transfer_balance+335711686285720131</id>
<sourceRef>
    <id>90989098</id>
    <type>service</type>
</sourceRef>
<sourceBucket>
    <currentBalance>2.14748531157E9</currentBalance>
    <validFrom>1647993600000</validFrom>
    <validTo>0</validTo>
</sourceBucket>
<uri>http://hostname:port/bcws/webresources/v1.0/billunits/balancegroups/
transferbalance/0.0.0.1+-event-audit-transfer_balance+335711686285720131</uri>
<rest_status_code>201</rest_status_code>
```

ApplyLoan

ApplyLoan request is triggered when the following parameter is present in the input request.

```
<operation>ApplyLoan</operation>
```

> **Note**
>
> - Configure **ApplyLoan** under **restendpoint** in **config.json** file.
> - The output request will be always POST.

Input Request

Table 6-3 describes the parameters accepted in the XML `<requestDetails>` tag.

Table 6-3 Input Request Parameters

Notification Type	Mandatory	Description
sourceMSISDN	Mandatory	Source MSISDN for ApplyLoan.
loanAmount	Mandatory	Amount for loan.
loanResourceId	Optional	ResourceId of the amount passed. If not present in the input request, default value of **resourceid** from the config file is used.
sourceZoneMapTarget	Optional	Cellid for source MSISDN. If not present in the input request, location will not be sent to the REST server.

Sample XML Input Request

Following is a sample XML input request sent out from DAP module.

```
<operation>ApplyLoan</operation>
<requestDetails>
<sourceMSISDN>635495522</sourceMSISDN>
<loanAmount>5</loanAmount>
<loanResourceId>840</loanResourceId>
<serviceType>/service</serviceType>
<sourceZoneMapTarget>52021005DC03EA</sourceZoneMapTarget>
</requestDetails>
```

REST Request Parameters

Table 6-4 lists the parameters sent to the REST server.

Table 6-4 REST Request Parameters

Notification Type	Mapping From Input Request
accountRef.id	Always set as **"0.0.0.1+-account+1"**.
service.id	From **sourceMSISDN**.
service.type	From **serviceType**. If not present in the input request, **restendpoint.ApplyLoan. serviceType** from the config file is used.
amount	From **loanAmount**.

Table 6-4 (Cont.) REST Request Parameters

Notification Type	Mapping From Input Request
resourceId	From **loanResourceId**. If not present in the input request, **restendpoint.ApplyLoan.resourceid** from the config file is used.
zoneMapTarget	From **userLocation**. If not present in the input request, this parameter will not be sent.

Sample REST API Request

Following is a sample REST API request sent out from RESTClient towards REST endpoint.

```
{
    "accountRef": {
        "id": "0.0.0.1+-account+1"
    },
  "service": {
        "id" : "635495522",
        "type" : "/service"
    },
    "amount": 5,
    "resourceId": 840,
    "zoneMapTarget": "52021005DC03EA"
}
```

REST API Response

Tags present in the response will not be validated. JSON message will be directly converted to XML.

Sample REST API Response

Following is the snippet of response received from the REST endpoint.

```
{
    "extension": null,
    "availableLoanBalance": 172,
    "currentBalance": 800,
    "amount": 122,
    "loanFee": 10,
    "tax": 8.54,
    "balances": [],
    "availableLoanLimit": 9510.91,
    "creditLimit": 10000,
    "loanObj": {
        "id": "0.0.0.1+-event-billing-loan_debit+335782055029906029",
        "uri": null
    },
    "loanFeeObj": {
        "id": "0.0.0.1+-event-billing-loan_fee+335782055029904493",
        "uri": null
```

```
    }
}
```

XML Response

XML response will have extra tag <rest_result_code> containing the status code from REST response header. The status code from RESTClient to DAP will always will be 200. <rest_status_code> can be used for checking the status code from REST server.

Sample XML Response

Following is the XML response received by DAP for a particular request.

```
<extension>null</extension>
<amount>122</amount>
<loanFee>10</loanFee>
<loanObj>
    <id>0.0.0.1+-event-billing-loan_debit+335782055029906029</id>
    <uri>null</uri>
</loanObj>
<currentBalance>800</currentBalance>
<creditLimit>10000</creditLimit>
<loanFeeObj>
    <id>0.0.0.1+-event-billing-loan_fee+335782055029904493</id>
    <uri>null</uri>
</loanFeeObj>
<tax>8.54</tax>
<availableLoanLimit>9510.91</availableLoanLimit>
<availableLoanBalance>172</availableLoanBalance>
<rest_status_code>201</rest_status_code>
```

GenericRequest

For sending any request (other than BalanceTransfer and ApplyLoan) to a third party REST endpoint, generic request option can be used. You can configure the required operation with the corresponding endpoint in the **config.json** file and then use the same operation name in the input xml triggered from DAP to REST client.

Configure the new operation in the **config.json** file as follows:

```
"restendpoint": {
   "OperationName" : {
      "endpoint" : "restendpoint",
      "type" : "POST",
      "servergroupid" : "BRM"
    }
 }
```

> **Note**
>
> - Generic request Handler supports POST, PUT, and GET requests.
> - servergroupid and endpoint can be custom configurations.

Sample Configuration

```
"restendpoint": {
    "GenericBalanceTransferDetails" : {
      "servergroupid" : "BillingCare"
    }
}
```

Input Request

Table 6-5 describes the parameters accepted in the XML input request.

Table 6-5 Input Request Parameters

Notification Type	Mandatory/Optional	Description
operation	Mandatory	Operation name for the request which is configured.
type	Optional	If present, it will override the type present in the config for the operation.
endPoint	Optional	If present, it will override the endpoint present in the config for the operation.
requestDetails	Mandatory for POST/PUT	For GET, requestDetails will not be sent to the REST server. For POST/PUT, XML request will be converted to JSON and sent as body in the OUTPUT request.
arrayXmlTags	Optional	Tags that need to be converted to array type in JSON request. Multiple tags can be used with comma separator. Example: <arrayXmlTags>bundles,package</arrayXmlTags>

Sample XML Request

```
<operation>GenericRequest</operation><requestDetails>
<msisdn>923450010213</msisdn> <package> <packageType>Addon</packageType>
<bundles> <bundleName>500001</bundleName> </bundles>
</package></requestDetails> <arrayXmlTags>bundles,package</arrayXmlTags>
```

Sample JSON Converted Request

```
JSON Request{
    "arrayXmlTags": "bundles,package",
    "requestDetails": {
        "package": [{
            "bundles": [{"bundleName": 500001}],
            "packageType": "Addon"
        }],
        "msisdn": 923450010213
    },
    "operation": "GenericRequest"
}
```

Sample XML Input Request

This example shows a GET request for balance transfer audit object using Generic request Handler.

```
<operation>GenericBalanceTransferDetails</operation>
<endPoint>http://hostname:port/bcws/webresources/v1.0/billunits/balancegroups/
transferbalance/0.0.0.1+-event-audit-transfer_balance+335887608146194890</
endPoint>
<type>GET</type>
```

where, *hostname* and *port* is the hostname and port of the machine where REST server is running.

REST Request Parameters

```
{
    "endPoint": "http://hostname:port/bcws/webresources/v1.0/billunits/
balancegroups/transferbalance/0.0.0.1+-event-audit-
transfer_balance+335887608146194890",
    "type": "GET",
    "operation": "GenericBalanceTransferDetails"
}
```

REST API Response

Tags present in the response will not be validated. JSON message will be directly converted to XML.

Sample REST API Response

```
{
"extension":null,
"id":"0.0.0.1+-event-audit-transfer_balance+335887608146194890",
"sourceAccountRef":{
"id":"0.0.0.1+-account+37201",
"uri":"http://hostname:port/bcws/webresources/v1.0/accounts/0.0.0.1+-
account+37201"
},
"targetAccountRef":{
"id":"0.0.0.1+-account+34057",
"uri":"http://hostname:port/bcws/webresources/v1.0/accounts/0.0.0.1+-
account+34057"
},
"transferDate":1649681472000,
"sourceRef":{
"id":"0.0.0.1+-balance_group+40785",
"type":"balanceGroup"
},
"targetRef":{
"id":"0.0.0.1+-balance_group+34953",
"type":"balanceGroup"
},
"transferAmount":11,
"transferAmountType":840,
"chargeSource":false,
"chargeDestination":true,
"sourceImpactedBucket":[{"validFrom":1647993600000,"validTo":0,"amount":11}],
```

```
"targetImpactedBucket":[{"validFrom":1647993600000,"validTo":0,"amount":-11}],
"sourceTransferFee":null,
"targetTransferFee":{
"amount":3,
"feeTax":0.6,
"resourceId":840
}
}
```

XML Response

XML response will have extra tag `<rest_result_code>` containing the status code from REST response header. The status code from RESTClient to DAP will always will be 200. `<rest_status_code>` can be used for checking the status code from REST server.

Sample XML Response

```
<targetRef>
<id>0.0.0.1+-balance_group+34953</id>
<type>balanceGroup</type>
</targetRef>
<extension>null</extension>
<targetImpactedBucket>
<amount>-11</amount>
<validFrom>1647993600000</validFrom>
<validTo>0</validTo>
</targetImpactedBucket>
<chargeSource>false</chargeSource>
<targetTransferFee>
<amount>3</amount>
<resourceId>840</resourceId>
<feeTax>0.6</feeTax>
</targetTransferFee>
<transferAmount>11</transferAmount>
<sourceTransferFee>null</sourceTransferFee>
<targetAccountRef>
<id>0.0.0.1+-account+34057</id>
<uri>http://hostname:port/bcws/webresources/v1.0/accounts/0.0.0.1+-
account+34057</uri>
</targetAccountRef>
<transferAmountType>840</transferAmountType>
<transferDate>1649681472000</transferDate>
<chargeDestination>true</chargeDestination>
<id>0.0.0.1+-event-audit-transfer_balance+335887608146194890</id>
<sourceAccountRef>
<id>0.0.0.1+-account+37201</id>
<uri>http://hostname:port/bcws/webresources/v1.0/accounts/0.0.0.1+-
account+37201</uri>
</sourceAccountRef>
<sourceRef>
<id>0.0.0.1+-balance_group+40785</id>
<type>balanceGroup</type>
</sourceRef>
<sourceImpactedBucket>
<amount>11</amount>
<validFrom>1647993600000</validFrom>
```

```
<validTo>0</validTo>
</sourceImpactedBucket>
<rest_status_code>200</rest_status_code>
```

Part II
REST Server

This part provides the REST Server related information. It contains the following chapters:

7

What is RESTServer?

RESTServer is designed to accept subscriber MSISDN from client applications and in response returns the real time location of the subscriber (as fetched from HLR).

Request Processing

RESTServer accepts the GET request on a specific port that is configured. RESTServer then transforms input JSON request into SOAP request format with subscriber number in the body. RESTServer then forwards the SOAP request to OSD application endpoint.

Response Handling

RESTServer checks whether UserLocation is present in the OSD application response. If present, then UserLocation is sent in JSON response to the client. If any OSD error is reported, it is as well forwarded to the client.

Features

- RESTServer supports GET requests to LocationInfo.

8
Configuration

This chapter explains how to configure the RESTServer.

RESTServer Configuration

RESTServer reads its configuration from the **application.yaml** file. The **application.yaml** file is located in the **/IN/service_packages/REST/etc** directory.

RESTServer application.yaml Section

To organize the configuration data within the config file:

- Some sections are nested within other sections.
- Comments are prefaced with a “#” at the beginning of the line.

Editing the File

Open the configuration file on your system using a standard text editor. Do not use text editors, such as Microsoft Word, that attach control characters. These can be, for example, Microsoft DOS or Windows line termination characters (for example, ^M), which are not visible to the user, at the end of each row. This causes file errors when the application tries to read the configuration file.

Always keep a backup of your file before making any changes to it. This ensures you have a working copy to which you can return.

Loading Config Changes

If you change the configuration file, you must restart the service to enable the new options to take effect.

SSL Configuration

When endpoint is https, SSL configuration is used for certification. You need to configure SSL in the **application.yaml** file under ssl section in server. PKCS12 keystore should also be created.

Create new certificate if required:

```
openssl req -newkey rsa:4096 -x509 -sha256 -days 365 -nodes -out
Certificate.crt -keyout Certifcate.key
```

To import the certificate, run the openssl command as an administrator or root user.

Create txt file from key and certificate:

```
cat Certificate.key Certifcate.crt > Certifcate.pem.txt
```

Create keystore with openssl:

```
openssl pkcs12 -export -in  Certifcate.pem.txt -out keystore.pkcs12
```

For example:

```
cat example.key example.crt > example.pem.txt
openssl pkcs12 -export -in  example.pem.txt -out mykeystore.pkcs12
```

9
Background Processes

This chapter explains the process which runs automatically as part of the Oracle Communications Convergent Charging Controller application. This process is started automatically by the system services (**/IN/bin/OUI_systemctl.sh**) in the SLC node.

RESTServer

Purpose

The RESTServer (REST) interface accepts REST requests for location information.

Startup

This task is started by the system services, by the following line in the service files:

/IN/service_packages/REST/bin/RestServerStartup.sh /IN/service_packages/REST/etc/application.yaml

Configuration

The high-level structure of the RESTServer is shown below:

```
server:
  port: 8080
  tls:
     private-key:
       keystore:
         passphrase: "<keystore-passphrase>"
         resource:
           resource-path: "<keystore-path>"
     protocols:
       - "TLSv1.3"
       - "TLSv1.2"
     cipher-suite :
        - "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
        - "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256"
        - "TLS_AES_128_GCM_SHA256"
        - "TLS_AES_256_GCM_SHA384"


soap :
   endpoint : "http://1.1.2.2:1024"
   username : "<username>"
   password : "<base64password>"
   retryinterval : 1000
   retrycount : 2

security:
  providers:
    - abac:
      # Adds ABAC Provider - it does not require any configuration
```

```
    - oidc:
        validate-with-jwk: false
        client-id: ""
        client-secret: ""
        identity-uri: ""
        realm: ""
        audience: ""
        proxy-host: ""
        redirect: false
        cookie-use: false
        header-use: true
    - oamoidc:
        validate_with_jwk: false
        token-endpoint-uri: "http://oam-endpoint:port/oauth2/rest/token"
        authorization-endpoint-uri: "http://oam-endpoint:port/oauth2/rest/
token"
        introspect-endpoint-uri: "http://oam-endpoint:port/oauth2/rest/token/
info"
        audience: "<audience>"
        cookie-name: "<cookie-name>"
        cookie-same-site: "<cookie-same-site>"
        header-use: true
        query-param-use: true
        redirect: true
        oidc-metadata-well-known: false
        oauth-identity-domain-name: "<domain>"
```

Parameters

Parameters of the RESTServer are listed below.

In **server** block:

port

Syntax:	`port : "value"`
Description:	Port on which REST Server will be listening to requests.
Type:	Integer
Optionality:	Optional
Example:	`port : "8080"`

In **tls > private-key** block:

keystore > resource > resource-path

Syntax:	`resource-path : "value"`
Description:	PKCS12 keystore path. It should be placed in Classpath (/IN/service_packages/REST/etc/).
Type:	String
Example:	`resource-path : “mykeystore.pkcs12"`

keystore > passphrase

Syntax:	`passphrase : "value"`
Description:	Password for the keystore.
Type:	String
Example:	`passphrase : "password"`

In **soap** block:

endpoint

Syntax:	`endpoint : "value"`
Description:	IP address of the OSD where OSDLocation is configured.
Type:	String
Example:	`endpoint : "osd_endpoint:port"`

username

Syntax:	`username : "value"`
Description:	Username of the ASP client for accessing SOAP request.
Type:	String
Example:	`username : "notif"`

password

Syntax:	`password : "value"`
Description:	Base64 encrypted password of ASP client for accessing SOAP request.
Type:	String
Example:	`password : "<base64password>"`

retryinterval

Syntax:	`retryinterval : number`
Description:	Retry interval for SOAP request if failed.
Type:	Integer
Default	1000
Example:	`retryinterval : 1000`

retrycount

Syntax:	`retrycount : number`
Description:	Retry times for SOAP request if failed.
Type:	Integer
Default	0
Example:	`retrycount : 2`

In **oamoidc** block:

validate_with_jwk

Syntax:	`validate_with_jwk : "value"`
Description:	Valid values are: **true**: Validate against jwk defined by "sign-jwk". **false**: Validate JWT through OIDC Server endpoint.
Type:	Boolean
Default	true
Example:	`validate_with_jwk : "false"`

token-endpoint-uri

Syntax:	`token-endpoint-uri : "value"`
Description:	URI of the token endpoint used to obtain the JWT based on the authentication code.
Type:	String
Example:	`token-endpoint-uri : "http://oam-endpoint:port/oauth2/rest/token"`

authorization-endpoint-uri

Syntax:	`authorization-endpoint-uri : "value"`
Description:	URI of an authorization endpoint.
Type:	String
Example:	`authorization-endpoint-uri : "http://oam-endpoint:port/oauth2/rest/token"`

audience

Syntax:	`audience : "value"`
Description:	Audience URI of custom scopes.
Type:	String
Example:	`audience : "<audience>"`

cookie-name

Syntax:	`cookie-name : "value"`
Description:	Name of the cookie.
Type:	String
Default	JSESSIONID
Example:	`cookie-name : "<cookie-name>"`

cookie-same-site

Syntax:	`cookie-same-site : "value"`
Description:	Used to set the SameSite cookie value when using cookie. The value can be **Strict** or **Lax**. Setting this to **Strict** will result in infinite redirects when calling OIDC on a different host.

Type:	String
Default	Lax
Example:	`cookie-same-site : "<cookie-same-site>"`

header-use

Syntax:	`header-use : "value"`
Description:	Whether to expect JWT in the header field.
Type:	Boolean
Default	false
Example:	`header-use : "true"`

query-param-use

Syntax:	`query-param-use : "value"`
Description:	Whether to expect JWT in the query parameter.
Type:	Boolean
Default	false
Example:	`query-param-use : "true"`

redirect

Syntax:	`redirect : "value"`
Description:	Whether to redirect to identity server when authentication failed.
Type:	Boolean
Default	true
Example:	`redirect : "true"`

oidc-metadata-well-known

Syntax:	`oidc-metadata-well-known : "value"`
Description:	If set to true, metadata will be loaded from default (well known) location, unless it is explicitly defined using oidc-metadata-resource. If set to false, it would not be loaded even if oidc-metadata-resource is not defined. In such cases, all URIs must be explicitly defined (Example: token-endpoint-uri).
Type:	Boolean
Default	true
Example:	`oidc-metadata-well-known : "false"`

oauth-identity-domain-name

Syntax:	`oauth-identity-domain-name : "value"`
Description:	This is used to access token from OAM server. `oauth-identity-domain-name` is added to the header “X-OAUTH-IDENTITY-DOMAIN-NAME” in the access token request sent to OAM.
Type:	String
Example:	`oauth-identity-domain-name : "<domain>"`

For **security > providers > oidc** block, check the following link for information on configurations:

https://helidon.io/docs/v1/apidocs/index.html?io/helidon/security/providers/oidc/common/OidcConfig.html

10

About Installation

This chapter provides information about the installed components for the Oracle Communications Convergent Charging Controller application described in this guide. It also lists the files installed by the application that you can check for, to ensure that the application is installed successfully.

Installation Overview

For information about the following requirements and tasks, see Installation Guide:

- Convergent Charging Controller system requirements
- Pre-installation tasks
- Installing and removing Convergent Charging Controller packages

RESTServer Package

Installation of Oracle Communications Convergent Charging Controller RESTServer includes **restScp** package on SLC.

Checking the Installation

Refer to the following checklist to ensure that RESTServer is installed correctly.

Checklist - SLC

Follow the steps in this checklist to ensure RESTServer is installed correctly on an SLC machine.

1. Log in to the SLC machine as root.
2. Check that the following directory structure exists, with subdirectories:

   /IN/service_packages/REST
3. Check that directories contain subdirectories and that all are owned by:

   smf_oper user (group esg)

Process list - SLC

If the application is running correctly, **RESTServer.jar** process should be running on each SLC, started during OUI_systemctl startup.

RESTServer Directories and Files

The RESTServer installation on SLC creates the following directories:

- **/IN/services_packages/REST/bin**
- **/IN/services_packages/REST/lib**
- **/IN/services_packages/REST/etc**

- **/IN/services_packages/REST/tmp**

Installing RESTServer installs the following interface:

/IN/services_packages/REST/bin/RESTServer.jar

Installing RESTServer installs the following configuration file:

/IN/services_packages/REST/etc/application.yaml.example

> **Note**
>
> You need to create the **application.yaml** file with actual values required during runtime.

11

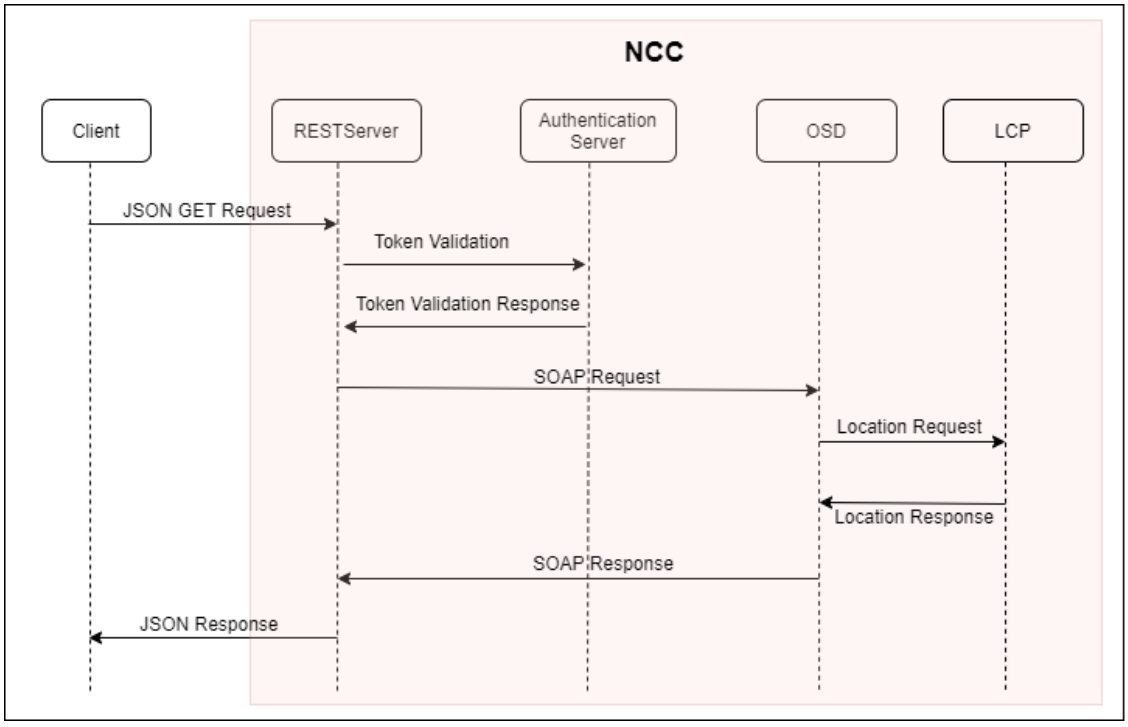
RESTServer Call Flows

This chapter provides a sample REST request flow.

Request Flow

RestServer gets the token and the subscriber number from the request. Based on the configuration, it checks the validity of the token. If the token is valid, then it creates a SOAP request with body containing the subscriber number. SOAP request is then sent to the configured OSD endpoint. Based on the response from OSD, a JSON response is created and sent back to the client.

Figure 11-1 RESTServer Call Flow



12

Supported Endpoints

RESTServer supports the following request:

- GET Location

  https://*hostname:port/LocationInfo/{service-number}*

GET Location

To request a Location, use cURL to send an HTTP/HTTPS request to the REST location endpoint.

```
curl -i
     -H "Authorization: Bearer "
     -H "Content-Type: application/json"
     -H "Accept: application/json"
     -X GET https://hostname:port/LocationInfo/{service-number}
```

Supported Method: GET

Request

Path Parameter

- service-number(required): string

There's no request body for this operation.

Response

Supported Media Types: application/json

1. 200 Response

   Success

2. 400 Response

   The request isn't valid.

3. 500 Response

   An internal server error occurred.

Examples

Request:

```
curl -X -H "Authorization: Bearer accessToken"
 GET 'http://hostname:port/LocationInfo/257531'
```

Response:

```
Success Case:
{
"Location": "52121005DC03EA"
}

Error Case:
{
"error": "Unable to get subscriber Location"
}
```

Note

Decoding Location: 52121005DC03EA

- MCC = 521
- MNC = 210
- LAC = 05DC
- CID = 03EA

13

Configuring OAuth Services

To use OAuth 2.0 for authentication, configure your REST API services and then register your client application as a trusted client on Oracle Access Manager.

To configure OAuth services using Oracle Access Manager:

1. Create an identity domain, which controls the authentication and authorization of your client applications. It also controls which features your client application can access in relation to the service.

   To create an identity domain, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL. For example:

   ```
   curl -i
        -H "Content-Type: application/json"
        -H "Accept: application/json"
        -H "Authorization:Basic credentials"
        -X POST
        http(s)://hostname:port/oam/services/rest/ssa/api/v1/oauthpolicyadmin/
   oauthidentitydomain
        -d '{
            "name": "identityDomain",
            "description": "Description for Convergent Charging
               Controller REST API Identity Domain",
            "tokenSettings":[
                {
                    "tokenType": "ACCESS_TOKEN",
                    "tokenExpiry": tokenExpiry
                }
            ]
        }'
   ```

   where:

   - *credentials* is the Base64-encoded value of your Oracle Access Manager administrator user name and password joined by a single colon (username:password).
   - *hostname:port* is the host and port of the Oracle Access Manager Administration Server.
   - *identityDomain* is the name of the Oracle Access Manager identity domain that you want to create.
   - *tokenExpiry* is the number of seconds before the token expires, such as 3600 for one hour.

   For more information, see "Add a new OAuth Identity Domain" in *REST API for OAuth in Oracle Access Manager*.

2. Create a resource server, which hosts protected resources and accepts and responds to protected resource requests using access tokens.

To create and configure your resource server, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL. For example:

```
curl -i
     -H "Content-Type: application/json"
     -H "Authorization:Basic credentials"
     -X POST
     http(s)://hostname:port/oam/services/rest/ssa/api/v1/oauthpolicyadmin/
application
     -d '{
         "name": "resourceServer",
         "idDomain": "identityDomain",
         "description": "Convergent Charging
            Controller REST API Resource Server",
         "scopes":[
             {
                 "scopeName":"scopeName",
                 "description":"All Access"
             }
         ]
     }'
```

where:

- *resourceServer* is the name of your resource server, such as Convergent Charging Controller.
- *scopeName* is the name of the scope, such as All.

For more information, see "Add a new Resource Server" in *REST API for OAuth in Oracle Access Manager*.

3. Create a client application that makes protected resource requests on behalf of the resource owner and with the resource owner's authorization. Convergent Charging Controller REST API clients are web applications with an OAuth 2.0 client type of Confidential Client. Clients must use a grant type of Client Credentials for requesting access to Convergent Charging Controller REST API resources.

   To create a client application, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL. For example:

```
curl -i
     -H "Content-Type:application/json"
     -H "Authorization:Basic credentials"
     -X POST
     http(s)://hostname:port/oam/services/rest/ssa/api/v1/oauthpolicyadmin/
client
     -d '{
         "secret": "client_secret",
         "id": "client_id",
         "name": "clientName",
         "scopes": [
             "resourceServer.scopeName"
         ],
         "clientType": "CONFIDENTIAL_CLIENT",
         "idDomain": "identityDomain",
         "description": "Description of client of Convergent Charging
            Controller REST API Server",
```

```
        "grantTypes":[
            "CLIENT_CREDENTIALS"
        ],
        "defaultScope": "resourceServer.scopeName",
        "redirectURIs": [
            {
                "url":"http(s)://Convergent Charging
           ControllerHost:Convergent Charging
           ControllerPort",
                "isHttps": isHttps
            }
        ]
    }'
```

where:

- *client_secret* is the password for your client.
- *client_id* is the client ID for your client. It will be generated automatically if not specified.
- *clientName* is the name of your client.
- *Convergent Charging ControllerHost:Convergent Charging ControllerPort* is the host and port of the Convergent Charging Controller REST API Server.
- *isHttps* is a Boolean value that specifies whether the URL is accessed over HTTPS (true) or HTTP (false).

For more information, see "Add a new OAuth Client" in *REST API for OAuth in Oracle Access Manager*.

For more information about OAuth 2.0, see "Understanding the OAuth Service" in *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*.

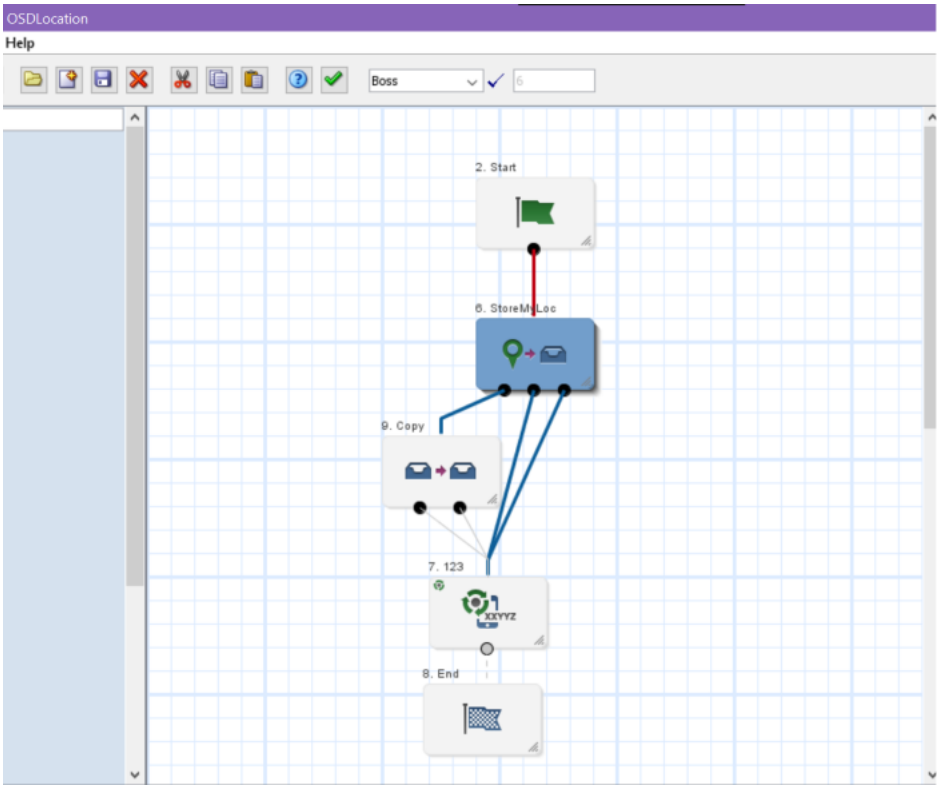
14

Configuring SOAP Services

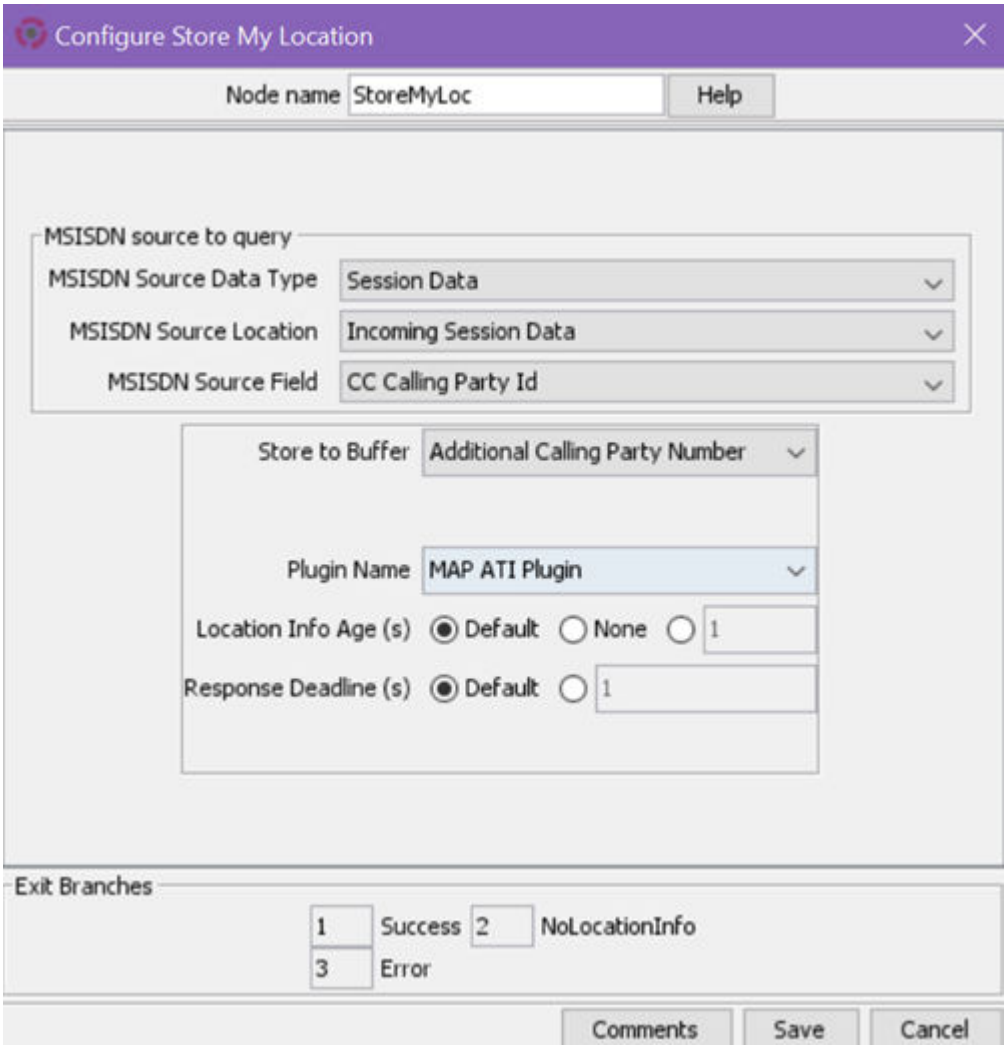
RESTServer sends SOAP request to OSD client present in the SLC machine.

Operations and **Operation Sets** used by RESTServer is **OSDLocation**. The following are the mandatory configurations that should be present in the **Open Services Development** (OSD) screen:

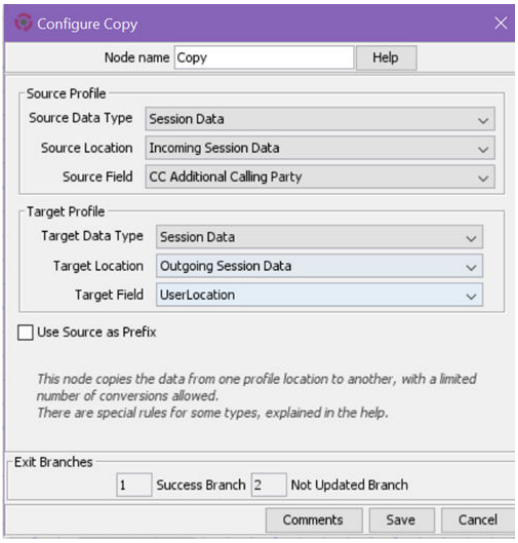
1. Configure OSDLocation in **Operations** and **Operation Sets** tab in the **Open Services Development** screen.
2. Create Control plan with **StoreMyLocation** and **Copy** node and link it to the OSDLocation operation.



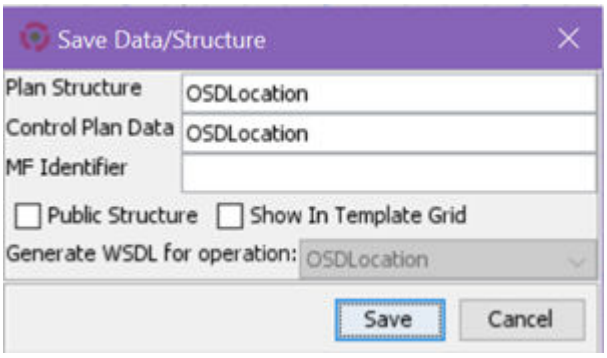
   a. In **StoreMyLocation** node, set **CC Calling Party id** (Incoming Session Data) in **MSISDN Source Field**. Location is copied to the profile tag set in **Store to Buffer** field. For example, Additional Calling Party Number.



b. In Copy node, set **UseLocation** (Outgoing Session Data) in **Target Field**. Copy node copies Location from buffer to profile tag UserLocation . If UserLocation profile tag is not present, then UserLocation profile tag needs to be created and mapped to the Outgoing Session Data.



c. Save the control plan to link it with OSDLocation operation.



3. Configure the RESTServer hostname with username and password in **Client ASPs** tab in the **Open Services Development** screen. OSDLocation operation should be added to this client ASP.