Oracle® Communications Services Gatekeeper

OAuth Guide

Release 5.0.0.1

E37518-01

October 2012

ORACLE®

Oracle Communications Services Gatekeeper OAuth Guide, Release 5.0.0.1

E37518-01

Contents

Preface

This book provides the OAuth Guide for Oracle® Communications Services Gatekeeper 5.0.0.1.

Audience

This document is intended for anyone who needs an understanding of the contents contained in this Oracle Communications Services Gatekeeper release.

This includes:

- System administrators charged with installing and maintaining Oracle Communications Services Gatekeeper
- Third-party application developers who wish to integrate telephony-based functionality into their products
- Operator-based system developers who wish to extend the functionality of Oracle Communications Services Gatekeeper or to integrate it with Partner Relationship Management (PRM) or Operations Support Systems (OSS) tools
- Managers, support engineers, and sales and marketing personnel for network operators and service providers

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 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.

Related Documents

For more information, see the following documents in the Oracle Communications Services Gatekeeper set and patch documentation:

- *Accounts and SLAs Guide*
- *Alarm Handling Guide*
- *Application Developer's Guide*

- *Communication Service Guide*
- *Concepts Guide*
- *Deployment Guide*
- *Installation Guide*
- *Java API Reference*
- *Licensing Guide*
- *OAM Java API Reference*
- *One API Application Developer's Guide*
- *Partner Relationship Management Guide*
- *Platform Development Studio Developer's Guide*
- *Platform Test Environment Guide*
- *Release Notes*
- *RESTful Application Development Guide*
- *SDK User's Guide*
- *Statement of Compliance*
- *System Administrator's Guide*
- *System Backup and Restore Guide*

1

OAuth and Services Gatekeeper

This chapter describes the implementation and use of Open Authorization Protocol (OAuth) in Oracle Communications Services Gatekeeper.

About Services Gatekeeper Support for OAuth Authentication Server

The Services Gatekeeper OAuth implementation allows service providers to offer authorized third-party application access to protected Resource owner resources.

Full details of the OAuth specification are available at: http://tools.ietf.org/html/draft-ietf-oauth-v2-25.

OAuth 2.0 Concepts

OAuth is an open standard for authorization. It allows users (subscribers) to share their private resources (for example, photos, videos, contact lists, location and billing capability) stored on one site with another site without having to hand out their credentials, typically a username and password stored with their service provider.

Client applications wishing to access protected resources are granted a token once access is approved by the Resource owner. Each token grants access to a specific site or an application (for example, a video editing site) for a specific resource (for example, location of the user) and for a defined duration (for example, the next two hours). This allows a Resource owner to grant a third party site access to their information stored with another service provider, without sharing their access permissions or the full extent of their data

Terminology

Table 1–1 lists OAuth terminology and definitions.

Table 1–1 OAuth Terminology and Definitions

Term	Definition
OAuth	Open Authorization Protocol
resource	The Web resource protected by OAuth Protocol
Resource owner	An entity capable of granting access to a protected resource. In the operator context this is defined as Resource owner URI (tel: or sip:)

Table 1–1 (Cont.) OAuth Terminology and Definitions

Term	Definition
Application Client	An application making protected resource requests on behalf of the Resource owner and with the Resource owner's authorization. The term client does not imply any particular implementation characteristics (for example, whether the application executes on a server, a desktop, or other devices).
scopeId	Unique string that identifies a resource and used as part of scope-token by application client.
Protocol Endpoint	Network URI representing the location of a service to: ■ obtain an authorization code and other values ■ obtain an access token ■ submit a grant. ■ access a resource
Client Identifier	A unique string representing the registration information provided by the client.
Authorization Grant	Represents the authorization given by the Resource owner to a client application. An Authorization Grant is a credential representing the resource owner's authorization (to access its protected resources) used by the client to obtain an access token.
Access Token	Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client.
Refresh Token	Refresh tokens are credentials used to re-obtain access tokens.
Authorization Server	Server that issues authorization codes and access token.
Resource Server	Server that hosts protected resources and validates access token during resource access.
Authorization Endpoint	Used to obtain authorization from the Resource owner via user-agent redirection.
Grant Endpoint	URI supported by Services Gatekeeper to post the authentication result to issue the authorization code (defined by Services Gatekeeper).
Redirection Endpoint	After completing its interaction with the Resource owner, the Authorization Server directs the Resource owner's user-agent back to the client. The Authorization Server redirects the user-agent to the client's redirection endpoint previously established with the Authorization Server during the client registration process or when making the authorization request.
Authentication Server	Server that validates Resource owner identity (defined by Services Gatekeeper).
Delegated Authentication	Authentication mode supported by Services Gatekeeper to integrate with 3rd party authentication systems.
Subscriber Manager	Component in Services Gatekeeper to validate subscribers provisioned in Services Gatekeeper database.
Custom Subscriber Manager	Component that authenticates Resource owner's username and password with a custom identity store (such as LDAP).
Group URI	URI that represents a group of Resource Owners.
Group Owner	Owner of the Group URI. Issues authorization token on behalf of the group members.

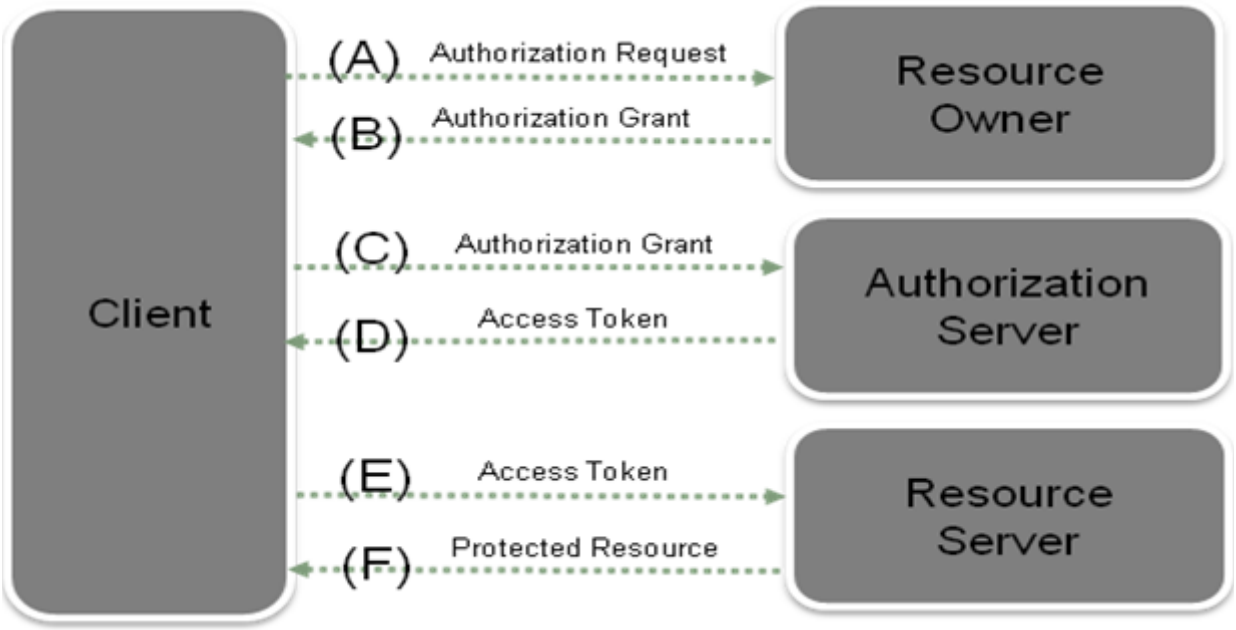
Table 1–1 (Cont.) OAuth Terminology and Definitions

Term	Definition
applicationInstanceID	String that uniquely identifies the ApplicationInstance. One applicationInstanceId can be mapped with one OAuth2 Client Identifier, so that SLA can be proceed for OAuth2 based traffic.

Entities and Relationships

The OAuth protocol flow can be illustrated through the following steps illustrated in Figure 1–1:

Figure 1–1 OAuth Flow and Entity Relationships



(A) The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or preferably indirectly via the Authorization Server as an intermediary.
(B) The client receives an authorization grant which is a credential representing the resource owner's authorization, expressed using one of four grant types defined in this specification or using an Extension grant type. The authorization grant type depends on the method used by the client to request authorization and the types supported by the Authorization Server.
(C) The client requests an access token by authenticating with the Authorization Server and presenting the authorization grant.
(D) The Authorization Server authenticates the client and validates the authorization grant, and if valid issues an access token.
(E) The client requests the protected resource from the resource server and authenticates by resenting the access token.
(F) The resource server validates the access token, and if valid, serves the request.

Protocol Endpoints

The OAuth2.0 specification defines three types of protocol endpoints.

- **Redirection Endpoint**: The redirection endpoint is a URI used by Authorization Server to return authorization credentials responses from the Authorization Server to the client via the Resource owner user-agent.
- **Authorization Endpoint**: The authorization endpoint is used to interact with the Resource owner (typically the subscriber) and obtain an Authorization Grant which will be issued to an application client by the Resource owner. The Authorization Server must first verify the identity of the Resource owner before granting the Authorization Grant. This Authorization Grant will be exchanged by the application client for an access token.
- **Token Endpoint**: The token endpoint is used by the client to obtain an access token by presenting its Authorization Grant (the authorization code) or refresh token. The token endpoint is used with every Authorization Grant except for the implicit grant type (since an access token is issued directly).

Services Gatekeeper Mapping of OAuth

This section explains the Services Gatekeeper functionality mapped to the OAuth specification.

OAuth Terminology Mapping to Services Gatekeeper

- **resource**: Defined by the OAuth2.0 specification. In Services Gatekeeper, this is mapped to API(s) and methods protected through the OAuth token resource is mapped as a combination of Communication Service northbound interface (Plug-in) name, method name, token expire period, parameters and subResource. resource is uniquely identified by scopeId.

  For more information, see "Resource Mapping".

- **scope**: Defined by the OAuth2.0 specification. Determines the boundary of an OAuth token. The scope parameter is submitted as part of obtaining authorization grant.

  The format of the scope is defined in the OAuth 2.0 specification as:

  ```
  scope = scope-token *(SP scope-token)
  ```

  ```
  scope-token = 1*(%x21 / %x23-5B / %x5D-7E)
  ```

  Services Gatekeeper maps the scope-token parameter to:

  ```
  <scopeId>[?<param>=<value>[&<param>=<value>]*]+
  ```

  The scopeId identifies the resource and param represents the custom parameters that are defined as part of resource.

  Parameter values submitted as part of the scope can be interpreted by custom interceptors.

  For example:

  ```
  scope=chargeAmount?MaxAmount=5&itemId=123SPgetLocation?Accuracy=5
  where
  SP=blank space
  ```

- **scopeId**: A unique identifier that represents an OAuth resource during resource mapping and determines the access scope of a resource during an authorization grant.

Defined as part of mapping a Communication Service method as an OAuth resource, scopeId is used as the value of the 'id' attribute of the resource tag. Each resource is indicated as owned by a Resource owner by creating an association between the Resource owner (subscriber URI) and the scopeId. The scopeId is submitted as part of scope-token parameter as part of authorization grant request.

For example:

1. As part of resource configuration, the `oracle.Services Gatekeeper.parlayrest.plugin.PaymentPlugin` (Communication Service) method amountTransaction is created as a resource with scopeId chargeAmount:

```
<resource id="chargeAmount" name="Charge or refund"
interfaceName="oracle.Services Gatekeeper.parlayrest.plugin.PaymentPlugin"
methodName="amountTransaction"
tokenExpirePeriod="3600">
<parameter name="code" description="billable item id"/>
</resource>
```

2. The chargeAmount resource is mapped to tel:1234 as the resourceOwner.
3. During the authorization grant, the application sends `scope=chargeAmount` as part of the authorization request.

- **Resource owner**: Defined by the OAuth2.0 specification. Represented as the target address used in the REST API (Communication Service), and typically represented as one MSISDN. The MSISDN serves as the connection between the resource defined during configuration time and the resource protected during resource authorization and access time.
  - The Resource owner (MSISDN) and the scopeId (collection of resources) are mapped during configuration time.
  - Depending on the grant type, the OAuth authorization code/access token is issued by the Resource owner (MSISDN).
- **subResource**: A subResource enables protection and authorization of multiple resources (API methods) with a single token. Resources can have one or more subResources defined as part of resource definition. Authorization Grants to a resource also apply to its subResources.

  For example, for a payment Communication Service, a resource called **amountTransaction** can be created for charging transactions. A subResource called **checkTransactionStatus** is a method used to query the status of a charging transaction. When a Resource owner issues a token for resource **amountTransaction**, it can be automatically used against the **checkTransactionStatus** subResource as well.
- **Application Client:** You can map one Services Gatekeeper Application Instance Id with one OAuth application client by configuring an SLA according to the associated application group and service group.

Services Gatekeeper relies on the authentication endpoint to validate the Resource owner. In the Services Gatekeeper default implementation of the authentication endpoint, the validation will be handled by the Subscriber Manager. See "Resource Owner - Resource Mapping" for more information.

Endpoints Mapping

Service Gatekeeper supports two standard endpoints defined by the OAuth specification.

The following describes the OAuth endpoint mapping in Services Gatekeeper:

- **Redirection endpoint**: Provided by an application client during the Authorization Grant.
- **Authorization endpoint**: By default, the authorization endpoint is configured to the following URL:

  `https://%AT_HOST%:%PORT%/oauth2/authorization`

- **Token endpoint**: By default, the token endpoint is configured to the following URL:

  `https://%AT_HOST%:%PORT%/oauth2/token`

In addition to the standard endpoints, Services Gatekeeper supports two custom endpoints to facilitate integration with external/custom Authentication Servers:

- **Authentication Endpoint:** Verifies the identity of a Resource owner. The authentication endpoint is an extension point offered by Services Gatekeeper handling Resource owner authentication and Authorization Grant collection.

  This endpoint interacts with a Resource owner, assuring that the subscriber's identity is valid and providing necessary information for the Resource owner to authorize an application to obtain an Authorization Grant.

  By default, the authentication endpoint uses the following URL:

  `https://%AT_HOST%:%PORT%/oauth2/auth.jsp`

  This URL can be updated by editing the following MBean property:

  OauthService.OauthCommonMBean.AuthenticationURL

- **Grant Endpoint:** The Call back URI supported by Services Gatekeeper to process the successfully authenticated requests and issue Authorization Grant to application clients.

  After a Resource owner grants access to protected resources through authentication, the authorization request is submitted to the grant endpoint, which sends the Resource owner to a redirect URI provided by the application along with the authorization code.

  By default, the grant endpoint uses the following URL:

  `https://%AT_HOST%:%PORT%/oauth2/grant`

  This URL can be updated by editing the following MBean property:

  OauthService.OauthCommonMBean.GrantURL

For more information about the interactions between these endpoints, see "Delegated Authentication"

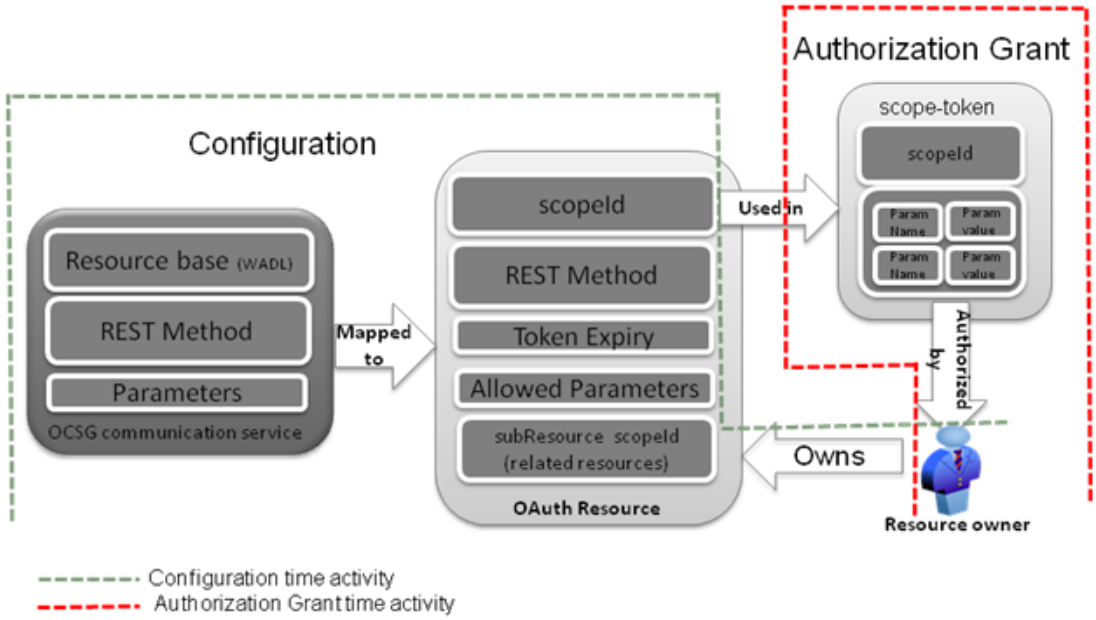
Scope-resource-Communication Service Mapping

OAuth2.0 token based security can be used to protect any RESTful Communication Service (CS), including a customized CS method, as long as the specified method is configured as a protected resource.

Figure 1–2 illustrates the relationship between:

- Communication Service and an OAuth resource
- resource and Resource owner
- scope and the resource

Figure 1–2 OAuth Service and Resource Entities Relationships



The Communication Service to resource mapping is explained in detail in the "Resource Management" section.

The scope (defined in the OAuth specification) consists of one or more scope-tokens. Each scope-token contains a scopeId (which identifies the resource) and a list of parameter name-values pairs associated with this scopeId. The scope is submitted as part of the authorization grant request. The scopeId, submitted as part of the scope-token, is interpreted and enforced by the default Services Gatekeeper OAuth interceptor. The scope-token parameter values can be interpreted by a custom interceptor to enforce the scope during resource access time.

More details on the scope can be found in the "OAuth Terminology Mapping to Services Gatekeeper" section.

OAuth Protection for APIs Involving Multiple Resource owners

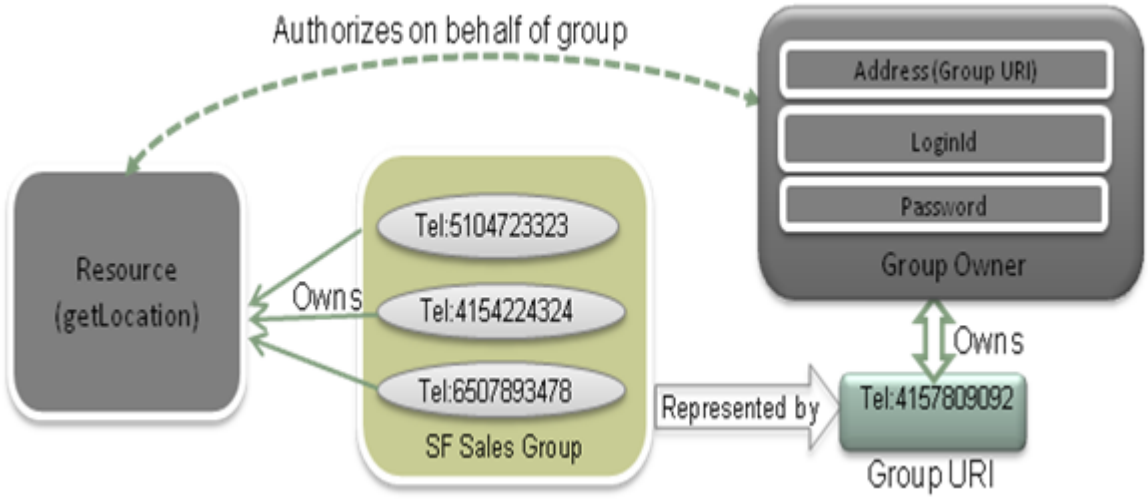
Some APIs require authorization from multiple resource owners. For example:

A Sales Manager may want to expose the location of his sales associates to a location tracking application. This use case requires using the getGroupLocation method in the Location API with multiple Resource owner addresses.

With Services Gatekeeper, it is possible to create a group of resource owners and associated them with a group URI. A group owner can be created to issue an Authorization Grant on behalf of the members that are part of the group URI. Similar to the resource owner, the group owner is represented by an address (Group URI), loginId and password. The Authorization Grant issued by a group owner can be easily used with APIs that accept the URIs of group members as one of the request parameters.

The relationship between resources, Resource owner(s), Groups, Group URI and Group Owner is illustrated in Figure 1–3:

Figure 1–3 Resource and Group Relationships



Though the above diagram uses tel: URIs, it is possible to use sip: URIs as well.

Protecting an API that takes group URIs consists of the following steps:

1. Creating a group with a group URI using the Parlay X 3.0 AddressListManagement.
2. Adding members to the group using the Parlay X 3.0 AddressListManagement.
3. Creating a Group owner related to this Group URI and password for the Group owner.
4. Communicating the group URI and password to a group owner (outside the scope of Services Gatekeeper).
5. The group owner issuing an Authorization Grant to an application to use the member URI as part of API method that requires multiple URIs.
6. Accessing a resource (invoking an API method) that accepts multiple resource owners as method parameters.

The default Subscriber Manager supports a mechanism to provision the group owner represented by the group URI. Services Gatekeeper supports this requirement by using the Parlay X 3.0 AddressList Management Communication Service to create groups. See the following specification for more information about address list management: http://www.3gpp.org/ftp/specs/archive/29_series/29.199-13/29199-13-702.zip.

Compliance

This section describes Services Gatekeeper compliance with the OAuth 2.0 specification.

Supported Communication Services

The OAuth2.0 security mechanism can only be enforced for the REST APIs deployed on Services Gatekeeper. It is not possible to protect the SOAP APIs through OAuth

token enforcement. By default, Services Gatekeeper can protect OneAPI (REST) SMS, MMS, Location, Payment communication services. It is also possible to protect custom REST communication services deployed on Services Gatekeeper.

For more information on supported Communication Services, see the *RESTful Application Developer's Guide*.

Supported OAuth Server Roles

By default, Services Gatekeeper supports the following OAuth server roles:

- Authorization Server: issuing Authorization Grant and access tokens
- Resource Server: protecting resources and enforcing OAuth token-based access to resources.

Supported Authorization Grant Types

An Authorization Grant is a credential representing the Resource owner's authorization (to access its protected resources) used by the client to obtain an access token.

By default, Services Gatekeeper supports the following grant types:

- Authorization Code Grant
- Implicit Grant

Extension Grant Flows Enabled Through Supported Grant Types

Services Gatekeeper facilitates customization of the authorization flow by supporting two custom Endpoints described in "Endpoints Mapping".

Instead of authenticating with the default Subscriber Manager, it is possible to define a custom authentication URL that authenticates and obtains user's consent and posts the authorization result back to Services Gatekeeper to issue the authorization code.

Delegated authentication is only supported with the Authorization code grant type.

See "Delegated Authentication" for more information.

Supported Token Types

Services Gatekeeper supports the following token types defined in OAuth2 specification:

- **Bearer**
- **MAC**

Supported Client Profiles

Services Gatekeeper supports Web based, user-agent-based, and native application profiles.

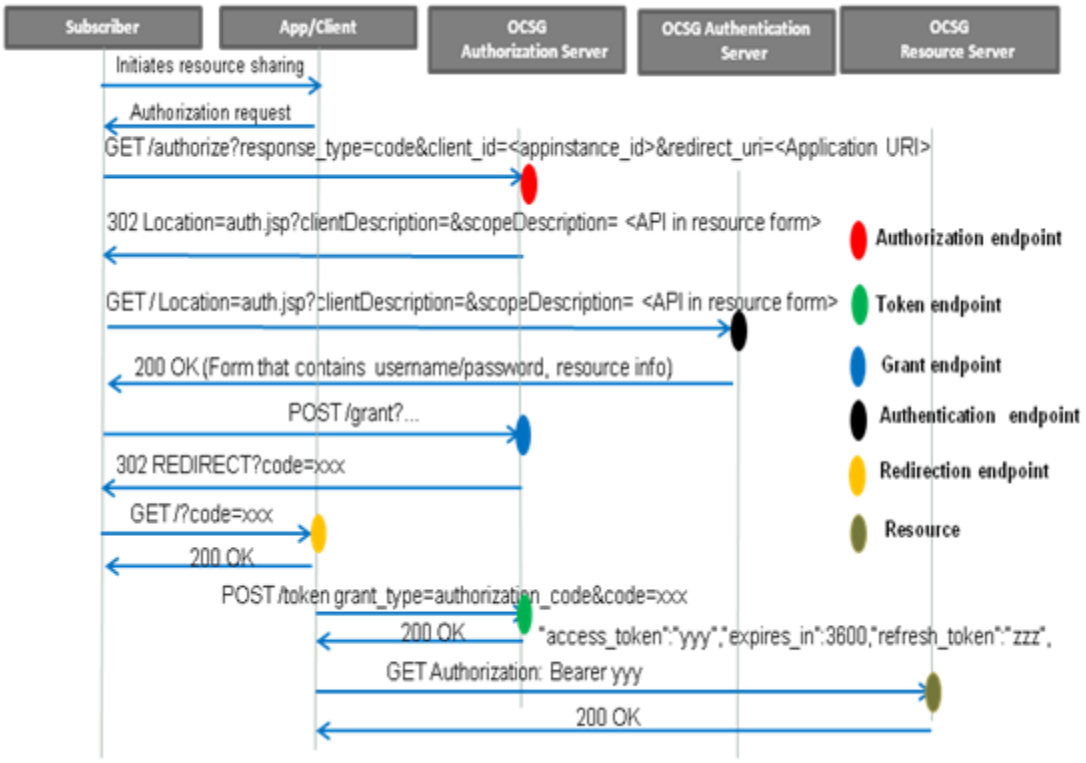
OAuth Flows Supported by Services Gatekeeper

Services Gatekeeper supports the Authorization Code Grant and the Implicit Grant.

Authorization Code Grant

A sequence flow diagram with an OAuth authorization code grant is depicted in Figure 1–4.

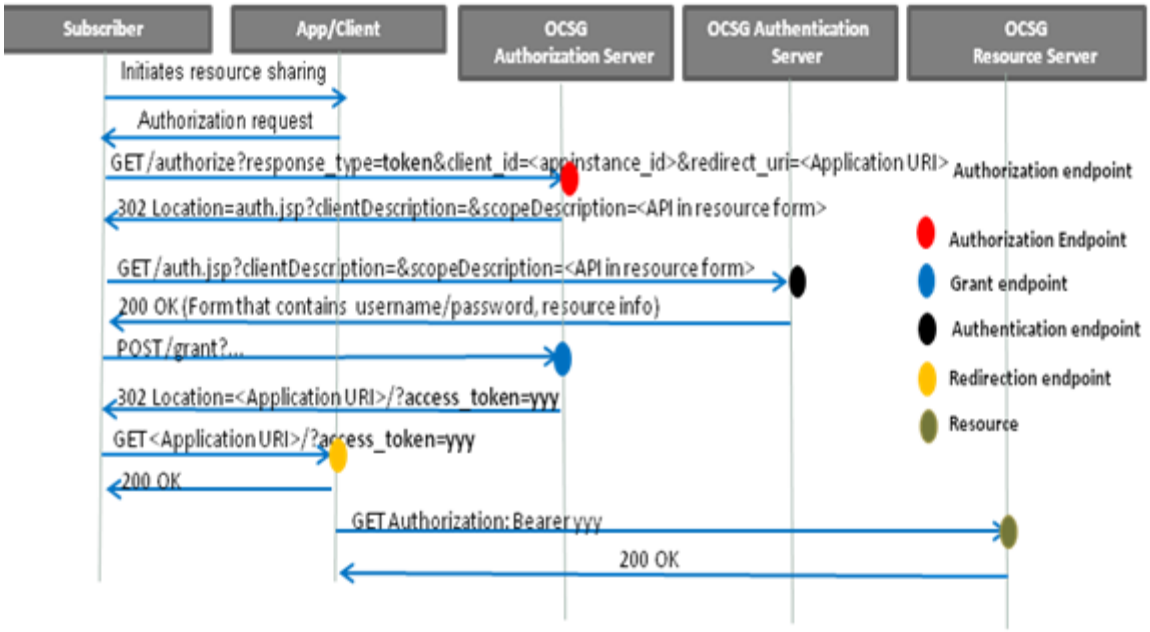
Figure 1–4 Authorization Code Grant



Implicit Grant

A sequence flow diagram with an OAuth implicit grant is depicted in Figure 1–5.

Figure 1–5 Implicit Grant



Supported URIs (Subscribers)

Services Gatekeeper supports the following two Resource owner URIs for use with OAuth:

- tel: URI (as described in RFC 2806)
- sip: URI (as described in RFC3261)

Resource Management

This section describes OAuth resource management as supported by Services Gatekeeper.

Resource Mapping

Services Gatekeeper can be used to protect Communication Services using OAuth by mapping the communication services as OAuth resources.

In order to protect an API with OAuth, a Communication Service (API) and method can be modeled as an OAuth resource. Multiple APIs can be protected through OAuth by creating multiple resources. Resource mapping involves creating a unique scopeId, representing the resource, and mapping the scopeId to API name and method name.

As part of the resource definition, additional context parameters (which may or may not map directly to the API method parameters) can be defined. During resource access time, these parameters can be interpreted using a custom interceptor.

Services Gatekeeper allows the flexibility to create multiple resources for one Communication Service. Resources are aggregated and defined as XML elements in a single XML file. The scopeId defined in this XML file is used as part of the scope submitted during an authorization grant request. Created resources need to be mapped to the resource owners (identified by subscriber tel:/sip: URIs). For the

scopeIds (resources) owned by Resource owner, the Resource owner can issue authorization grants to the application clients.

The Services Gatekeeper OAuth resource format is defined in the **oauth_resource.xsd** file located in the **oauth2_nt.ear file**. To view the resource schema definition use an archive manager to expand the **wlng_nt_oauth2.ear** found in:

$Middleware_Home/ocsg_5.0/applications

Here are the contents of the oauth_resource.xsd file:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://oracle/ocsg/oauth2/management/xml"
    xmlns:tns="http://oracle/ocsg/oauth2/management/xml"
elementFormDefault="qualified">
    <element name="resources">
        <complexType>
            <complexContent>
                <extension base="tns:OAuthResources">
                </extension>
            </complexContent>
        </complexType>
    </element>
    <complexType name="ResourceParameter">
        <annotation>
            <documentation>
                Parameter of resource.
            </documentation>
        </annotation>
        <attribute name="name" type="string" use="required">
            <annotation>
                <documentation>
                    Parameter name.
                </documentation>
            </annotation>
        </attribute>
        <attribute name="description" type="string" use="required">
            <annotation>
                <documentation>
                    Parameter description.
                </documentation>
            </annotation>
        </attribute>
    </complexType>
    <complexType name="OAuthResources">
        <annotation>
            <documentation>
                All supported OAuth2.0 resources in the resource server of OCSG.
            </documentation>
        </annotation>
        <sequence>
            <element name="resource" type="tns:OAuthResource" minOccurs="0"
maxOccurs="unbounded"></element>
        </sequence>
    </complexType>
    <complexType name="OAuthResource">
        <annotation>
            <documentation>
                Define a OAuth2.0 resource.
            </documentation>
```

```
        </annotation>
        <sequence>
            <element name="parameter" type="tns:ResourceParameter" minOccurs="0"
maxOccurs="unbounded"></element>
            <element name="subResource" type="string" minOccurs="0"
maxOccurs="unbounded"></element>
        </sequence>
        <attribute name="id" type="string" use="required">
            <annotation>
                <documentation>
                    Resource identifier.
                </documentation>
            </annotation>
        </attribute>
        <attribute name="name" type="string" use="required">
            <annotation>
                <documentation>
                    Resource name.
                </documentation>
            </annotation>
        </attribute>
        <attribute name="interfaceName" type="string" use="required">
            <annotation>
                <documentation>
                    Plug-in north interface name of the resource.
                </documentation>
            </annotation>
        </attribute>
        <attribute name="methodName" type="string" use="required">
            <annotation>
                <documentation>
                    Plug-in north method name of the resource.
                </documentation>
            </annotation>
        </attribute>
        <attribute name="tokenExpirePeriod" type="int" use="optional"
default="3600">
            <annotation>
                <documentation>
                    Token expire period by seconds.
                </documentation>
            </annotation>
        </attribute>
    </complexType>
</schema>
```

Table 1–2 lists the resource structure and attributes.

Table 1–2 Resource Structure and Attributes

Attributes	Type	Description
Id	String	Unique identifier for the resource scope (required). As part of an authorization grant, the Id (as the scopeId), is submitted as part of the scope-token parameter value.
Name	String	Resource name (required). A concise description of a resource which can be used for display purposes.
InterfaceName	String	Plug-in north interface name of the resource (required).
MethodName	String	Plug-in north method name of the resource (required).

Table 1–2 (Cont.) Resource Structure and Attributes

Attributes	Type	Description
TokenExpirePeriod	Int	Number of seconds until a token expires (optional). If multiple resources (scopes) are granted with a single token, the earliest token expiration period will be enforced on the token. If the resource has subResources, then the earliest token expiration period configured among all resources will be used.
subResource	String	One or more resources that can exist within the scope of the resource (optional). The value of this field should be an id of another resource.
Parameter	ResourceParameter	One or more parameters valid for the resource (optional). These parameter(s) are submitted as part of the OAuth authorization grant. During an authorization grant, a resource may accept several parameters, and each of the parameters can have two attributes, name and description. Parameters defined as part of the resource do not need to be directly related to the method parameters of an API. The semantics of the parameters can be interpreted by a custom interceptor by examining the RequestContext. For example, for the following scope value: `chargeAmount?code=123` The chargeAmount is the scopeId mapped in Services Gatekeeper, the code represents the parameter name and 123 represents the parameter value. As part of communication service access (resource access), a custom interceptor can be written to interpret the OAuth token scope and RequestContext and validate the token usage against the authorization scope.

Resource Mapping Example

The following is an example XML representation of the OAuth resource mapping for a OneAPI Communication Service:

```
<?xml version="1.0" encoding="UTF-8"?>
<resources xmlns="http://oracle/Services Gatekeeper/OAuth2.0 /management/xml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- amountTransaction  -->
  <resource id="chargeAmount" name="Charge or refund"
  interfaceName="oracle.Services Gatekeeper.parlayrest.plugin.PaymentPlugin"
methodName="amountTransaction"
  tokenExpirePeriod="3600">
  <parameter name="code" description="billable item id"/>
  <subResource>checkTransactionStatus</subResource>
  </resource>

  <!-- list amount transactions -->
  <resource id="listAmount" name="List amount transactions"
    interfaceName="oracle.Services Gatekeeper.parlayrest.plugin.PaymentPlugin"
methodName="listTransaction"
    tokenExpirePeriod="3600">
  <subResource>checkTransactionStatus</subResource>
  </resource>

  <!-- get amount transaction -->
  <resource id="checkTransactionStatus" name="Get amount transaction"
    interfaceName="oracle.Services Gatekeeper.parlayrest.plugin.PaymentPlugin"
methodName="checkTransactionStatus"
```

```
    tokenExpirePeriod="3600"/>
</resources>
```

In that example, the following resources are defined:

- **chargeAmount**: a token can be obtained for this resource, but this restricts the access of the resource (OneAPI charging API) to the code specified as part of the scope parameter value.
- **listAmount**
- **checkTransactionStatus**

In addition, one subResource is also defined. The **checkTransactionStatus** subResource is defined as a subResource for the **chargeAmount** and **listAmount** resources. By defining **checkTransactionStatus** as a subResource, Services Gatekeeper facilitates using the same access token obtained for **chargeAmount** while accessing the **checkTransactionStatus** resource.

Provisioning of Mapped Resources

Services Gatekeeper contains a JMX interface for uploading resource mapping files. The interface name is included in the **OAuthResourceMBean**, which can be accessed with the OAM WebLogic interface or using the Services Gatekeeper Platform Test Environment (PTE).

This MBean supports the following operations:

- **loadResourceXml**: Loads a resource configuration file in XML format into Services Gatekeeper
- **retrieveResourceXml**: Retrieves a resource configuration file in XML format.
- **retrieveResourceList**: List all resources in a human readable format.

See "Using the OAuthResourceMBean" for more information

Client Management

A client may be mapped to an existing Services Gatekeeper application instance Id. This mapping lets you use existing SLA enforcement in conjunction with OAuth security. Application clients can be managed through the **OAuthClientMBean**, which can be accessed using the OAM WebLogic interface or using the Services Gatekeeper Platform Test Environment (PTE).

This MBean supports the following operations:

- addClient
- updateClient
- removeClient
- getResourceClientInfo
- listClients.

See "Using the OAuthClientMbean" for more information.

Resource Owner - Resource Mapping

A Resource owner for each resource in Services Gatekeeper must be defined to protect the resource. The Resource owner is managed using the **OAuthResourceOwnerMBean**.

This Mbean supports the following operations:

- addResourceOwner
- updateResourceOwner
- removeResourceOwner
- getResourceOwnerInfo

For all operations, the format of a **resourceScope** is space separated scopeId list.

See "Creating Resource Owner/Resource Mappings Using Regular Expressions" for more information.

Default Subscriber Manager

Services Gatekeeper fulfills the Authentication Server role by supporting the creation and authentication of subscribers within the Services Gatekeeper database.

To authenticate users, Services Gatekeeper supports a component called the Subscriber Manager which is used for subscriber provisioning, authentication, and expanding GroupURIs. You manage subscribers using the **SubscriberMBean**.

This Mbean supports the following operations:

- **addSubscriber**: Add a subscriber with address, loginId and password.
- **updateSubscriber**: Update a subscriber specified by address.
- **removeSubscriber**: Delete a subscriber specified by address
- **getSubscriberInfo**: Get information of a subscriber specified by address or loginId
- **verifySubscriber**: Authenticate a subscriber specified by address or loginId and password.

See "Using the SubscriberMBean" for more information.

Deployment and Configuration

To deploy and configure the Services Gatekeeper OAuth functionality, do the following:

1. Confirm the OAuth EAR files are deployed.
2. Update OAuth Service configuration using the **OAuthCommonMBean**. See "Using the OAuthCommonMBean" for more information.
3. Create the protected resources in Services Gatekeeper. See "Using the OAuthResourceMBean" for more information.
4. Configure the Authentication URL.
5. Configure the Resource owner and define the ownership of resources as described in "Resource Management". Also, see "Creating Resource Owner/Resource Mappings Using Regular Expressions" for more information.
6. Create the client identifier(s) and credentials in Services Gatekeeper that will request access to protected resources. See "Using the OAuthClientMbean" for more information.

See "Example: Protecting the OneAPI Payment Service with OAuth" for an example of the configuration steps.

OAuth Configuration

This section explains how to use the **OAuthCommonMBean** to configure Services Gatekeeper for use with OAuth.

Using the OAuthCommonMBean

Managed object: Container Services -> OAuthService

MBean: oracle.Services Gatekeeper.oauth2.management.OAuthCommonMBean

Table 1–3 describes the available attributes.

Table 1–3 Attributes for OAuthCommonMBean

Attributes	Type	Description
TokenType	string	The token type (MAC and Bearer are supported). The default value is: **Bearer**
GrantURL	string	The address used to submit a Resource owner's grant. Represents the address used to submit a resource owner's grant. The default value is: **https://*host:port*/oauth2/grant** where *host* and *port* are the Services Gatekeeper AT node ip address and https port.
MacAlgorithm	string	The MAC algorithm used to calculate the request MAC for access token. The default value is: **hmac-sha-1**
NoOwnerRequestSupport	boolean	Whether or not no owner is supported. The default value is: **true** Specifies the OAuth behavior when there is no address (resource owner) information declared in REST API request parameters. **true**: accept such request **false**: reject such request with 401 "invalid_ request"
GroupUriEnabled	boolean	Whether the group URI option is enabled. The default value is: **true**
CleanDbPeriod	int	Number of seconds until the database is cleaned. The default value is: **60** seconds
SendAnonymousId	boolean	Whether to include AnonymousId in applyAccessToken response event. The default value is: **true**
AuthorizationCodeExpire Period	int	Number of seconds until the authorization code expires. The default value is: **600** seconds
IssueRefreshToken	boolean	Whether to issue a refresh token when issuing an access token. The default value is: **false**

Table 1–3 (Cont.) Attributes for OAuthCommonMBean

Attributes	Type	Description
Authentication URL	string	URL used to authenticate the resource owner and obtain consent for the application requested scope.
IssueRefreshTokenWhenRefresh	boolean	Whether to issue a refresh token when the token is refreshed. The default value is: **false**

Creating Protected Resources

To enable OAuth protection for RESTful Communication Services (including custom Communication Services), Communication Services can be mapped to OAuth resources.

The resource mapping can then be uploaded to Services Gatekeeper using the **OAuthResourceMBean**.

Using the OAuthResourceMBean

Managed object: Container Services -> OAuthService

MBean: oracle.Services Gatekeeper.oauth2.management.OAuthResourceMBean

Following is a list of attributes and operations for the configuration and maintenance of resources.

- Operation: loadResourceXml
- Operation: retrieveResourceXml
- Operation: retrieveResourceList

Operation: loadResourceXml Scope: Cluster

Loads an XML format resource information file using the oauth_resource.xsd schema into Services Gatekeeper.

Signature:

```
loadResourceXml(String xml)
```

See "Resource Mapping" for more information about mapping resources using oauth_resource.xsd.

Operation: retrieveResourceXml Scope: Cluster

Displays resource information in XML format.

Signature:

```
retrieveResourceXml()
```

Operation: retrieveResourceList Scope: Cluster

Displays resource information in list format.

Signature:

```
retrieveResourceList()
```

Configuring Authentication

In order to grant an authorization code, a Resource owner must be authenticated and must also authorize the scope of the grant. The Resource owner can be authenticated using two methods:

- "Using the Default Subscriber Manager"
- "Using Delegated Authentication"

A Resource owner controls its resources and can allow a client to access them. For each resource, one or more resource owners need to be defined using the SubscriberMBean.

Using the Default Subscriber Manager

Services Gatekeeper offers a built-in subscriber repository to authenticate subscribers. To use the default Subscriber Manager to authenticate subscribers, do the following:

1. Create the subscribers in Services Gatekeeper using the **SubscriberMBean**. This MBean can be accessed either through the OAM Console or using the PTE Tools MBean browser. See "Using the SubscriberMBean" for more information.
2. Verify that the AuthenticationURL property of OAuthCommonMbean is set to:

   `https://<AT_HOST>:<AT_PORT>/oauth2/auth.jsp`

Using the SubscriberMBean

Managed object: Container Services -> SubscriberService

MBean: oracle.Services Gatekeeper.subscriber.management.SubscriberMBean

Following is a list of operations for configuring and maintaining subscribers.

- Operation: addSubscriber
- Operation: getSubscriberInfo
- Operation: removeSubscriber
- Operation: updateSubscriber
- Operation: verifySubscriber

Operation: addSubscriber Scope: Cluster

Adds a new subscriber to the Services Gatekeeper Authentication Server.

Signature:

```
addSubscriber ()
```

Table 1–4 describes the attributes used to create a new subscriber in Services Gatekeeper.

Table 1–4 Attributes for New Subscriber

Attributes	Type	Description
Address	string	Subscriber address
LoginId	string	Subscriber login ID
password	string	Subscriber password

Operation: getSubscriberInfo Scope: Cluster

Retrieves subscriber information using an address or login ID.

Signature:

```
getSubscriberInfo()
```

Table 1–5 describes the attributes used to query a subscriber's info in Services Gatekeeper.

Table 1–5 Attributes for Querying Subscriber Info

Attributes	Type	Description
Address	string	Subscriber address
LoginId	string	Subscriber login ID

Operation: removeSubscriber Scope: Cluster

Removes a subscriber by address.

Signature:

```
removeSubscriber()
```

Table 1–6 describes the attributes used to remove a subscriber in Services Gatekeeper.

Table 1–6 Attributes for Removing a Subscriber

Attributes	Type	Description
Address	string	Subscriber address

Operation: updateSubscriber Scope: Cluster

Updates a subscriber's properties.

Signature:

```
updateSubscriber()
```

Table 1–7 describes the attributes used to update a subscriber in Services Gatekeeper.

Table 1–7 Attributes for Updating a Subscriber

Attributes	Type	Description
Address	string	Subscriber address
LoginId	string	Subscriber login ID
password	string	Subscriber password

Operation: verifySubscriber Scope: Cluster

Verifies if a submitted address and password belong to a subscriber.

Signature:

```
verifySubscriber()
```

Table 1–8 describes the attributes used to verify a subscriber in Services Gatekeeper.

Table 1–8 Attributes for Verifying a Subscriber

Attributes	Type	Description
Address	string	Subscriber address
password	string	Subscriber password

Using Delegated Authentication

Instead of provisioning users in the Services Gatekeeper database, you can delegate authentication to a custom Authentication Server.

See "Delegated Authentication" for more information

Creating the Resource Owner/Resource Mapping

Protecting Resource owner resources requires assigning Resource Owners to resources. Use the **OAuthResourceOwnerMBean** to do this. This MBean can be accessed either through the OAM Console or using the PTE Tools MBean browser.

The following sections describe how Resource owner/resource mappings can be created.

- Creating Resource Owner/Resource Mappings Using Regular Expressions
- Creating Individual Resource Owner/Resource Mappings

Creating Resource Owner/Resource Mappings Using Regular Expressions

Managed object: Container Services -> OAuthService

MBean: oracle.Services Gatekeeper.oauth2.management.OAuthResourceOwnerMBean

An XML rules file that defines the resource owner/resource mapping can be created and uploaded to the Services Gatekeeper using the OAuthResourceMBean's loadResourceRuleXml operation. The schema definition for this XML file can be found in resource_rule.xsd.

The rules file contains a list of rules, each defining the mapping between resource owner addresses represented by a regular expression, and a list of one or more resources. The rules file is interpreted at the point of the OAuth authorization grant, and rules must be defined in order of priority from most specific to most general.

In the following example, resource owner/resource mappings are defined with the most specific rules at the beginning of the file, proceeding to the most general rules at the end. This enables rules to run in the expected order, since placing a more general rule, such as `^.*$`, at the top of the file would always generate a match before a more specific rule, such as `^1390.*$`, and would cause the more specific rule to never execute.

Example 1–1 Regular Expression Rules File

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:addressResourceRules
xmlns:tns="http://oracle/ocsg/oauth2/management/resource_rule"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!-- numbers start with '1390' own the 'location' and 'payment' services  -->
<tns:rule addressPattern="^1390.*$" resources="location payment"/>
<!-- numbers start with '139' own the 'location' service  -->
<tns:rule addressPattern="^139.*$" resources="location"/>
<!-- All other resources are protected and no resource owners are defined. -->
<!-- Any request with a scope not defined in this xml file will be rejected. -->
```

```
<tns:rule addressPattern="^.*$" resources=""/>
</tns:addressResourceRules>
```

Following is a list of operations for configuring a resource using regular expressions.

- Operation: loadResourceRuleXML
- Operation: retrieveResourceRuleXML
- Operation: addResourceOwner

Operation: loadResourceRuleXML

Scope: Cluster

Loads resource rules defined in an XML file. See Example 1–1, "Regular Expression Rules File" for an example XML file.

Signature:

```
loadResourceRuleXml(String fileContent)
```

Table 1–9 describes these parameters.

Table 1–9 Parameters for loadResourceRuleXML

Parameter	Description
fileContent	An XML file containing rule definitions.

Operation: retrieveResourceRuleXML

Scope: Cluster

Retrieves the current rules XML file.

Signature:

```
updateResourceOwner(String address, String resourceScope)
```

Table 1–10 describes these parameters.

Table 1–10 Parameters for retrieveResourceRuleXML

Parameter	Description
address	Address of the Resource owner. The format of address depends on which resource the client trying to access. For most Communication Services, only **tel:**[+][**0-9**]+ is supported, for example: `tel:15415550100` For sip-related services, the sip URI is supported.
resourceScope	The value of an id attribute (scopeId) which is defined as part of a resource. This scopeId is submitted as part of the authorization grant request. The Resource owner grants authorization for the scope (API, method, parameters) defined by this scopeId.

Creating Individual Resource Owner/Resource Mappings

Managed object: Container Services -> OAuthService

MBean: oracle.Services Gatekeeper.oauth2.management.OAuthResourceOwnerMBean

Following is a list of operations for configuring resource owners individually.

- Operation: addResourceOwner

Operation: addResourceOwner

Scope: Cluster

Adds a Resource owner in Services Gatekeeper.

The value of the id attribute, scopeId, is used in creating a resource owner for a resource. This scopeId is the value of the resourceScope parameter.

Signature:

```
addResourceOwner(String address, String resourceScope)
```

Table 1–11 describes these parameters.

Table 1–11 Parameters for addResourceOwner

Parameter	Description
address	Address of the Resource owner. The format of address depends on which resource the client trying to access. For most Communication Services, only **tel:**[+][**0-9**]+ is supported. For example: `tel:15415550100` For sip-related services, the sip URI is supported.
resourceScope	The value of an id attribute (scopeId) which is defined as part of resource. This scopeId is submitted as part of the authorization grant request. The Resource owner grants authorization for the scope (API, method, parameters) defined by this scopeId.

Operation: updateResourceOwner

Scope: Cluster

Updates the properties of a Resource owner in Services Gatekeeper.

Signature:

```
updateResourceOwner(String address, String resourceScope)
```

Table 1–12 describes these parameters.

Table 1–12 Parameters for updateResourceOwner

Parameter	Description
address	Address of the Resource owner
resourceScope	Scope of the resource identified by scopeId.

Operation: removeResourceOwner

Scope: Cluster

Removes a Resource owner by address from Services Gatekeeper.

Signature:

```
removeResourceOwner(String address)
```

Table 1–13 describes these parameters.

Table 1–13 Parameters for removeResourceOwner

Parameter	Description
address	Address of the Resource owner

Operation: getResourceOwnerInfo

Scope: Cluster

Retrieves Resource owner information by address from Services Gatekeeper.

Signature:

```
getResourceOwnerInfo(String address)
```

Table 1–14 describes these parameters.

Table 1–14 Parameters for getResourceOwnerInfo

Parameter	Description
address	Address of the Resource owner

Configuring Clients

Services Gatekeeper allows the creation of an application client to support the authorization code grant type. A client registers with its password and the allowed redirect URI in Services Gatekeeper. The **OAuthClientMBean** is used to configure application clients, and can be accessed either through the OAM Console or using the PTE Tools MBean browser. See "Using the OAuthClientMbean" for more information.

Using the OAuthClientMbean

Managed object: Container Services -> OAuthService

MBean: oracle.Services Gatekeeper.oauth2.management.OAuthClientMBean

Following is a list of attributes and operations for configuration and maintenance of clients.

- Operation: addClient
- Operation: updateClient
- Operation: removeClient
- Operation: getClientInfo
- Operation: listClients

Operation: addClient

Scope: Cluster

Adds an application client to Services Gatekeeper.

Signature:

```
addClient(String id, String name, String password, String description, String
allowedRedirectionURI, Boolean supportImplicitGrant, String appInstanceID)
```

Table 1–15 describes these parameters.

Table 1–15 Parameters for addClient

Parameter	Description
id	Client identifier
name	Client name
password	Client password
description	Client description
allowedRedirectionURI	Allowed redirection URIs, split by a space character (for example, "URI1 URI2 URI3")
supportImplicitGrant	Whether implicit grant is supported
AppInstanceId	Application instance Id in Services Gatekeeper

Operation: updateClient

Scope: Cluster

Updates an existing client in Services Gatekeeper.

Signature:

```
updateClient(String id, String name, String password, String description, String
allowedRedirectionURI, Boolean supportImplicitGrant, String appInstanceID)
```

Table 1–16 describes these parameters.

Table 1–16 Parameters for updateClient

Parameter	Description
id	Client identifier
name	Client name
password	Client password
description	Client description
allowedRedirectionURI	Allowed redirection URIs, split by a space character (for example, "URI1 URI2 URI3")
supportImplicitGrant	Whether implicit grant is supported
AppInstanceId	Application instance Id in Services Gatekeeper

Operation: removeClient

Scope: Cluster

Removes a client in Services Gatekeeper by Id.

Signature:

```
removeClient(String id)
```

Table 1–17 describes these parameters.

Table 1–17 Parameters for removeClient

Parameter	Description
id	Client identifier

Operation: getClientInfo

Scope: Cluster

Retrieves client information in Services Gatekeeper by Id.

Signature:

```
getClientInfo(String id)
```

Table 1–18 describes these parameters.

Table 1–18 Parameters for getClientInfo

Parameter	Description
id	Client identifier

Operation: listClients

Scope: Cluster

Retrieves a list of all clients in Services Gatekeeper.

Signature:

```
listClients(int offset, int size))
```

Table 1–19 describes these parameters.

Table 1–19 Parameters for listClients

Parameter	Description
Offset	Offset within the complete result set. Must be >= 0.
Size	Number of entries to return. 0 means no limit.

Protecting Custom REST APIs with OAuth

It is possible to create a custom RESTful communication service using the Platform Development Studio (PDS) wizard and deploy the service on the Services Gatekeeper platform. This service will be an OAuth2.0 protected service automatically.

The service needs to be provisioned as an OAuth2.0 resource as described in the following steps:

1. Deploy the service into the Services Gatekeeper server.

   For example:

   ```
   Interfacename=com.foo.Demo
   Method=bar
   ```

2. Execute the loadResourceXml operation in the OAuthResourceMbean to create a new resource element. For example, use the following XML:

   ```
   <resource id="foo_id " name="Demo Service" interfaceName=" com.foo.Demo"
   methodName=" bar"  tokenExpirePeriod="3600"></resource>
   ```

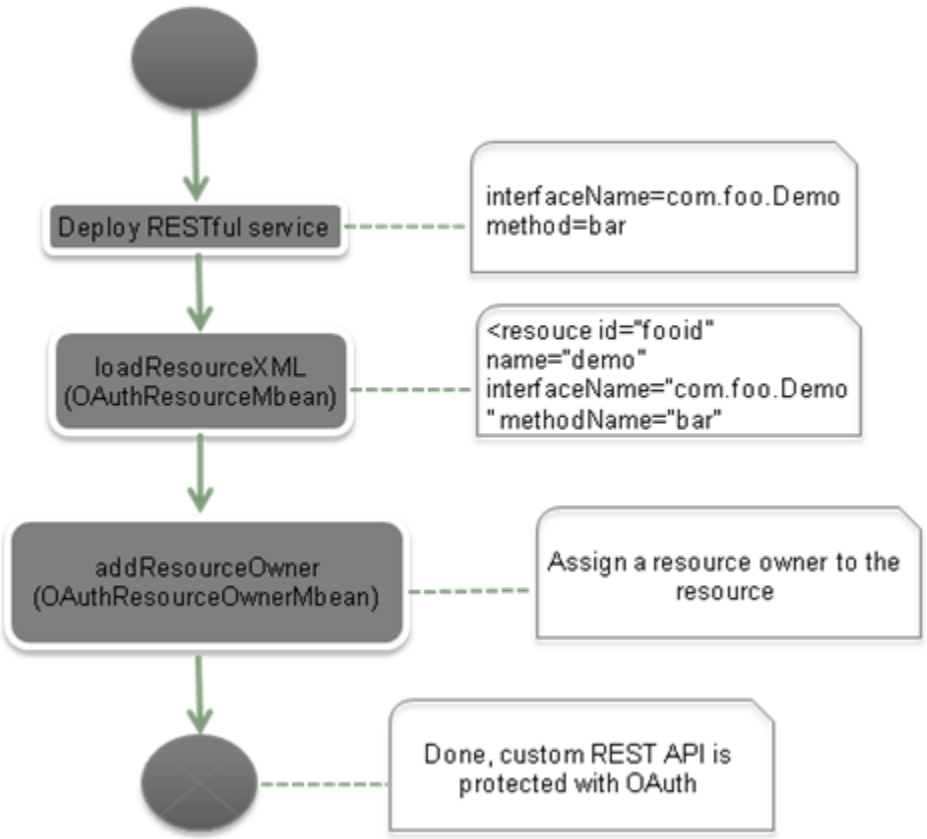
3. Execute the addResourceOwner operation in the OAuthResourceOwnerMbean with the following values (where the resource_owner_address is the tel:/sip: URI):

   ```
   address=${resource_owner_address}
   resourceScope=foo_id
   ```

After completing the above steps, the custom RESTful communication service can be accessed if a resource owner grants access to any application client.

Figure 1–6 illustrates the above steps.

Figure 1–6 Protecting a Custom Communication Service



Example: Protecting the OneAPI Payment Service with OAuth

This section details how to protect the OneAPI Payment Service with OAuth.

Steps in Protecting the OneAPI Payment Service with OAuth

1. "Adding a Client in Services Gatekeeper"
2. "Configuring the Authentication URL"
3. "Adding One API Payment Communication Service as an OAuth resource"
4. "Adding a New Subscriber"
5. "Assigning the Resource to the Subscriber to Act as Resource owner"

Adding a Client in Services Gatekeeper

1. Open the MBean browser in PTE or use the OAM to access the **addclient** operation in the OAuthClient MBean at:

   Wlng -> OauthServer -> OauthClientMBean -> addClient()

2. Use the following parameters for the addclient operation:

- Id: app123
- Name: App123_name
- Password: app123
- Description: Demo Application
- AllowRedirectURI: `https://localhost/app/redirect.php`
- AppInstanceId: domain_user

3. Submit the parameters to add the new client.

Configuring the Authentication URL

1. Open the MBean browser in PTE or use the OAM to access the AuthenticationURL configuration setting in the OAuthCommon MBean at:

   Wlng -> OauthServer -> OauthCommonMBean -> AuthenticationURL

2. Use the following parameters for the **AuthenticationURL** field:

   - AuthenticationURL: `https://`<AThost>:<Port>`/oauth2/auth.jsp`

3. Submit the parameter to set the authentication URL.

Adding One API Payment Communication Service as an OAuth resource

1. Open the MBean browser in PTE or use the OAM to access the **loadResourceXml** operation in the OAuthResourceMBean at:

   Wlng -> OauthServer -> OauthResourceMBean -> loadResourceXml()

2. Use the following sample XML content for the **FileContent** parameter field:

```
<?xml version="1.0" encoding="UTF-8"?>
<resources xmlns="http://oracle/Services Gatekeeper/oauth2/management/xml"
mlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!--  OneAPI Payment  amountTransaction    -->
  <resource id="POST-/payment/acr:Authorization/transactions/amount"
name="Charge or refund" interfaceName="oracle.Services
Gatekeeper.parlayrest.plugin.PaymentPlugin" methodName="amountTransaction"
tokenExpirePeriod="3600">
  <parameter name="code" description="billable item id"/>
  </resource>
</resources>
```

3. Submit the parameter to add the payment service as an OAuth resource.

Adding a New Subscriber

1. Open the MBean browser in PTE or use the OAM to access the **addSubscriber** operation in the SubscriberMBean at:

   Wlng -> SubscriberService -> SubscriberMBean -> addSubscriber()

2. Use the following parameters for the addSubscriber operation:

   - Address: tel:888
   - LoginID: Jack
   - Password: 888

3. Submit the parameters to create a new subscriber.

Assigning the Resource to the Subscriber to Act as Resource owner

1. Open the MBean browser in PTE or use the OAM to access the **addResourceOwner** operation in the OAuthResourceOwnerMBean at:

   Wlng -> OauthServer -> OauthResourceOwnerMBean -> addResourceOwner()

2. Use the following parameters for the addResourceOwner operation:
   - Address: tel:888
   - resourceScope: `POST-/payment/acr:Authorization/transactions/amount`
3. Submit the parameters to create a new subscriber.

OAuth Runtime

This section describes the Services Gatekeeper runtime OAuth actions. The Services Gatekeeper runtime supports an OAuth Authorization Server which validates Resource owner identity, and issues authorization codes and access tokens.

Token Issuance

This section describes the mechanism by which Services Gatekeeper manages OAuth token issuance.

Default Authentication and Authorization

Services Gatekeeper includes an authorization JSP page to enter Resource owner information and validate the scope requested by an application. This page displays the scope and resource details that a Resource owner uses to issue an Authorization Grant to an application.

The Auth.jsp that handles the default authentication is located in the oauth2_service.war at the following location:

$Middleware_Home`/ocsg_5.0/applications/wlng_at_oauth2.ear`

Authorization for Group URIs

The default Subscriber Manager treats a Group owner with a group URI as a normal Resource owner, so a group owner with a group URI has its own password.

When Services Gatekeeper issues an Authorization Grant for a given group URI as a resource owner, the token can be used to access a resource on behalf of any of the group members.

The owner of the group URI and password will be able to authorize an application to access resources that are owned by each member in the group.

This feature can be enabled or disabled using the **groupUriEnabled** Mbean property in the **OauthCommonMbean**.

See "OAuth Protection for APIs Involving Multiple Resource owners" for more information.

Token Validation

Services Gatekeeper supplies an interceptor to validate access tokens. The interceptor must be configured with a minimum index value in the interceptor chain to allow validation requests to be handled properly. By default, it is the second interceptor.

The token validation interceptor checks the following:

- Whether the access token is expired
- Whether the Resource owner matches the end user ID in the RESTful API request
  - An exact match for a Resource owner
  - If token is issued for group URI, then request URI should be a member of the same group
- Whether requested resource is within the scope of a token

If the token is a MAC type, additional checks are required for the following:

- Body hash
- Nonce
- Mac

Custom interceptors can be developed and deployed for parameter value checking and token type checking (if MAC Type) of requests.

Token Management

This section describes managing OAuth tokens via JMX interfaces using the **TokenManagementMBean**, which can be accessed with the OAM WebLogic interface or using the Services Gatekeeper Platform Test Environment (PTE). See "Using the TokenMangementMBean" for more information.

Using the TokenMangementMBean

Managed object: Container Services -> OAuthService

MBean: oracle.ocsg.oauth2.management.TokenManagementMBean

Following is a list of supported operations for this MBean.

- Operation: listAccessTokensByEndUser
- Operation: listRefreshTokensByEndUser
- Operation: listAccessTokensByClientIdAndEndUser
- Operation: listRefreshTokensByClientIdAndEndUser
- Operation: listAccessTokensByClientId
- Operation: listRefreshTokensByClientId
- Operation: countAccessTokensByClientId
- Operation: countRefreshTokensByClientId
- Operation: revokeAccessToken
- Operation: revokeRefreshToken

Operation: listAccessTokensByEndUser

Scope: Cluster

List access tokens by the address of an end user.

Signature:

```
listAccessTokensByEndUser(String endUserId)
```

Table 1–20 describes these parameters.

Table 1–20 Parameters for listAccessTokensByEndUser

Parameter	Description
endUserId	Resource owner URI.

Operation: listRefreshTokensByEndUser

Scope: Cluster

List refresh tokens by the address of an end user.

Signature:

```
listRefreshTokensByEndUser(String endUserId)
```

Table 1–21 describes these parameters.

Table 1–21 Parameters for listRefreshTokensByEndUser

Parameter	Description
endUserID	Resource owner URI.

Operation: listAccessTokensByClientIdAndEndUser

Scope: Cluster

List access tokens by the client Id in addition to the address of an end user.

Signature:

```
listAccessTokensByClientIdAndEndUser(String clientId, String endUserId)
```

Table 1–22 describes these parameters.

Table 1–22 Parameters for listAccessTokensByClientIdAndEndUser

Parameter	Description
clientID	Client identifier.
endUserId	Resource owner URI.

Operation: listRefreshTokensByClientIdAndEndUser

Scope: Cluster

List refresh tokens by the client Id in addition to the address of an end user.

Signature:

```
listRefreshTokensByClientIdAndEndUser(String clientId, String endUserId)
```

Table 1–23 describes these parameters.

Table 1–23 Parameters for listRefreshTokensByClientIdAndEndUser

Parameter	Description
clientID	Client identifier.
endUserId	Resource owner URI.

Operation: listAccessTokensByClientId

Scope: Cluster

List access tokens by the client Id.

Signature:

```
listAccessTokensByClientId(String clientId, int offset, int size)
```

Table 1–24 describes these parameters.

Table 1–24 Parameters for listAccessTokensByClientId

Parameter	Description
clientID	Client identifier.
Offset	Offset within a complete result set. Must be >= 0.
Size	Number of entries to return. 0 means no limit.

Operation: listRefreshTokensByClientId

Scope: Cluster

List refresh tokens by the client Id.

Signature:

```
listRefreshTokensByClientId (String clientId, int offset, int size)
```

Table 1–25 describes these parameters.

Table 1–25 Parameters for listRefreshTokensByClientId

Parameter	Description
clientID	Client identifier.
Offset	Offset within a complete result set. Must be >= 0.
Size	Number of entries to return. 0 means no limit.

Operation: countAccessTokensByClientId

Scope: Cluster

List the number of access tokens by client Id.

Signature:

```
countAccessTokensByClientId(String clientId)
```

Table 1–26 describes these parameters.

Table 1–26 Parameters for countAccessTokensByClientId

Parameter	Description
clientID	Client identifier.

Operation: countRefreshTokensByClientId

Scope: Cluster

List the number of refresh tokens by client Id.

Signature:

```
countRefreshTokensByClientId(String clientId)
```

Table 1–27 describes these parameters.

Table 1–27 Parameters for countRefreshTokensByClientId

Parameter	Description
clientID	Client identifier.

Operation: revokeAccessToken

Scope: Cluster

Revoke an access token.

Signature:

```
revokeAccessToken(String token)
```

Table 1–28 describes these parameters.

Table 1–28 Parameters for revokeAccessToken

Parameter	Description
token	OAuth access token.

Operation: revokeRefreshToken

Scope: Cluster

Revoke a refresh token.

Signature:

```
revokeRefreshToken(String token)
```

Table 1–29 describes these parameters.

Table 1–29 Parameters for revokeRefreshToken

Parameter	Description
token	OAuth access token.

EDRs Generated by the OAuth Service

Table 1–30 describes the EDRs that can be generated by the OAuth service.

Table 1–30 OAuth Service EDRs

EDR ID	Class	Method	Description	Additional Attributes in the EDR
20001	oracle.ocsg.oauth2.interceptor.OAuth2AppListener	postStart	Generated when the OAuth service is started.	None
20002	oracle.ocsg.oauth2.interceptor.OAuth2AppListener	preStop	Generated when the OAuth service is stopped.	None
20003	oracle.ocsg.oauth2.ejb.server.OAuthServiceBean	authorize	Generated when an authorization code is issued to an application.	OAuth2ClientId OAuth2ResourceOwner OAuth2Scopes OAuth2AuthorizeType If the response is a code: OAuth2AuthorizationCode If the response is a token: OAuth2AccessToken OAuth2TokenType
20004	oracle.ocsg.oauth2.ejb.server.OAuthServiceBean	applyToken	Generated when an access token is issued to an application.	OAuth2ClientId OAuth2ResourceOwner OAuth2GrantType OAuth2AuthorizationCode OAuth2AccessToken OAuth2TokenType If a refresh token is generated: OAuth2RefreshToken
20005	oracle.ocsg.oauth2.ejb.server.OAuthServiceBean	refreshToken	Generated when an application requests a refresh token.	OAuth2ClientId OAuth2ResourceOwner OAuth2GrantType OAuth2OrignalRefreshToken OAuth2AccessToken OAuth2TokenType If a refresh token is generated: OAuth2RefreshToken
20006	oracle.ocsg.oauth2.interceptor.OAuth2Interceptor	invoke	Generated when an application accesses a resource with an OAuth access token.	OAuth2ClientId OAuth2ResourceOwner OAuth2AccessToken OAuth2TokenType OAuth2ResourceClass OAuth2ResourceMethod

Customization

This section contains information on customizing Service Gatekeeper's OAuth functionality.

Delegated Authentication

Authentication can be done with a customized authentication service provider instead of with the default Subscriber Management Service. The custom authentication provider is responsible for the Resource owner identity validation and handling the grant collection flow.

A delegated authentication service used with Services Gatekeeper is responsible for:

1. Hosting the Authentication Endpoint.
2. Presenting the expanded scope and authenticating a Resource owner
3. Redirecting the Resource owner to the Grant Endpoint hosted by Services Gatekeeper upon successful authentication of the Resource owner.

See "Endpoints Mapping" for more information.

Delegated Authentication Process Flow

This section describes the flow of requests between Services Gatekeeper and a delegated authentication service. Sample responses to the requests along with a description of the flow are provided.

1. An application sends a standard OAuth2.0 Authorization request to Services Gatekeeper in a format that looks like:

   ```
   GET /oauth2/authorize?client_id=client123&redirect_
   uri=https://www.google.com/asdf&response_
   type=code&scope=POST-/payment/acr:Authorization/transactions/amount&state=123
   HTTP/1.1
   ```

   After receiving the OAuth2 authorization request, Services Gatekeeper will load more detailed information using the **client_id** and **scope** parameters in the request. This additional information is appended to the Location header in the 302 redirect response directed to the configured authentication endpoint.

   The Location header contains following elements:

   - The delegating authentication endpoint
   - The original OAuth2.0 Authentication request parameters
   - The Grant endpoint
   - Detailed information for the **client_id** and scope parameters.

   An example 302 response is provided here:

   ```
   HTTP/1.1 302 Moved Temporarily
   Location: https://authentication_url?client_id=client123&redirect_
   uri=https://www.google.com/asdf&response_
   type=code&scope=POST-/payment/acr:Authorization/transactions/amount&state=123&g
   rant_url=grant&client_
   info=%7B%22clientId%22%3A%22client123%22%2C%22clientName%22%3A%22client123%22%2
   C%22clientDescription%22%3A%22client123+desc%22%7D&scopes_
   info=%5B%7B%22scopeId%22%3A%22POST-%2Fpayment%2Facr%3AAuthorization%2Ftransacti
   ons%2Famount%22%2C%22scopeDescription%22%3A%22Charge+or+refund%22%2C%22paramete
   rs%22%3A%5B%7B%22code%22%3A%22billable+item+id%22%7D%5D%7D%5D
   ```

In addition to the original OAuth2.0 authorization request parameters, the detailed format specs of all additional parameters are defined in Table 1–31:

Table 1–31 OAuth 2.0 Authorization Request Parameters

Parameter	Description
grant_url	The URL can be submitted later according Resource owner's approval. See "Endpoints Mapping" for more information.

client_info	Client information will be constructed into a JSON Object as shown below. Encoding complies with the following specification: http://www.w3.org/Addressing/URL/url-spec.html
	`{`
	`"clientId":"client123",`
	`"clientName":"Oracle",`
	`"clientDescription":"Oracle Description"`
	`}`

scopes_info	scope information will be constructed into a JSON Object as shown below. Encoding complies with the following specification: http://www.w3.org/Addressing/URL/url-spec.html
	`[`
	`{`
	`"scopeId":"POST- payment acr:Authorization transactions amount",`
	`"scopeDescription":"Charge+or+refund",`
	`"parameters":[{"code":"billable+item+id"}]`
	`}`
	`]`

2. The Resource owner's browser continues to access the authentication endpoint identified in the Location Header.

   The Authentication endpoint should accept the redirected request, authenticate the Resource owner for proper credentials and render an interactive graphical interface for authorization. The Resource owner can use the information in the interface to understand the scope and client information of the authorization request and determine if the request should be authorized.

   After the Resource owner authorizes the scope the Authentication endpoint redirects the Resource owner to the Grant endpoint with following parameters through an HTTP POST operation. The OAuth flow continues normally after redirecting the Resource owner towards the grant endpoint. The client then receives the authorization code at the Redirect endpoint.

   Table 1–32 lists the OAuth 2.0 grant endpoint POST parameters.

Table 1–32 OAuth 2.0 Grant Endpoint POST Parameters

Parameter	Description
user_address	Address of Resource owner

Table 1–32 (Cont.) OAuth 2.0 Grant Endpoint POST Parameters

Parameter	Description
grant_scopes	The scope that the Resource owner grants to the application. The value of the scope parameter is expressed as a list of space-delimited strings. Each string adds an additional access range to the selected scope parameter. According to the Resource owner decisions at the Authentication endpoint, the granted scope can be narrower than the originally requested scope. Services Gatekeeper will reject a granted scope that is wider than originally requested scope. Based on the implementation of the Authentication endpoint and the Resource owner interaction, additional parameter may be appended to each scope id. These scope parameters will be available to an interceptor so that stricter enforcement can be applied according to different parameters. The scope format is: scopeId?[<*param*>=<*value*>[&<*param*>=<*value*>]*]. For example: `grant_scopes=chargeAmount?maxAmount=100&minAmount=100 getLocation?requestedAccuracy=100 sendSMS`
response_type	As in the first authorization request
client_id	As in the first authorization request
redirect_uri	As in the first authorization request
state	As in the first authorization request
scope	As in the first authorization request

Customized OAuth Interceptor

This section describes the basic principles around creating a customized OAuth interceptor.

As described in Delegated Authentication, it is possible to add additional parameters to the scope-token so that custom interceptors can be created for fine-grained resource access and traffic control.

Table 1–33 lists the OAuth parameters available in the RequestContext object for an OAuth2.0 enabled Communication Service. Customized interceptors can make use of these parameters to further fine tune authorized access to protected resources.

For information on creating custom interceptors, see *Platform Development Studio Developer's Guide*.

Table 1–33 OAuth 2.0 Grant Endpoint POST Parameters

Attribute Name	Access	Type	Description
OAUTH2_SCOPE_PARAMETER	read-only	java.util.Map	Contains all parameters of the current request scope.
CONTEXT_OAUTH2_RESOURCE_ OWNER	read-only	java.lang.String	The Resource owner of the token, which is usually the same as the address in the request. When the Resource owner is a group URI or the scheme of address in request is 'acr', they may be different.
CONTEXT_OAUTH2_PARAMETER	read-only	java.util.Map	Contains all endpoint parameters of this request. This parameter must start with "oracle_" or "ocsg_".
CONTEXT_OAUTH2_STATE	read-write	java.util.Map	Values of this attribute will be available during the lifecycle of one OAuth access token.

Examples: Customized OAuth Interceptor

Below is an example for retrieving and using OAuth associated information from the requestContext within a customized interceptor.

To retrieve the MSIDN of an OAuth Resource owner:

```
/**
* The following example shows a way to retrieve Oauth2 resource owner MSIDN
*/
@Override
public Object invoke(final Context context) throws Exception {
  String currentResourceOwner = (String) context.getRequestContext().get("CONTEXT_
OAUTH2_RESOURCE_OWNER");
  If (currentResourceOwner == null)
throw new DenyPluginException("Not a OAuth based request!");
  else
System.out.println("Current Oauth2 resource owner is:" + currentResourceOwner);
  context.invokeNext(this);
}
```

To control the maximum charged value using an additional scope parameter called maxAmount:

```
/**
* The following example shows a way to control the maximum charged value by means
of additional scope parameter
* "maxAmount".
*/
@Override
public Object invoke(final Context context) throws Exception {
if (context.getType().equals(AmountCharingPlugin.class)) {
    Map<String, String> scopeParameters = (Map<String,
String>)context.getRequestContext().get("OAUTH2_SCOPE_PARAMETER");
int maxAmount = Integer.parseInt(scopeParameters.get("maxAmount").toString());

if (((ChargeAmount)context.getArguments()[0]).getAmount() > maxAmount)
  throw new DenyPluginException("Specified chargeAmount request exceed
limitation.");
 }
  context.invokeNext(this);
}
```

Custom Subscriber Manager

Service Gatekeeper offers the flexibility to integrate with custom identity stores for user authentication. It is possible to develop a customized Subscriber Manager so that users can be authenticated against external identity stores such as LDAP.

Developing a custom Subscriber Manager involves the following steps:

1. Implementing oracle.Services Gatekeeper.subscriber.SubscriberManager, and customizing the implementation of this interface
2. Registering the custom implementation with the OAuth security framework using the following method.

```
void oracle.Services
Gatekeeper.subscriber.SubscriberManager.registerInstance("default",
SubscriberManager instance);
```

   For additional information on customizing the default Subscriber Manager, including method details, see the Services Gatekeeper JavaDoc.

Application Developer Guide

This section contains information helpful to Application Developers using OAuth2.0 with Services Gatekeeper.

Interacting with the Services Gatekeeper OAuth Service

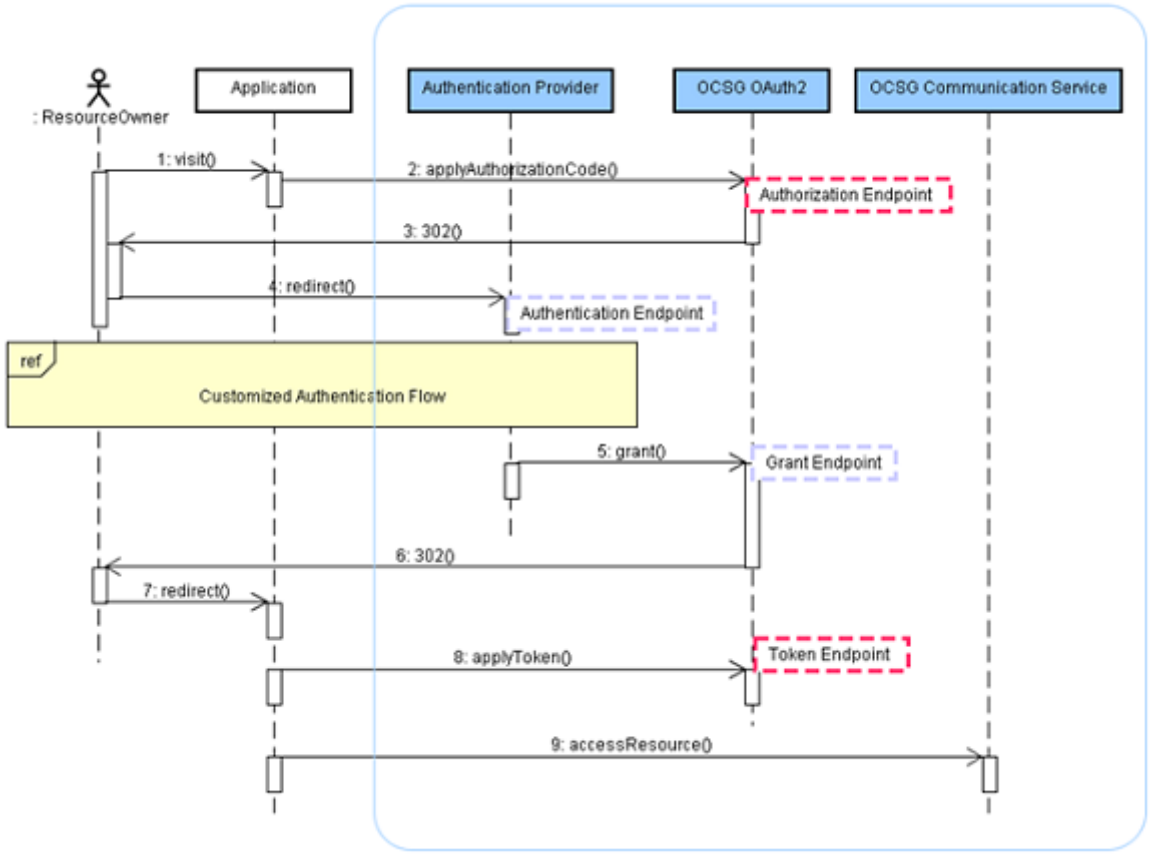
This section describes what OAuth endpoints are available, what steps are involved in obtaining an access token and how an application can use an access token to access a REST resource in Services Gatekeeper.

The following endpoints are available in the OAuth2.0 Service:

- Authorization Endpoint
- Token Endpoint
- Authentication Endpoint
- Grant Endpoint

Figure 1–7 demonstrates the end to end flow to obtain an access token and use the access token to access the resource.

Figure 1–7 OAuth Endpoints and Functional Responsibility



OAuth Access Flow In Services Gatekeeper

The OAuth access flow in Services Gatekeeper is described in the following steps:

1. A Resource owner visits an application Web site and initiates a request that requires granting access to protected resources to an application.
2. The application redirects the Resource owner to the **Authorization Endpoint** with the application information including the client id and scope id.

   For example, the application can provide a link to trigger a HTTP GET request where the following information is included in the HTTP query string:

   - **HTTP Request:** `GET`
   - **URI**: `https://host:port/oauth2/authorize`
   - **Parameters**:
     - **response_type** -- Supported values are code or token
     - **client_id**. -- The client identifier
     - **redirect_uri** -- Required
     - **scope** -- The scope of the access request expressed as a list of space-delimited, case sensitive strings. The Services Gatekeeper Authorization Server accepts zero to multiple scope-tokens in the following format for scope-token:

<scopeId>[?<*param*>=<*value*>[&<*param*>=<*value*>]*]+

where **scopeId** is the resource identifier and **param** is the name of one of the allowed parameters defined as part of resource.

For example:

chargeAmount?code=1976

An example scope would look like:

```
GET /oauth2/authorize?response_type=code&client_id=app123&state=xyz
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1

Host: server.Services Gatekeeper.com
```

3. Services Gatekeeper validates the resource owner identity and obtains the resource owner's consent on the requested scope.
4. The application exchanges the authorization code for an authorization token via the Token Endpoint. Services Gatekeeper server returns the token directly.

   The request can be described as follows:

   - **HTTP Request**: `POST`
   - **URI**: `https://`<AT_HOST>:<AT_PORT>`/oauth2/token`
   - **Parameters**:
     - **grant_type**: Value can be set to authorization_code, if the request is not a SAML assertion.
     - **code**: The authorization code received from the Authorization Server.
     - **redirect_uri**: The redirection URI used by the Authorization Server to return the authorization response in the previous step.
     - **client_id**: The client identifier.
     - **client_secret**: The client password.
   - **Authorization Header**:

     The client application may use the http Basic authentication scheme as defined in RFC2617 to authenticate with the Services Gatekeeper server. The client_id is used as the username and the client_secret is used as the password.

     For example:

     ```
     Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
     ```

     Alternatively, the Authorization Server may support including the client credentials in the request body using the following parameters:

     - **client_id**: The client identifier.
     - **client_secret**: The application password.
   - **HTTP Response:**
     - **access_token**: The authorization code generated by Services Gatekeeper.
     - **token_type**: The Bearer or MAC authorization code received from Services Gatekeeper.
     - **expires_in**: The duration in seconds of the access token lifetime.

- **refresh_token**: The refresh token which can be used to obtain new access tokens using the same Authorization Grant.
- **scope**: The scope of the access request expressed as a list of space-delimited, case sensitive strings.
- **anonymous_id**: Uniquely identifies the Resource owner.
- **mac_key**: The MAC key is used to verify the later request of access protect resource. (MAC-Type access token).
- **mac_algorithm**: The MAC algorithm used to calculate the request MAC. The value must be either hmac-sha-1 or hmac-sha-256.

The response will be different depending on the token type submitted in the request.

This is an example for a Bearer-Type Access Token with HTTP Basic Authentication:

Request:

```
POST /oauth2/token HTTP/1.1
Host: localhost:7999
Content-length: 128
Authorization: Basic YXBwMTIzOmFwcDEyMw==
Content-Type: application/x-www-form-urlencoded
Connection: Close

grant_type=authorization_
code&code=75dfe1c9-9784-4545-846f-e1493f087017&redirect_
uri=http%3A%2F%2Flocalhost%2Fapp%2Fredirect.php
```

Response:

```
HTTP/1.1 200 OK
Cache-Control: no-store
Connection: close
Content-Length: 327
Content-Type: application/json

{"access_token":"44fb85f8-e400-41b3-9bd4-68617d131039","token_
type":"MAC","expires_
in":3600,"scope":"POST-/payment/acr:Authorization/transactions/amount","mac
_algorithm":"hmac-sha-1","mac_
key":"-3677656698299327487","secret":"-3677656698299327487","anonymous_
id":"1debde44-f9d4-41d6-88fe-bbb77fea37c8","algorithm":"hmac-sha-1"}
```

The following example is for a MAC-Type Access Token with included client credentials in the request body.

Request:

```
POST /oauth2/token HTTP/1.1
Host: server.Services Gatekeeper.com
Content-Type: application/x-www-form-urlencoded
   grant_type=authorization_code&client_id=app123&client_
secret=app123&code=i1WsRn1uB1&redirect_
uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
```

```
{  "access_token":"SlAV32hkKG",    "token_type":"mac",    "expires_
in":3600,    "refresh_token":"8xLOxBtZp8",    "secret":"23sasd#adf@#"
"algorithm":"hmac-sha-1"}
```

5. The application now access the protected resource.

   The application needs to add an HTTP Authorization Header when accessing the granted resource. The value of authorization head depends on the access token type.

   For a Bearer token, the application can directly transmit the access token via an HTTP authorization header in the request.

   For a MAC token, the application constructs the HTTP authorization header using a MAC key together with the access token. For more information, see: http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-00.

   Below is a sample PHP code snippet illustrating the insertion of the token in the HTTP Authorization Header:

```
$body_hash=base64_encode(hash('sha1',$http_body,true));
$payload=$nonce."\n".$http_method."\n".$request_path."\n".$host_
name."\n".$host_port."\n".$body_hash."\n".$ext."\n";
$mac = base64_encode(hash_hmac('sha1', $payload, $mac_key, true));
$oauth2_header='MAC id="'.$mac_key_
id."\",nonce=\"".$nonce."\",bodyhash=\"".$body_hash."\",mac=\"".$mac.'"';
```

   Following is an example of a request containing a Bearer authorization token:

```
POST /oneapi/1/payment/acr%3AAuthorization/transactions/amount HTTP/1.1
     Host: localhost:7999
     Content-Type: application/x-www-form-urlencoded
     Authorization: Bearer vF9dft4qmT

{"amountTransaction":{"endUserId":"acr:Authorization",
"paymentAmount":{"chargingInformation":{"description":"chargeAmount",
"currency":"USD",
"amount":"2","code":""},
"chargingMetaData":{"onBehalfOf":"Example Games Inc",
"purchaseCategoryCode":"Game",
"channel":"",
"taxAmount":"0",
"mandateId":"",
"serviceId":"",
"productId":""}},
"transactionOperationStatus":"Charged",
"referenceCode":"REF-12345",
"clientCorrelator":""}
}
```

   Following is an example of a request containing a MAC authorization token:

```
POST /oneapi/1/payment/acr%3AAuthorization/transactions/amount HTTP/1.1
Host: localhost:7999
Content-length: 415
Authorization: MAC id="176c04f0-d4d4-4385-b2d6-b19649f21b78",
nonce="273156:di3hvdf8",
bodyhash="junEVZu4M9q1qVaxAByY7lYQun8=",
mac="TudmT3bM5UgqvkL8nq1EuhcZ6O8="
Content-Type: application/json
X-Session-ID: app:-7562122823730178188
```

```
Connection: Close

{"amountTransaction":{"endUserId":"acr:Authorization",
"paymentAmount":{"chargingInformation":{"description":"chargeAmount",
"currency":"USD",
"amount":"2",
"code":""},
"chargingMetaData":{"onBehalfOf":"Example Games Inc",
"purchaseCategoryCode":"Game",
"channel":"",
"taxAmount":"0",
"mandateId":"",
"serviceId":"",
"productId":""}},
"transactionOperationStatus":"Charged",
"referenceCode":"REF-"
}
```

Errors and Exceptions

Table 1–34 describes the error conditions and resulting operation responses provided by the Services Gatekeeper OAuth 2.0 Authorization Server.

Table 1–34 Exception Scenarios

Type	Error	Response

invalid_ request	The request is missing a required parameter, includes an invalid parameter value, or is otherwise malformed.	`HTTP/1.1 400 Bad Request`
		`Content-Type: application/json`
		`Cache-Control: no-store`
		`{`
		`"error":"invalid_request"`
		`}`

invalid_realm	No authentication header with the default Services Gatekeeper realm	`HTTP/1.1 401 Unauthorized` `WWW-Authenticate: realm="default"`

invalid_grant	The provided Authorization Grant (for example authorization code or Resource owner credentials) is invalid, expired, revoked, and does not match the redirection URI used in the authorization request, or was issued to another client.	`HTTP/1.1 400 Bad Request`
		`Content-Type: application/json`
		`Cache-Control: no-store`
		`{`
		`"error":"invalid_grant"_`
		`}`

Table 1–34 (Cont.) Exception Scenarios

Type	Error	Response

invalid_client	Client authentication failed (for example, unknown client, no client authentication included, or unsupported authentication method). The Authorization Server may return an HTTP 401 (Unauthorized) status code to indicate which HTTP authentication schemes are supported. If the client attempted to authenticate via the Authorization request header field, the Authorization Server must respond with an HTTP 401 (Unauthorized) status code, and include the WWW-Authenticate response header field matching the authentication scheme used by the client.	`HTTP/1.1 400 Bad Request`
		`Content-Type: application/json`
		`Cache-Control: no-store`
		`{`
		`"error":"invalid_client"_`
		`}`

unauthorized _client	The client is not authorized to request an authorization code using this method.	`HTTP/1.1 401 Unauthorized`
unauthorized _owner	Invalid Resource owner credential in the authorization request	`HTTP/1.1 401 Unauthorized`
insufficient_ scope	Insufficient scope in the authorization request	`HTTP/1.1 403 Forbidden`
invalid_token	Invalid Resource owner's credential the authorization request	`HTTP/1.1 401 Unauthorized`
invalid_token	Duplicated authorization code in the authorize request	`HTTP/1.1 401 Unauthorized`
insufficient_ scope	Invalid scope in the refresh token request	`HTTP/1.1 403 Forbidden`
invalid_token	Discarded refresh token in the refresh token request	`HTTP/1.1 401 Unauthorized`

invalid_token	No Authenticate header When using MAC-type Access Token to access resource	`HTTP/1.1 401 Unauthorized`
		`WWW-Authenticate: MAC`
		`error="invalid_token",`

invalid_token	Invalid MAC-type Access Token When accessing resource	`HTTP/1.1 401 Unauthorized` `WWW-Authenticate: MAC`

Table 1–34 (Cont.) Exception Scenarios

Type	Error	Response

invalid_token	No Authenticate header When using Bearer-type Access Token to access resource	`HTTP/1.1 401 Unauthorized`
		`WWW-Authenticate:`
		`error="invalid_token",`

invalid_token	Invalid Bearer-type Access Token when accessing resource	`HTTP/1.1 401 Unauthorized`
		`WWW-Authenticate:`
		`error="invalid_token",`

insufficient_ scope	The request requires higher privileges (scope) than provided by the access token	`HTTP/1.1 403 Forbidden`

access denied	The Resource owner or Authorization Server denied the request.	`HTTP/1.1 302 Found`
		`Location:`
		`https://client.example.com/cb?error=access_ denied&state=xyz`

unsupported _response_ type	The Authorization Server does not support obtaining an authorization code using this method.	`HTTP/1.1 302 Found`
		`Location:`
		`https://client.example.com/cb?error=unsuppor ted_response_type`

unsupported _grant_type	The Authorization Grant type is not supported by the Authorization Server.	`HTTP/1.1 400`
		`Location:`
		`https://client.example.com/cb?error=unsuppor ted_grant_type`

invalid_scope	The requested scope is invalid, unknown, or malformed.	`HTTP/1.1 302 Found`
		`Location:`
		`https://client.example.com/cb?error=invalid_ scope`

server_error	The Authorization Server encountered an unexpected condition which prevented it from fulfilling the request.	`HTTP/1.1 400`
		`error="server_error",`
		
		The HTTP response code for server_error depends on which endpoint is responding.
		The Authorization and Grant endpoints return 302 error responses as defined by the OAuth specification.
		The Services Gatekeeper Token endpoint returns a 400 error response.

temporarily_ unavailable	The Authorization Server is currently unable to handle the request due to a temporary overloading or maintenance of the server.	Not Supported

Table 1–34 (Cont.) Exception Scenarios

Type	Error	Response

other	Undetermined	`HTTP/1.1 500`
		`Content-Type: application/json`
		`Cache-Control: no-store`
		
		`{`
		`"error":"500"`
		`}`

Interaction of OAuth and Services Gatekeeper SLAs

In Services Gatekeeper, OAuth is used as an additional security mechanism. This lets you use SLA and policy control features in conjunction with OAuth.

Layering Policies and Precedence

An application instanceId defined in Services Gatekeeper can be mapped to a client_id defined in the OAuth specification, seamlessly integrating regular SLA functionality. Because OAuth is enforced as a security mechanism, OAuth security is enforced before a Services Gatekeeper SLA.

For additional information, see *Accounts and SLAs Guide*.