Oracle® Communications Services Gatekeeper

OAuth Guide

Release 6.1

E64624-01

September 2016

Oracle Communications Services Gatekeeper OAuth Guide, Release 6.1

E64624-01

Copyright © 2012, 2016, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Preface vii

Audience vii
Documentation Accessibility vii
Related Documents vii

1 Using OAuth With Services Gatekeeper

About Services Gatekeeper Support for OAuth Authentication Server 1-1
Using SAML Assertions to Access Resources 1-1
Understanding OAuth 2.0 Concepts 1-2
Understanding OAuth Terminology 1-2
About the OAuth/Services Gatekeeper Entities and Their Relationships 1-4
About the OAuth Protocol Endpoints 1-4
Understanding How Services Gatekeeper Works with OAuth 1-5
OAuth Component to Services Gatekeeper Component Mapping 1-5
Understanding the OAuth Endpoints 1-6
Authentication of Network Flows 1-7
Authenticating RESTful APIs Acting on Application-Initiated Traffic 1-7
Authenticating REST APIs Acting on Network-Initiated Traffic 1-8
Mapping a Resource to a Services Gatekeeper Method 1-9
Securing Resources with Multiple Owners 1-10
Support For SAML Assertions 1-11
Support for Anonymous Customer References 1-12
Accessing the OAuth Log Messages 1-13
Understanding OAuth Specification Compliance 1-13
Supported Communication Services 1-13
Supported OAuth Server Roles 1-13
Supported Authorization Grant Types 1-13
Extension Grant Flows Enabled Through Supported Grant Types 1-13
Supported Token Types 1-14
Supported Client Profiles 1-14
OAuth Flows Supported by Services Gatekeeper 1-14
Authorization Code Grant 1-14
Implicit Grant 1-15
Refresh Token Grant 1-16
Supported URIs (Subscribers) 1-17

2 Protecting Services Gatekeeper Resources with OAuth

Managing OAuth Resources 2-1
Understanding Resource Mapping 2-1
Understanding the Services Gatekeeper Resource Server 2-1
Understanding the Services Gatekeeper Authorization Server 2-2
Understanding the Services Gatekeeper Authentication Server 2-2
Provisioning Mapped Resources 2-2
Managing Clients 2-2
Mapping Resources to Resource Owners 2-3
Authenticating Subscribers 2-3
About the MBeans Used to Provide OAuth Functionality 2-3
Understanding OAuth EAR Files 2-3
EDRs and Alarms 2-3
Deploying and Configuring OAuth Functionality 2-4
Configuring OAuth 2-4
Creating Protected Resources 2-4
Protecting RESTful Communication Services 2-4
Protecting Subscription Resources 2-5
Configuring Authentication 2-5
Using the Default Subscriber Manager 2-5
Using Delegated Authentication 2-5
Configuring Resource Rules to Protect Resources 2-5
Creating Individual Resource Owners 2-5
Creating a Resource Rules File Using Regular Expressions 2-6
Uploading the Resource Rules to Services Gatekeeper 2-6
Configuring Clients to Protect Access to Resources 2-6
Configuring SAML (Optional) 2-7
Protecting Resources in a Custom Communication Service 2-7
Example: Protecting the OneAPI Payment Service with OAuth 2-8
Steps to Protecting the OneAPI Payment Service with OAuth 2-8
Adding a Client in Services Gatekeeper 2-8
Configuring the Authentication URL 2-9
Adding One API Payment Communication Service as an OAuth resource 2-9
Adding a New Subscriber 2-9
Assigning the Resource to the Subscriber to Act as Resource owner 2-9
Understanding the OAuth Resource Format 2-10
Resource Representation Example 2-12

3 Monitoring OAuth Services in Service Gatekeeper

Understanding OAuth Runtime Actions 3-1
Issuing OAuth Tokens 3-1
Default Authentication and Authorization 3-1
Authorization for Group URIs 3-1
Understanding Token Validation 3-2
Managing Tokens 3-2
EDRs Generated by the OAuth Service 3-2
OAuth/Services Gatekeeper Errors and Exceptions 3-4

4 Developing Services Gatekeeper Services Using OAuth

Understanding How to Apply SAML Tokens 4-1
Understanding Token Request Messages 4-1
Understanding Token Response Messages 4-2
Understanding SAML Assertion Validation Messages 4-3
Understanding OAuth Customization 4-4
Implementing a Third-Party Authentication Service 4-4
Authentication Process Flow 4-5
Creating an OAuth Interceptor 4-7
Examples: Using a Custom OAuth Interceptor to Retrieve OAuth Information 4-8
Integrating a Third-Party Subscriber Repository 4-9
Creating an OAuth Extension Handler for New Credentials 4-9
Customizing OAuth Resource Grant Tests 4-9
OAuth Application Developer Guide 4-9
Interacting with the Services Gatekeeper OAuth Service 4-10
OAuth Access Flow In Services Gatekeeper 4-10

Preface

This guide explains how to use the Open Authorization Protocol (OAuth) features with Oracle Communications Services Gatekeeper.

Audience

This document is intended for developers who create applications for use with Services Gatekeeper that allow access to protected resources.

This includes:

- Third-party application developers who want to integrate telephony-based functionality into their products
- Operator-based system developers who want to extend the functionality of Services Gatekeeper or to integrate it with Partner Relationship Management (PRM) or Operations Support Systems (OSS) tools

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.

Related Documents

For more information, see the following documents in the Services Gatekeeper documentation:

- *Oracle Communications Services Gatekeeper Application Developer's Guide*
- *Oracle Communications Services Gatekeeper Security Guide*
- *Oracle Communications Services Gatekeeper Communication Service Reference Guide*
- *Oracle Communications Services Gatekeeper OAM Java API Reference*
- *Oracle Communications Services Gatekeeper Platform Test Environment User's Guide*

1 Using OAuth With Services Gatekeeper

This chapter provides an overview of how Oracle Communications Services Gatekeeper uses the Open Authorization Protocol v2.0 (OAuth) to protect resources.

About Services Gatekeeper Support for OAuth Authentication Server

The Services Gatekeeper OAuth implementation allows service providers to offer authorized third-party applications access to protected resource owner resources over HTTP.

OAuth is an additional layer of security, in addition to the SLAs and policy control security features that Services Gatekeeper already offers. These security features all work together.

You map the **instanceId**s defined in Services Gatekeeper to the **client_id**s defined in OAuth to seamlessly integrate Oauth resources into SLAs. Because OAuth is enforced as a security mechanism, OAuth security is enforced before a Services Gatekeeper SLA.

For more information on the communication services that support OAuth security, see the *Services Gatekeeper Application Developer's Guide*.

For traditional communication services, OAuth security mechanism can only be enforced for REST-based APIs deployed on Services Gatekeeper. For REST and SOAP based APIs created in the Partner and API Management Portal, both SOAP and REST APIs can use OAuth.

OAuth security supports Transport Layer Security (TLS) if required by a client. See "Configuring OAuth" for the required configuration setting.

See "OAuth Compliance" in *Services Gatekeeper Statement of Compliance* for the exact OAuth specifications supported.

Using SAML Assertions to Access Resources

You can use Security Assertion Markup Language (SAML) credentials to gain access to resources protected by OAuth 2.0 in Services Gatekeeper. This enables you to combine the centralized identity management of Access Manager and an existing trust relationship with identity management.

You can create single sign on (SSO) features that provide your subscribers with one authorized SAML token (federated identity) for use in accessing multiple third-party resources that you support. Services Gatekeeper acts as the SSO authorization server for SSOs that you create using an identity management system such as Oracle Identity Management (OIM). OIM is a subset of the Oracle Fusion Middleware Identity and Access Management product.

See "Support For SAML Assertions", "Understanding the Services Gatekeeper Authorization Server", and "Configuring SAML (Optional)" for details.

The Oracle Fusion Middleware WebLogic server provides the underlying support for SAML capabilities. For details see "Security Assertion Markup Language (SAML)" in *Oracle Fusion Middleware Understanding Security for Oracle WebLogic Server 12c* here:

`http://docs.oracle.com/cd/E24329_01/web.1211/e24446/security.htm#INTRO232`

Understanding OAuth 2.0 Concepts

OAuth is an open standard for authorization. It allows your subscribers to share their private resources with a third party without having to provide their own security credentials. These resources could be photos, videos, contact lists, location and billing capability, and so on, and are usually stored with another service provider. For example, photos stored on a dedicated photo Web storage site.

OAuth does this by granting requesting (client) applications a token, once access is approved by the resource owner. Each token grants access to a specific resource for a specific period. The requesting application uses the token for access to resources stored with another service provider, instead of the owner's credentials.

A resource can be:

- A single file such as a photo or video.
- Access to a website, such as the services of a video editing website.
- Personal information such as their location or billing capability.

Services Gatekeeper can also act as a SAML authorization server for applications (service providers) that require it. Services Gatekeeper accepts SAML assertions from Oracle Identity Manager (as the identity provider), and returns SAML access tokens to it. Application can use these SAML access tokens for SSO authentication or to provide enhanced security profiles required for using derived values such as signatures or hash-method authentication codes (HMACs). SAML access tokens are also useful for client-server integration scenarios where the subscriber may not be present.

Understanding OAuth Terminology

Table 1–1 lists OAuth terminology and definitions.

Table 1–1 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).
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.
Authentication Server	Server that validates resource owner identity (defined by Services Gatekeeper).
Authorization Endpoint	Used to obtain authorization from the resource owner using user-agent redirection.

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

Term	**Definition**
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.
Authorization Server	Server that issues authorization codes and 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.
Client Identifier	A unique string representing the registration information provided by the client.
Custom Subscriber Manager	Component that authenticates resource owner's username and password with a custom identity store (such as LDAP).
Delegated Authentication	Authentication mode supported by Services Gatekeeper to integrate with 3rd party authentication systems.
Grant Endpoint	URI supported by Services Gatekeeper to post the authentication result to issue the authorization code (defined by Services Gatekeeper).
Group Owner	Owner of the Group URI. Issues authorization token on behalf of the group members.
Group URI	URI that represents a group of Resource Owners.
OAuth	Open Authorization Protocol
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
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.
Refresh Token	Refresh tokens are credentials used to re-obtain access tokens.
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:)
Resource Server	Server that hosts protected resources and validates access token during resource access.
scopeId	Unique string that identifies a resource and used as part of scope-token by application client.
Subscriber Manager	Component in Services Gatekeeper to validate subscribers provisioned in Services Gatekeeper database.

About the OAuth/Services Gatekeeper Entities and Their Relationships

Figure 1–1 OAuth Flow and Entity Relationships

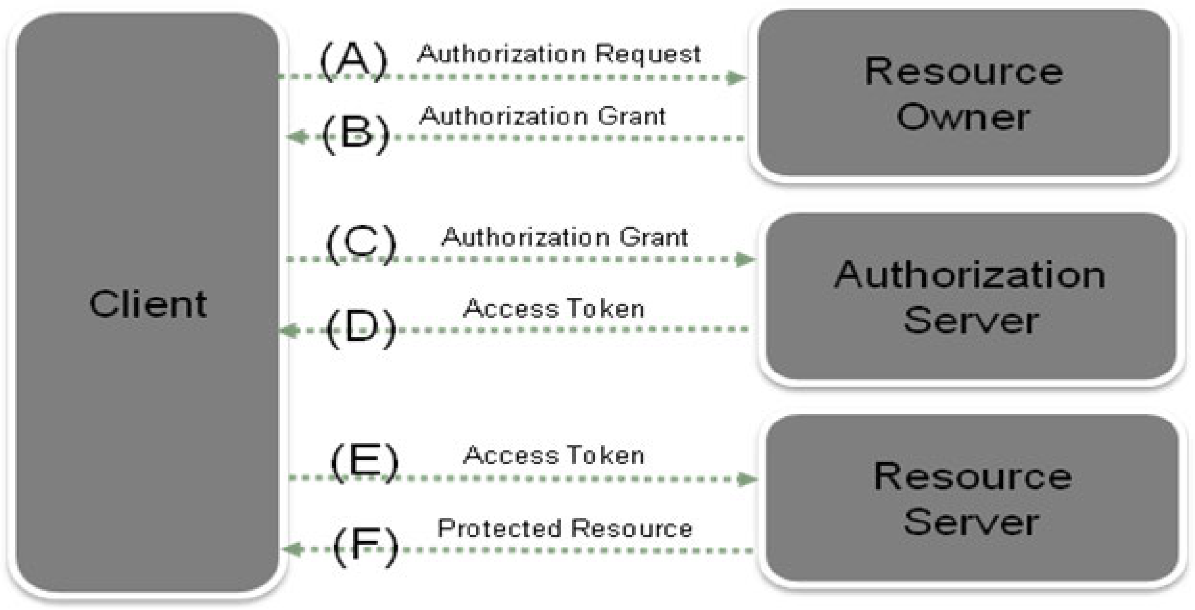


Figure 1–1 shows an example OAuth protocol flow for a resource access request:

(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 using the Authorization Server as an intermediary.
(B) The client receives an authorization grant, including 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 resending the access token.
(F) The resource server validates the access token, and if valid, serves the request.

About the OAuth Protocol Endpoints

The OAuth 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 using the resource owner user-agent.
- **Authorization Endpoint**: The authorization endpoint interacts 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 client uses the token endpoint 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).

Understanding How Services Gatekeeper Works with OAuth

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

OAuth Component to Services Gatekeeper Component Mapping

OAuth defines the following terms that map to the Services Gatekeeper concepts listed:

- **resource**: Defined by the OAuth specification. In Services Gatekeeper, a resource maps to the API(s) and methods protected by the OAuth token. This is a combination of communication service application-facing interface (Plug-in) name, method name, token expiration period, parameters, and the subResource. resource uniquely identified by **scopeId**.

  For more information, see "Understanding Resource Mapping".

- **scope** request parameter: Defined by the OAuth 2.0 specification and specified by the authorization server. Determines the limits of an OAuth token. A scope parameter defining these limits is submitted as part of obtaining an authorization grant.

  The format of the `scope` is defined in the OAuth 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>=<[&<param>>=<value]*]+
  ```

  Where:

  `scopeId` identifies a resource.

  `param` is a custom parameter defined as part of resource.

  `value` is the value for the 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 within the 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 protects and authorizes 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 applies 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 map a Services Gatekeeper Application Instance Id to an OAuth application client ID during client creation with the **OAuthclientMBean**. See "Managing Clients" for details.

Services Gatekeeper relies on the authentication endpoint to validate the resource owner. In the Services Gatekeeper default implementation of the authentication endpoint, the validation is handled by the Subscriber Manager. See "Mapping Resources to Resource Owners" for more information.

Understanding the OAuth Endpoints

Service Gatekeeper supports the standard endpoints defined in the OAuth 2.0 specification.

OAuth generates these endpoints that Services Gatekeeper uses:

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

  `https://`*Gatekeeper_server_IP:port*`/oauth2/authorize`
- **Token endpoint**: By default, the token endpoint is configured to the following URL:

  `https://`*Gatekeeper_server_IP: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://`*Gatekeeper_server_IP:port*`/oauth2/auth.jsp`

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

  OauthService.OauthCommonMBean.AuthenticationURL
- **Grant Endpoint:** The callback URI supported by Services Gatekeeper to process the successful authenticated requests and issue authorization grants 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 requester to a redirect URI provided by the application along with the authorization code.

  By default, the grant endpoint uses the following URL:

  `https://`*Gatekeeper_server_IP*`:`*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 "Implementing a Third-Party Authentication Service".

Authentication of Network Flows

Services Gatekeeper provides OAuth security for RESTful Web services required by application-initiated or network-initiated traffic when the requests call API management APIs.

Authenticating RESTful APIs Acting on Application-Initiated Traffic

OAuth security for application-initiated traffic operates in the following way:

1. A network operator creates an API in Partner Manager portal.

   At that time, the network operator does the following:

   a. Selects to apply OAuth security on the API.
   b. Specifies that the API is to be called when the traffic flows from the application to the network.
   c. Specifies REST API as the exposure for the interface in the application-initiated flow.
   d. Provides valid URIs for the authorization endpoint, token endpoint, and redirection endpoint as the values for **Authorization URI**, **Token URI**, and **Client Redirect URI**.
   e. Configures a resource list consisting of API methods that the operator exposes for use by applications valid for traffic flows from the application to the network.

      Each method exposed by the operator is termed a resource. Services Gatekeeper identifies each resource using the API identifier and the method name.

      Every method selected by the operator comes under the protection of OAuth2 security.

2. An authorized application developer subscribes to this API when the developer creates an application in Partner portal.
3. When the network operator approves the application in Partner Manager portal, Services Gatekeeper generates an OAuth2 access token which is stored with the application data and used by the application during its active state.

   Note: The access token is valid for a designated period. The application developer can refresh the access token in Partner portal.

4. When the application is in use, the API is called. The application sends the OAuth2 access token in the **Authorization** header field of the message it sends to the network.
5. Services Gatekeeper checks the OAuth2 token in that message and does one of the following:
   - If the token is valid, Services Gatekeeper adds the relevant data about the end user such as the application instance id, the application group name, the service provider and the service provider group to the current subject.
   - If the token is invalid, the authorization fails for that message.

Authenticating REST APIs Acting on Network-Initiated Traffic

OAuth security for network-initiated traffic operates in the following way:

1. A network operator creates an API in Partner Manager portal. The network operator does the following:
   a. Selects to apply OAuth security on the API.
   b. Specifies that the API is to be called when the traffic flows from the network to the application.
   c. Specifies REST API as the exposure for the interface in the network-initiated flow.

   d. Provides valid URIs for the authorization endpoint, token endpoint, and redirection endpoint as the values for **Authorization URI**, **Token URI**, and **Client Redirect URI**.

   e. Configures a resource list consisting of API methods that the operator exposes for use by applications valid for traffic flows from the network to the application.

      Each method exposed by the operator is termed a resource. Services Gatekeeper identifies each resource using the API identifier and the method name.

      Every method selected by the operator comes under the protection of OAuth2 security.

2. An authorized application developer subscribes to this API when the developer creates an application in Partner portal.

3. When the network operator approves the application in Partner Manager portal, Services Gatekeeper generates an OAuth2 access token which is stored with the application data and used by the application during its active state.

   Note: The access token is valid for a designated period. The application developer can refresh the access token in Partner portal.

4. When the application is in use, the API is called. The message that the network sends to the application contains the following information:

   - In the **Proxy-Authorization:** header field, the northbound authorization information. For example:

     ```
     Proxy-Authorization: Basic ZGFmcGFydG5lcl9hcHAxOmRhZnBhcnRuZXJfYXBwMQ==
     ```

   - In the **Authorization:** header field, the southbound OAuth2 token access information.

     ```
     Authorization: Bearer 6de54688-d10b-43ad-a994-0be2c978fe2d
     ```

5. When the traffic is initiated by the network, Services Gatekeeper does not attempt to validate the credentials.

   Before sending the message out, Services Gatekeeper does the following:

   - It retains the **Authorization** field containing the OAuth2 token access information.
   - It removes the **Proxy-Authorization** header field.

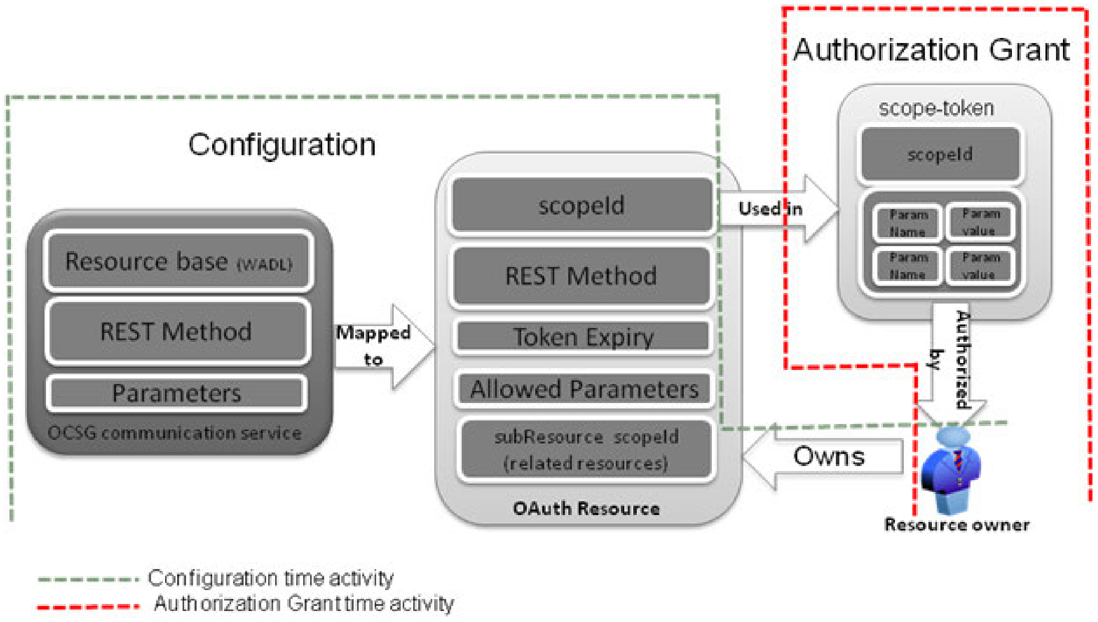
Mapping a Resource to a Services Gatekeeper Method

You can use OAuth token-based security to protect any RESTful communication service, including a customized communication service 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 "Managing OAuth Resources".

The scope (defined in the OAuth 2.0 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. By default the OAuth interceptor allows all resources to pass. You can also filter these resources to meet the needs of your implementation. If the OAuth interceptor does not meet your implementations's needs, you can replace it with a custom interceptor. For details on modifying the OAuth interceptor or creating a custom interceptor see "*Using Service Interceptor to Manipulate Requests*" in *Services Gatekeeper Extension Developer's Guide*.

See "OAuth Component to Services Gatekeeper Component Mapping" for more details on the `scope`.

Securing Resources with Multiple 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, you can create a group of resource owners and associate them with a group URI. You then create a group owner to issue an authorization grant on behalf of the group members. The group owner is represented by an **address** (Group URI), **loginId** and **password** much like a resource owner. 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 relationships between resources, resource owner(s), groups, group URI and group owner are illustrated in Figure 1–3:

Figure 1–3 Resource and Group Relationships

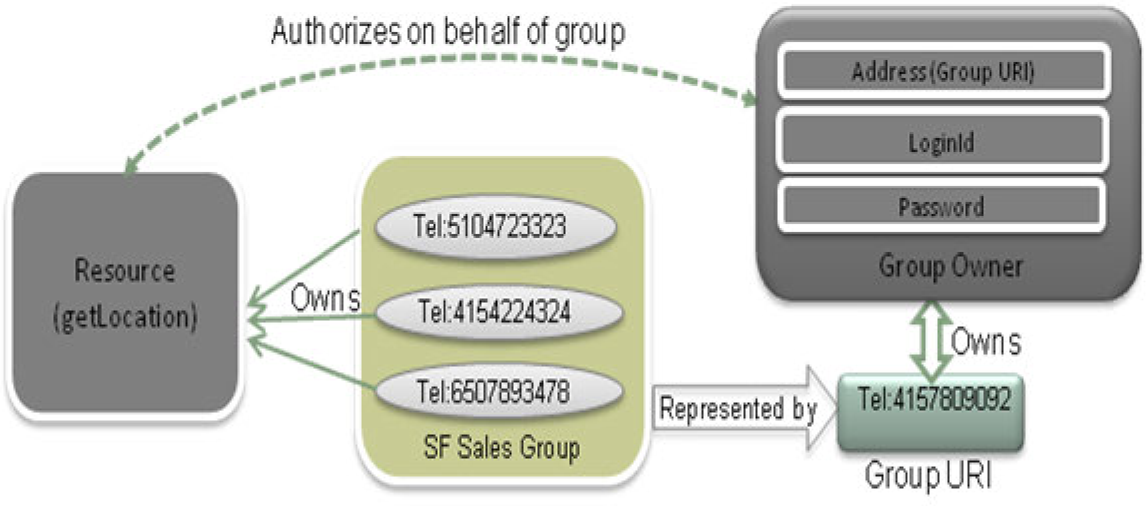


Figure 1–3 show Tel: URIs, but you can also use SIP: URIs.

Use these general steps to protect an API that takes group URIs:

1. Create a group with a group URI using the Parlay X 3.0 **AddressListManagement**.
2. Add members to the group using the Parlay X 3.0 **AddressListManagement**.
3. Create a group owner related to this group URI and password for the group owner.
4. Communicate the group URI and password to a group owner (outside the scope of Services Gatekeeper).
5. The group owner issues an authorization grant to an application to use the member URI as part of API method that requires multiple URIs.
6. Subscribers access the 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 **AddressListManagement** communication service to create groups. See the 3GPP specification for more information about address list management:.

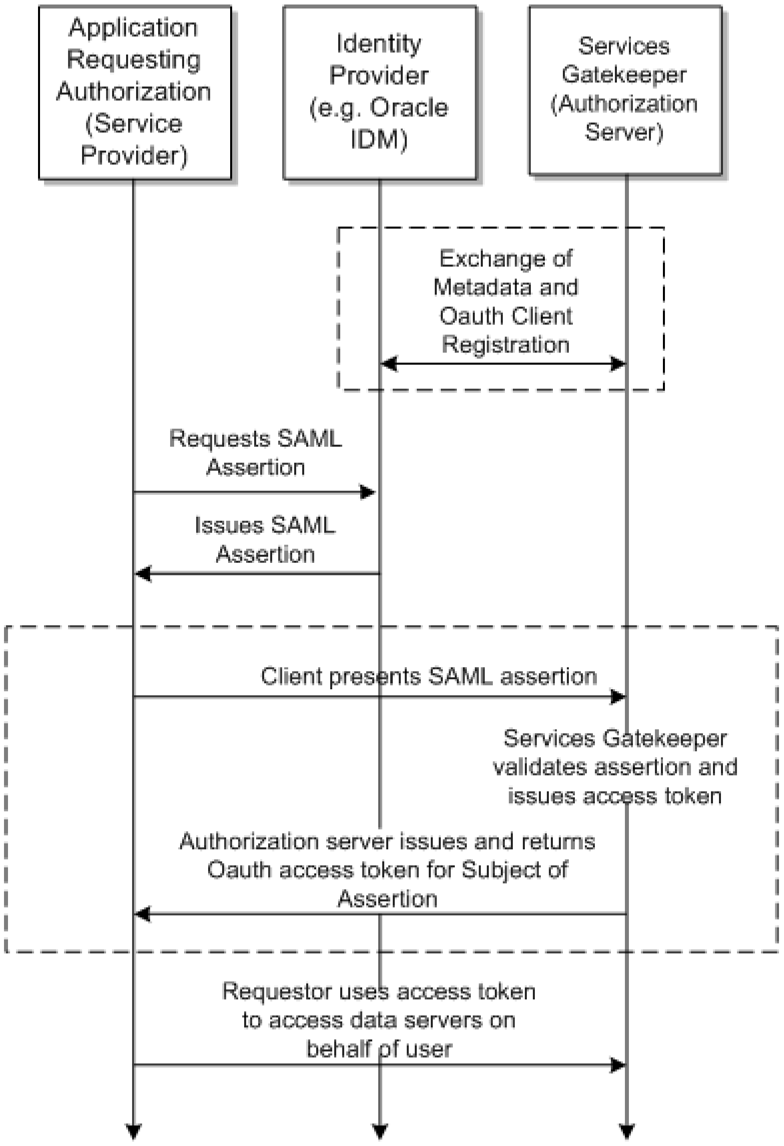
`http://www.3gpp.org/ftp/specs/archive/29_series/29.199-13/29199-13-702.zip`

Support For SAML Assertions

In order to use SAML assertions, the application must have a trusted relationship with the identity provider, and be able to request end user authorization assertions. Services Gatekeeper is pre-configured to trust the identity provider with accounts and policies for the application.

Figure 1–4 shows how traffic flows Services Gatekeeper interacts with the identity provider (Oracle IDM) to support an SSO application. First the identity provider and Services Gatekeeper (as the authorization server) exchange metadata and the OAuth client registration information. Once that trusted relationship is set up, the service provider can request a SAML assertion from the identity provider. The service provider then presents the SAML assertion together with an application identifier to Services Gatekeeper as the authorization server. Services Gatekeeper then exchanges the end user authentication assertion and application identity for an access token. The access token can then be used as evidence of authentication when accessing 3rd party resources through Services Gatekeeper. You can also apply any policies associated with the API access.

Figure 1–4 SSO Traffic Flow


Application Requesting Authorization (Service Provider)
Identity Provider (e.g. Oracle IDM)
Services Gatekeeper (Authorization Server)
Exchange of Metadata and Oauth Client Registration
Requests SAML Assertion
Issues SAML Assertion
Client presents SAML assertion
Services Gatekeeper validates assertion and issues access token
Authorization server issues and returns Oauth access token for Subject of Assertion
Requestor uses access token to access data servers on behalf of user

See "Understanding the Services Gatekeeper Authorization Server" and "Configuring SAML (Optional)" for details on configuring SAML.

Support for Anonymous Customer References

In addition to basic and session ID authentication, Services Gatekeeper also supports OAuth authentication of applications requesting or querying Anonymous Customer

References (ACR) identifiers. Services Gatekeeper can use OAuth to provide subscribers SSO-like access to resources such as photos or websites.

Services Gatekeeper supports allowing OAuth access to subscriber resources using ACRs. Requests can use a valid **accessToken** to retrieve or update ACR information when submitted. The Services Gatekeeper OAuth Interceptor uses the provided **accessToken** to confirm a subscriber's identity before releasing ACR information.

For more information adding anonymous customer references, see "Adding RESTful Anonymous Customer Reference Support" in *Services Gatekeeper Application Developer's Guide*.

Accessing the OAuth Log Messages

OAuth stores logged output messages in the *Middleware_home*/**user_projects/domains/***domain_name***/oauth2_log/oauth2.log** file.

Understanding OAuth Specification Compliance

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

For more information, see "OAuth Compliance" in *Services Gatekeeper Statement of Compliance* for the specifications that Services Gatekeeper OAuth supports.

Supported Communication Services

The OAuth security mechanism can be used with the REST-based communication services provided with Services Gatekeeper, and any custom REST-based API that you deploy.

See "Part II Creating Applications Using the RESTful Interface" in *Services Gatekeeper Application Developer's Guide* for the list of REST-based communication services.

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 customizing authorization flow by supporting the custom Endpoints described in "Understanding the OAuth Endpoints".

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.

This delegated authentication is only supported with the authorization code grant type.

See "Implementing a Third-Party Authentication Service" for more information.

Supported Token Types

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

- **bearer**
- **MAC**
- **saml-bearer**

Supported Client Profiles

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

OAuth Flows Supported by Services Gatekeeper

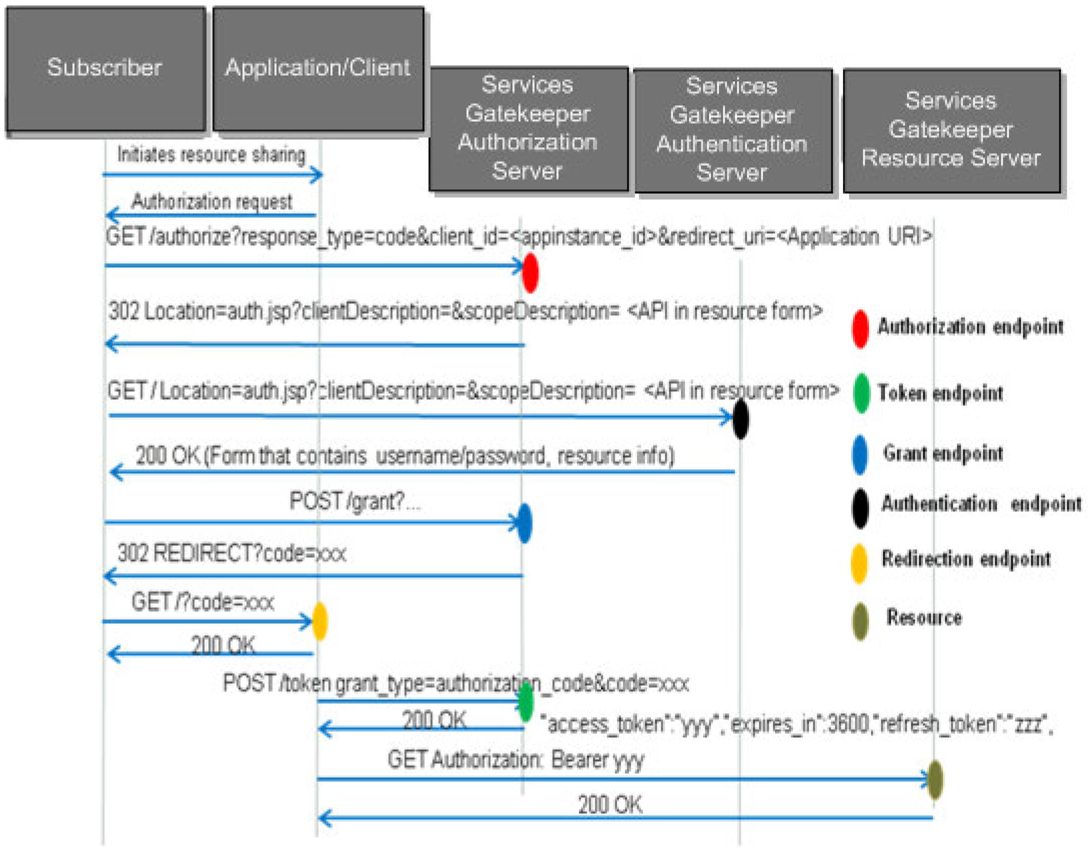
The following sections illustrate the OAuth authorization grant flows that Services Gatekeeper supports:

- Authorization Code Grant
- Implicit Grant
- Refresh Token Grant

Authorization Code Grant

Figure 1–5 shows an OAuth authorization code grant sequence flow diagram.

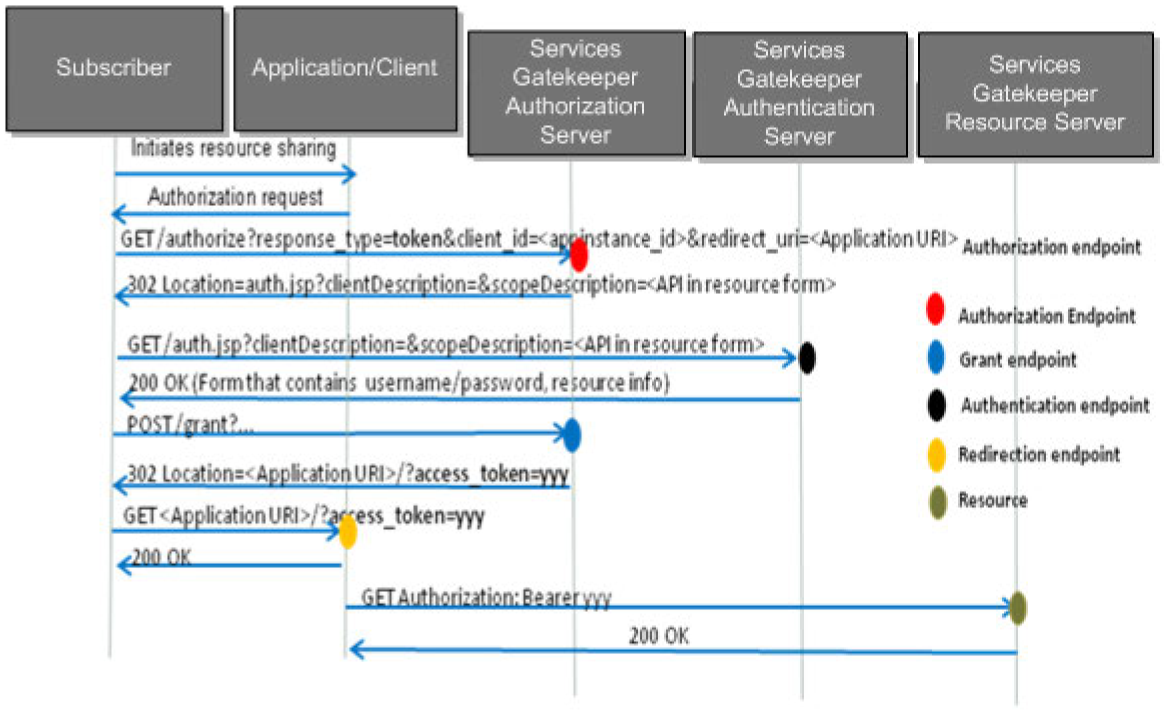
Figure 1–5 OAuth Authorization Code Grant


Subscriber
Application/Client
Services Gatekeeper Authorization Server
Services Gatekeeper Authentication Server
Services Gatekeeper Resource Server
Initiates resource sharing
Authorization request
GET /authorize?response_type=code&client_id=<appinstance_id>&redirect_uri=<Application URI>
302 Location=auth.jsp?clientDescription=&scopeDescription= <API in resource form>
GET / Location=auth.jsp?clientDescription=&scopeDescription= <API in resource form>
200 OK (Form that contains username/password, resource info)
POST /grant?...
302 REDIRECT?code=xxx
GET /?code=xxx
200 OK
POST /token grant_type=authorization_code&code=xxx
200 OK
"access_token":"yyy","expires_in":3600,"refresh_token":"zzz",
GET Authorization: Bearer yyy
200 OK
Authorization endpoint
Token endpoint
Grant endpoint
Authentication endpoint
Redirection endpoint
Resource

Implicit Grant

Figure 1–6 shows an Oauth implicit grant sequence flow diagram.

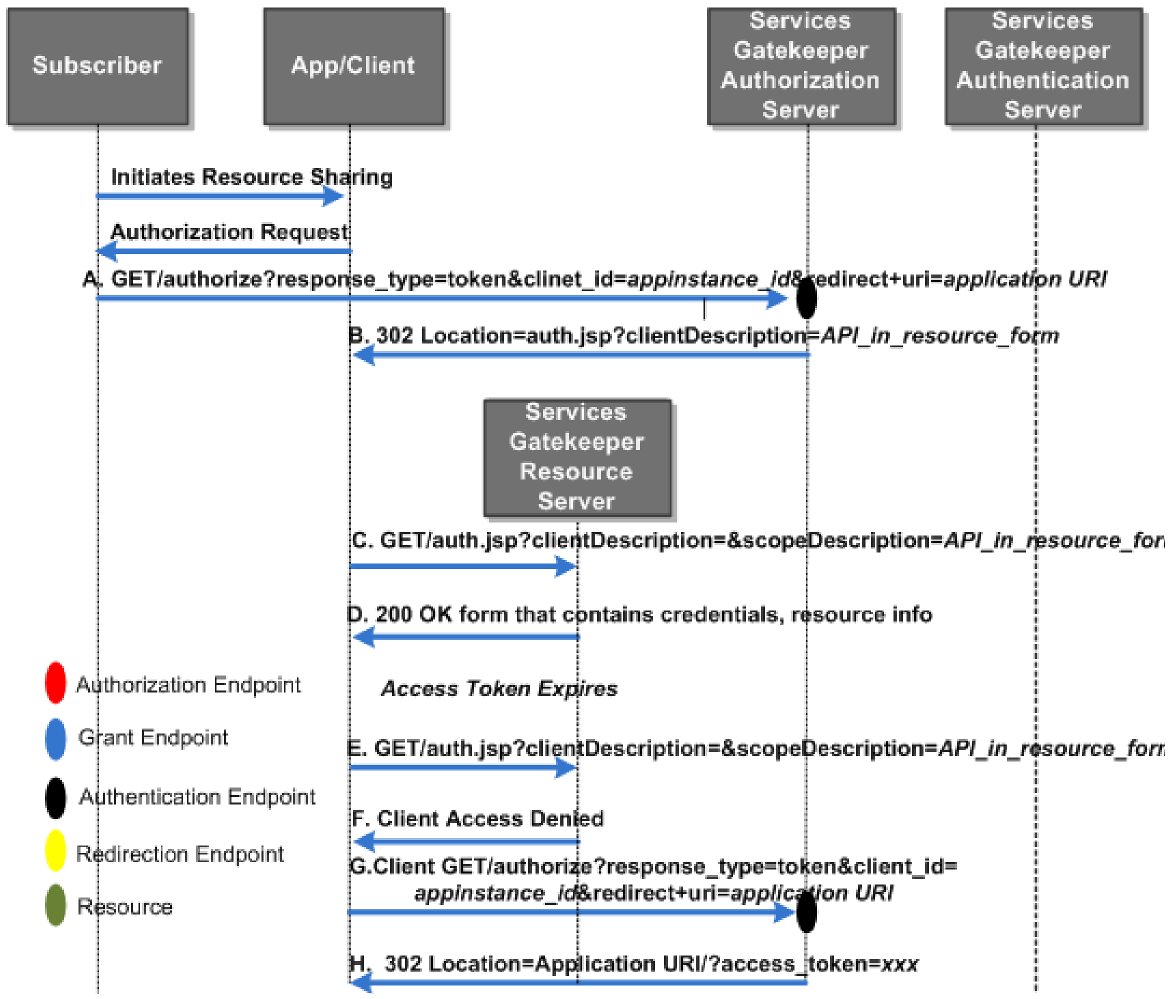
Figure 1–6 OAuth Implicit Grant


Subscriber
Application/Client
Services Gatekeeper Authorization Server
Services Gatekeeper Authentication Server
Services Gatekeeper Resource Server
Initiates resource sharing
Authorization request
GET /authorize?response_type=token&client_id=<appinstance_id>&redirect_uri=<Application URI>
Authorization endpoint
302 Location=auth.jsp?clientDescription=&scopeDescription=<API in resource form>
GET /auth.jsp?clientDescription=&scopeDescription=<API in resource form>
200 OK (Form that contains username/password, resource info)
POST /grant?...
302 Location=<Application URI>/?access_token=yyy
GET <Application URI>/?access_token=yyy
200 OK
GET Authorization: Bearer yyy
200 OK
Authorization Endpoint
Grant endpoint
Authentication endpoint
Redirection endpoint
Resource

Refresh Token Grant

Figure 1–7 shows a refresh token grant flow diagram. If the client has way of recognizing when the access token expires (smart client), it skips steps E and F in this diagram. Clients without this knowledge (dumb clients) execute steps E and F.

Figure 1–7 OAuth RefreshToken Grant


Subscriber
App/Client
Services Gatekeeper Authorization Server
Services Gatekeeper Authentication Server
Initiates Resource Sharing
Authorization Request
A. GET/authorize?response_type=token&clinet_id=appinstance_id&redirect+uri=application URI
B. 302 Location=auth.jsp?clientDescription=API_in_resource_form
Services Gatekeeper Resource Server
C. GET/auth.jsp?clientDescription=&scopeDescription=API_in_resource_for
D. 200 OK form that contains credentials, resource info
Access Token Expires
E. GET/auth.jsp?clientDescription=&scopeDescription=API_in_resource_forr
F. Client Access Denied
G.Client GET/authorize?response_type=token&client_id= appinstance_id&redirect+uri=application URI
H. 302 Location=Application URI/?access_token=xxx
Authorization Endpoint
Grant Endpoint
Authentication Endpoint
Redirection Endpoint
Resource

Supported URIs (Subscribers)

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

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

2 Protecting Services Gatekeeper Resources with OAuth

This chapter explains how to protect Oracle Communications Services Gatekeeper resources with the Open Authorization Protocol (OAuth). It starts with information you need to understand about the OAuth technology, then explains the deployment procedure, and finally lists some examples to help you understand the process.

Managing OAuth Resources

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

Understanding Resource Mapping

You protect Service Gatekeeper communication services by mapping those communication services to OAuth resources.

In order to protect an API with OAuth, you model the API method as an OAuth resource, one API to one resource. Resource mapping involves creating a unique **scopeId** to represent the resource, and then mapping the **scopeId** to the API and method names.

You can also define additional context parameters for the resource to use as targets for custom interceptors. These parameters may or may not map directly to the API method parameters.

You can create multiple resources for a single communication service. In this case, 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). Resource owners then issue authorization grants to the **scopeId**s (resources) that they own.

See "Understanding the OAuth Resource Format" for details on the resource format and a resource example.

Understanding the Services Gatekeeper Resource Server

As an OAuth Resource Server, Services Gatekeeper manages the protected resources contained within a service provider's network and accepts and responds to third-party application requests for access to protected resources.

In Services Gatekeeper, a resource is considered to be a communication service method. Resources can have subresources, each of which is represented by a method derived from the resource. 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.

Understanding the Services Gatekeeper Authorization Server

As an OAuth Authorization Server, Services Gatekeeper obtains a subscriber's permission for access to protected resources when an application makes a request. Services Gatekeeper issues a token to the client application, which is used to access protected resources without requiring the resource owner to provide actual credentials.

When a client application requests access to a protected resource, the Services Gatekeeper REST handler initially checks the request, ensuring that the body contains the needed authorization information. If the request is valid, Services Gatekeeper forwards the request onto the proper communication service. An OAuth interceptor verifies that the token contained in the request is valid before completing the request. Services Gatekeeper then sends a response back to the client application.

Services Gatekeeper can also act as a SAML identity manager that accepts SAML token requests and grants the corresponding SAML tokens. You can use these tokens to supply permission to use resources protected by OAuth. You also have the ability to grant access to resources with federated identities such as single sign-ons (SSOs). You create the SSOs using an identity manager such as Oracle Fusion Middleware Identity and Access Management product (which contains Oracle Identity Manager), or another identity access management product. The SSO authorizations then allow the subscriber access to all resources protected by the SSO.

Understanding the Services Gatekeeper Authentication Server

As an Authentication Server, Services Gatekeeper maintains a subscriber database used to store the owners of protected resources. OAuth 2.0 resource owners in Services Gatekeeper must be setup as subscribers in the Authentication Server first. The Authentication Server is used to authenticate subscribers, as resource owners, before initiating authorization to protected resources.

The included Authentication Server is intended for OAuth 2.0 demonstration and development purposes.

Provisioning Mapped Resources

Services Gatekeeper contains a JMX interface for uploading resource mapping files. The interface name is included in the **OAuthResourceMBean** MBean, which you can access using MBean browser, such as the WebLogic Administration Console, the Oracle Access Manager (OAM) interface (a part of the Oracle Fusion Middleware Identity and Access Management product), the Services Gatekeeper Platform Test Environment (PTE). You load or retrieve resources using this MBean.

For more information on the fields and methods of **OAuthResourceMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Managing Clients

You can map a client to an existing Services Gatekeeper application **instanceId**. This mapping lets you use existing SLA enforcement with OAuth security. You manage application clients using the **OAuthClientMBean**, which is available from the OAM

WebLogic interface or the PTE. You use this MBean to search for clients or return a list of them. You can also add, remove, or update clients.

For more information on the fields and methods of **OAuthClientMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Mapping Resources to Resource Owners

You must define a resource owner for each resource that Services Gatekeeper protects. You manage resource owners using the **OAuthResourceOwnerMBean**. You use this MBean to add, remove, or update resource owners. For all operations, the format of a **resourceScope** is space separated **scopeId** list.

For more information on the fields and methods of **OAuthResourceOwnerMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Authenticating Subscribers

Services Gatekeeper acts as an authentication server. You create and authenticate subscribers in the Services Gatekeeper database.

To authenticate users, Services Gatekeeper supports a component called the Subscriber Manager which provisions subscribers, authenticates them, and expands **GroupURIs**. You manage subscribers using the **SubscriberMBean** MBean.

For more information on the fields and methods of **SubscriberMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

About the MBeans Used to Provide OAuth Functionality

You administer the Services Gatekeeper OAuth server using these MBeans in *domain_home*/**ocsg/container_services/OAuthService**:

- **OAuthCommonMBean**
- **OAuthClientMBean**
- **OAuthresourceMBean**
- **TokenManagerMBean**

For more information on the fields and methods of these MBeans, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*. For more information about Services Gatekeeper administration, see *Services Gatekeeper System Administrator's Guide*.

Understanding OAuth EAR Files

The OAuth EAR files are deployed as part of the Services Gatekeeper installation.

EDRs and Alarms

The Services Gatekeeper OAuth implementation functions as an application running on the WebLogic application server host. You can configuration EDRs and alarms manually if needed. See the chapter on "Managing and Configuring EDRs, CDRs and Alarms" in *Services Gatekeeper System Administrator's Guide* for more information.

Deploying and Configuring OAuth Functionality

To deploy and configure the Services Gatekeeper OAuth functionality:

1. Confirm that the OAuth EAR files are deployed.
2. Configure OAuth.

   See "Configuring OAuth" for more information.
3. Create the protected resources in Services Gatekeeper.

   See "Configuring Resource Rules to Protect Resources" for more information.
4. Configure the Authentication URL.

   See "Configuring Authentication" for more information.
5. Configure resource rules to protect the usage of your resources.

   See "Configuring Resource Rules to Protect Resources" for more information.
6. Create the client identifier(s) and credentials in Services Gatekeeper that will request access to protected resources.

   See "Configuring Clients to Protect Access to Resources" for more information.

See:

- "Example: Protecting the OneAPI Payment Service with OAuth" for an example of the configuration steps.
- "OAuth Access Flow In Services Gatekeeper" for an example of how Services Gatekeeper accesses a resource.

Configuring OAuth

To configure OAuth, use the methods of the **OAuthCommonMBean** MBean to update (or configure) Services Gatekeeper for use with OAuth.

For more information on the fields and methods of **OAuthCommonMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Creating Protected Resources

Create protected resources by doing the following:

- Protecting RESTful Communication Services
- Protecting Subscription Resources

Protecting RESTful Communication Services

To enable OAuth protection for RESTful communication services (including custom communication services), you map communication services to OAuth resources. You then use the **OAuthResourceMBean** to upload resource mapping to Services Gatekeeper.

For more information on the fields and methods of **OAuthResourceMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Protecting Subscription Resources

Services Gatekeeper uses operations in the **OAuthResourceMBean** to protect subscription resources. See “Services Gatekeeper OAuth 2.0 Authorization and Resource Servers” in *Services Gatekeeper Communication Service Reference Guide*.

Configuring Authentication

In order to grant an authorization code, a resource owner must be authenticated and authorize the scope of the grant. The resource owner can be authenticated using these 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** Mbean. Use the methods of this MBean to configure and maintain subscribers. Access the **SubscriberMBean** MBean through the OAM Console or the PTE Tools MBean browser.

   For more information on the fields and methods of **SubscriberMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

2. Verify that the **AuthenticationURL** property of **OAuthCommonMbean** is set to:

   `https://`*app_server_IP:app_server_port*`/oauth2/auth.jsp`

Using Delegated Authentication

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

See "Implementing a Third-Party Authentication Service" for more information.

See "Understanding the OAuth Resource Format" for details on the resource format.

Configuring Resource Rules to Protect Resources

Protecting resources with resource rules comprises the following tasks:

- Creating Individual Resource Owners
- Creating a Resource Rules File Using Regular Expressions
- Uploading the Resource Rules to Services Gatekeeper

Creating Individual Resource Owners

Create the individual resource owners using the methods of the **OAuthResourceOwnerMBean** MBean. For more information on the fields and methods of this MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Creating a Resource Rules File Using Regular Expressions

You map resource owners to their resources using regular expressions. To do so, you create an XML rules file that defines the resource owner/resource mapping and save this file.

See **resource_rule.xsd** for the schema definition.

The rules file that you create maps resource owner addresses, represented by a regular expression, to a list of one or more resources. The rules file is interpreted when OAuth grants the authorization, and rules must be defined in order of priority from most specific to most general.

In the following example, owner/resource mapping rules are defined with the most specific rules at the beginning of the file, proceeding to the most general rules at the end. Placing a more specific rule, with a regular expression such as `^1390.*$`, at the start of the file ensures that the expected cases are caught and correctly interpreted first. Then the rules with more general regular expressions, such as `^.*$`, toward the end of the file serves as a stop gaps to catch the unexpected or rarer cases.

Example 2–1 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>
```

Uploading the Resource Rules to Services Gatekeeper

Upload the saved resource rules XML file to Services Gatekeeper using the **loadResourceRuleXml** method in **OAuthResourceMBean** MBean.

To retrieve the current rules XML file, you use the **retrieveResourceRuleXml** method.

For more information on the fields and methods of the **OAuthResourceMBean** MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Configuring Clients to Protect Access to Resources

Services Gatekeeper enables you to create an application client to support the authorization code grant type. A client registers with its password and the allowed redirect URI in Services Gatekeeper.

Use the fields and methods of the **OAuthClientMBean** MBean to configure and maintain application clients. You can access this MBean using an MBean browser, such as the OAM Console, WebLogic Administration Console, or the Platform Test Environment (PTE).

For more information on the fields and methods of the **OAuthResourceMBean** MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Configuring SAML (Optional)

This section explains how to configure Services Gatekeeper as a SAML authorization server. It is an optional step.

To configure Services Gatekeeper to use SAML federated identities follow these general steps:

1. Configure Oauth in Services Gatekeeper. See these sections for details:
   - Configuring OAuth
   - Creating Protected Resources
   - Configuring Authentication
   - Configuring Resource Rules to Protect Resources
   - Configuring Clients to Protect Access to Resources
2. Import the certifications from your identity provider using the **keytool** program.

   See "Adding Certificates to the Application Tier Servers and Applications" in *Services Gatekeeper System Administrator's Guide* for details.
3. Set up a trusted relationship between the requesting application and your identity provider. See your identity provider documentation for instructions.
4. Configure your SAML application to add and remove excess tokens. To do so, use the **OauthSamlMbean** methods. For more information on the fields and methods of the **OauthSamlMbean** MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Protecting Resources in a Custom Communication Service

You can create a custom RESTful communication service using the Platform Development Studio (PDS) wizard and deploy the service on the Services Gatekeeper platform. This service is automatically protected by OAuth.

The service must 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 resource element. For example, use the following XML:

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

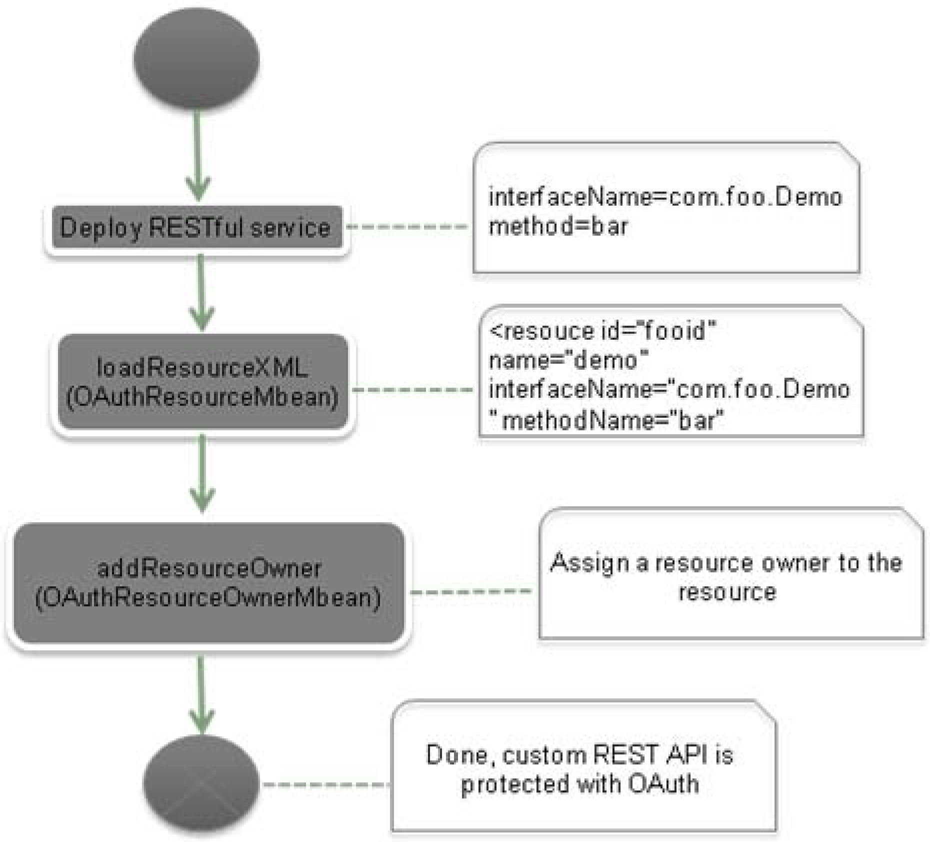
3. Execute the `addResourceOwner` operation in the **OAuthResourceOwnerMbean** with the following values (where the *resource_owner_address* is the TEL:/SIP: URI):

   ```
   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 2–1 illustrates the above steps.

Figure 2–1 Protecting a Custom Communication Service


Deploy RESTful service
interfaceName=com.foo.Demo method=bar
loadResourceXML (OAuthResourceMbean)
<resouce id="fooid" name="demo" interfaceName="com.foo.Demo "methodName="bar"
addResourceOwner (OAuthResourceOwnerMbean)
Assign a resource owner to the resource
Done, custom REST API is protected with OAuth

Example: Protecting the OneAPI Payment Service with OAuth

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

Steps to 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 WebLogic interface 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: *password*
   - 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 WebLogic interface to access the **AuthenticationURL** configuration setting in the **OAuthCommon** MBean at:

   Wlng, OauthServer, OauthCommonMBean, AuthenticationURL

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

   AuthenticationURL: `https://`*app_tier_host*:*app_tier_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 WebLogic interface 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: *password*

3. Submit the parameters to create a subscriber.

Assigning the Resource to the Subscriber to Act as Resource owner

1. Open the MBean browser in PTE or use the OAM WebLogic interface 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 subscriber.

Understanding the OAuth Resource Format

The Services Gatekeeper OAuth resource format is defined in the **oauth_resource.xsd** file located in the *middleware_home*/**ocsg_6.0/applicationsoauth2_nt.ear** file. To view the resource schema definition use an archive manager to expand **wlng_nt_oauth2.ear**.

Example 2–2 shows the contents of the **oauth_resource.xsd** file:

Example 2–2 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>
      <unique name="resourceIdUnique">
          <selector xpath="tns:resource"/>
          <field xpath="@id"/>
      </unique>
</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 2–1 lists the resource structure and attributes.

Table 2–1 oauth_resource.xml 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).
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 Representation 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>
```

The following resources are defined in this example:

- **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.

3 Monitoring OAuth Services in Service Gatekeeper

This chapter explains how to manage an Oracle Communications Services Gatekeeper implementation that uses Open Authorization Protocol v2.0 (OAuth) features.

Understanding OAuth Runtime Actions

This section describes the Services Gatekeeper runtime OAuth actions and token management, and lists the EDRs and errors that OAuth generates.

Issuing OAuth Tokens

This section explains how Services Gatekeeper manages Oauth tokens.

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 **oauth2_service.war** at the following location:

middleware_home`/ocsg_6.0/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 access a resource on behalf of any of the group members.

The group owner's URI and password authorize an application to access resources that are owned by all member in the group.

You enable or disabled this feature using the **groupUriEnabled** Mbean property in the **OauthCommonMbean**.

For more information on the fields and methods of **OauthCommonMbean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Understanding 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.

Managing Tokens

To manage OAuth tokens, use the JMX interfaces **TokenManagementMBean** which you access from an MBean browser, such as the OAM WebLogic interface or the Services Gatekeeper Platform Test Environment (PTE).

For more information on the fields and methods of **TokenManagementMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

EDRs Generated by the OAuth Service

Table 3–1 describes the EDRs that can be generated by the OAuth service.

Table 3–1 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

OAuth/Services Gatekeeper Errors and Exceptions

Table 3–2 describes the error conditions and resulting operation responses provided by the Services Gatekeeper OAuth 2.0 authorization server.

Table 3–2 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"_` `}`
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 using 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`

Table 3–2 (Cont.) Exception Scenarios

Type	Error	Response

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`

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?e`
		`rror=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?e`
		`rror=unsupported_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?e`
		`rror=unsupported_grant_type`

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

Table 3–2 (Cont.) Exception Scenarios

Type	Error	Response

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

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

4 Developing Services Gatekeeper Services Using OAuth

This chapter explains the options you have for customizing the Open Authorization Protocol v2.0 (OAuth) functionality for use with Oracle Communications Services Gatekeeper.

Understanding How to Apply SAML Tokens

Network-originated request messages that apply a security assertion markup language (SAML) token must use the structure explained in this section.

Understanding Token Request Messages

Method Type: POST

URL: **http://***Gatekeeper_IPaddress***:***Gatekeeper_port***/oauth2/saml**

Request Parameters:

- **grant_type** - Required. Must use this value: **urn:ietf:params:oauth:grant-type:saml2-bearer**
- **client_id** - Optional. The client identifier.
- **scope** - Optional. A value defined by the authorization server.
- **assertion** - Required. The assertion being used as an authorization grant. The serialization must be encoded for transport within HTTP forms. Oracle recommends that you use base64url (defined in RFC 4648) to avoid unnecessarily long strings.

Example 4–1 shows an example SAML token request message.

Example 4–1 SAML Token Request Message Example

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

client_id=s6BhdRkqt3&
grant_type=urn%3Aoasis%3Anames%sAtc%3ASAML%3A2.0%3Aassertion&
assertion=PHNhbWxwOl...[omitted for brevity]...ZT4
```

Example 4–2 shows a sample SAML assertion string before encoding.

Example 4–2 SAML Token Assertion String Example

```
<Assertion IssueInstant="2010-10-01T20:07:34.619Z"
ID="ef1xsbZxPV2oqjd7HTLRLIBlBb7"
Version="2.0"
xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
  <Issuer>https://saml-idp.example.com</Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  [...omitted for brevity...]
  </ds:Signature>
  <Subject>
    <NameID
     Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
     brian@example.com
    </NameID>
    <SubjectConfirmation
        Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
     <SubjectConfirmationData
        NotOnOrAfter="2010-10-01T20:12:34.619Z"
        Recipient="https://authz.example.net/token.oauth2"/>
     </SubjectConfirmation>
  </Subject>
  <Conditions>
    <AudienceRestriction>
     <Audience>https://saml-sp.example.net</Audience>
    </AudienceRestriction>
  </Conditions>
  <AuthnStatement AuthnInstant="2010-10-01T20:07:34.371Z">
     <AuthnContext>
       <AuthnContextClassRef>
          urn:oasis:names:tc:SAML:2.0:ac:classes:X509
     </AuthnContextClassRef>
  </AuthnContext>
 </AuthnStatement>
</Assertion>
```

Understanding Token Response Messages

Method type: HTTP/1.1 200 OK

Response Parameters:

- **access_token** - Required. The access token issued by the authorization server.
- **token_type** - Required. Normally **bearer** or **mac**. But in the SAML flow, the token type could be extended to be **bearer** or **saml-bearer**.
- **expires_in** - Required. The amount of time, in seconds, that the access token is valid.
- **scope** - Required if different from the request message **scope**. Otherwise optional.

Example 4–3 shows an example SAML token response message.

Example 4–3 SAML Token Response Message

```
  HTTP/1.1 200 OK
     Content-Type: application/json;charset=UTF-8
     Cache-Control: no-store
     Pragma: no-cache

     {
       "access_token":"2YotnFZFEjr1zCsicMWpAA",
       "token_type":"bearer",
       "expires_in":3600
     }
```

Understanding SAML Assertion Validation Messages

Table 4–1 lists the SAML assertion validation check point processing rules that your applications send.

Table 4–1 SAML Assertion Validation Check Point Processing Rules

Checkpoint	Processing Rule
Issuer	The assertion validation `<Issuer>` element must contain a unique identifier for the entity that issued the assertion. Services Gatekeeper confirms that the value for this checkpoint is included in the trusted issuers list.
Audience	The assertion must contain a `<Conditions>` element which includes an `<AudienceRestriction>` element, which in turn, contains an `<Audience>` element with a URI reference. The token endpoint URL of the authorization server may be used as an acceptable value for an `<Audience>` element. The authorization server must verify that it is an intended audience for the assertion.
Subject	The assertion must contain a `<Subject>` element. The `<Subject>` element may identify the resource owner for whom the access token is being requested.
NotOnORAfter	The assertion must have an expiration time that limits the time window during which it can be used. The time limit can be expressed either as the **NotOnOrAfter** attribute of the `<Conditions>` element or as the **NotOnOrAfter** attribute of a suitable `<SubjectConfirmationData>` element.
Method	The `<Subject>` element must contain at least one `<SubjectConfirmation>` element that allows the authorization server to confirm it as a **bearer** assertion. The `<SubjectConfirmation>` element must have a **Method** attribute with a value of **urn:oasis:names:tc:SAML:2.0:cm:bearer**.
Signature	The assertion must be digitally signed by the issuer and the authorization server MUST verify the signature. Services Gatekeeper validates the signature and confirm that the trusted certificate is identical to the attached certificate in the assertion.

Example 4–4 shows an example SAML assertion validation message.

Example 4–4 Example SAML Assertion Validation Message

```
<Assertion IssueInstant="2010-10-01T20:07:34.619Z"
ID="ef1xsbZxPV2oqjd7HTLRLIBlBb7"
Version="2.0"
xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
  <Issuer>https://saml-idp.example.com</Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  [...omitted for brevity...]
  </ds:Signature>
  <Subject>
    <NameID
     Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
     brian@example.com
    </NameID>
    <SubjectConfirmation
        Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
     <SubjectConfirmationData
        NotOnOrAfter="2010-10-01T20:12:34.619Z"
        Recipient="https://authz.example.net/token.oauth2"/>
     </SubjectConfirmation>
  </Subject>
  <Conditions>
    <AudienceRestriction>
     <Audience>https://saml-sp.example.net</Audience>
    </AudienceRestriction>
  </Conditions>
  <AuthnStatement AuthnInstant="2010-10-01T20:07:34.371Z">
     <AuthnContext>
       <AuthnContextClassRef>
          urn:oasis:names:tc:SAML:2.0:ac:classes:X509
     </AuthnContextClassRef>
  </AuthnContext>
 </AuthnStatement>
</Assertion>
```

Understanding OAuth Customization

This section contains information on customizing the Service Gatekeeper OAuth functionality. It starts by explaining the customization options:

- Implementing a Third-Party Authentication Service
- Creating an OAuth Interceptor
- Integrating a Third-Party Subscriber Repository
- Creating an OAuth Extension Handler for New Credentials
- Customizing OAuth Resource Grant Tests

This chapter also explains how to develop applications using OAuth:

- OAuth Application Developer Guide
  - Interacting with the Services Gatekeeper OAuth Service
  - OAuth Access Flow In Services Gatekeeper

Implementing a Third-Party Authentication Service

You can delegate authentication to a third-party authentication service provider instead of with the default Subscriber Management Service. The 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 "Understanding the OAuth Endpoints" for more information.

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 OAuth Authorization request to Services Gatekeeper in a format that looks like this:

   ```
   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 add more detailed information to it using the **client_id** and **scope** request parameters. This additional information is appended to the location header in the 302 redirect response directed to the configured authentication endpoint.

   The location header contains these elements:

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

   This is an example 302 response:

   ```
   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&grant_url=grant&client_
   info=%7B%22clientId%22%3A%22client123%22%2C%22clientName
   %22%3A%22client123%22%2C%22clientDescription
   %22%3A%22client123+desc%22%7D&scopes_info=%5B%7B
   %22scopeId%22%3A%22POST-%2Fpayment
   %2Facr%3AAuthorization%2Ftransactions
   %2Famount%22%2C%22scopeDescription
   %22%3A%22Charge+or+refund%22%2C%22parameters
   %22%3A%5B%7B%22code%22%3A%22billable+item+id%22%7D%5D%7D%5D
   ```

   In addition to the original OAuth authorization request parameters, the detailed format specifications of all additional parameters are defined in Table 4–2:

Table 4–2 OAuth Authorization Request Parameters

Parameter	Description
grant_url	The URL can be submitted later according resource owner's approval. See "Understanding the OAuth Endpoints" for more information.

Table 4–2 (Cont.) OAuth Authorization Request Parameters

Parameter	Description

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 the parameters listed in Table 4–3 through an HTTP POST operation. The OAuth flow continues normally after redirecting the resource owner toward the Grant endpoint. The client then receives the authorization code at the Redirect endpoint.

   Table 4–3 lists the OAuth grant endpoint POST parameters.

Table 4–3 OAuth Grant Endpoint POST Parameters

Parameter	Description
user_address	Address of resource owner

Table 4–3 (Cont.) OAuth 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 rejects 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

Creating an OAuth Interceptor

This section describes the basic principles for creating a custom OAuth interceptor.

As described in "Implementing a Third-Party Authentication Service", 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 4–4 lists the OAuth parameters available in the `RequestContext` object for an OAuth 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 “Creating and Using Custom Interceptors” in*Services Gatekeeper Extension Developer's Guide*.

Table 4–4 OAuth RequestContext 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: Using a Custom OAuth Interceptor to Retrieve OAuth Information

This example shows how to retrieve and use 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, use an additional scope parameter called **maxAmount**:

```
/**
* The following example shows a way to control the maximum charged value using
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);
}
```

Integrating a Third-Party Subscriber Repository

Service Gatekeeper offers the flexibility to integrate with custom subscriber repositories for user authentication. You can develop a customized Subscriber Manager to authenticate users against external subscriber repositories 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 with this method.

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

   For additional information on customizing the default **SubscriberManager**, including method details, see **SubscriberManager** in the "All Classes" section of the *Services Gatekeeper Java API Reference*.

Creating an OAuth Extension Handler for New Credentials

Services Gatekeeper supports a rich set of credential types that OAuth uses for security. However, if your implementation uses new or evolving credential standards that Services Gatekeeper does not support, you have the option to create a customized extension handler to use them. The Platform Development Studio enables you to customize endpoint parameters, grant types, response types, or errors using the **OAuth2 Extension Handlers** wizard.

For details see "Generating an OAuth 2.0 Extension Handler" in *Services Gatekeeper Extension Developer's Guide.*

Customizing OAuth Resource Grant Tests

Services Gatekeeper enables you to add your own customized tests to OAuth resource requests. For example, you may want to restrict access to a resource to just subscribers from a specific block of email addresses. There are two ways to add the customized tests:

- Using an **auth.jsp** file which you then reference using a **REGEX_MATCH** parameter in the resource XML file. Services Gatekeeper then perform any tests (regular expressions) that you have added to the **jsp** file. Each subscriber must pass the tests in this file to get access to the resource. For details, see *Oracle Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server* at:

  https://docs.oracle.com/cd/E24329_01/web.1211/e21049/toc.htm
- Creating a custom services interceptor. For details see "Using Service Interceptors to Manipulate Requests" in *Services Gatekeeper Extension Developer's Guide*

OAuth Application Developer Guide

This section contains information useful for application developers using OAuth with Services Gatekeeper.

Interacting with the Services Gatekeeper OAuth Service

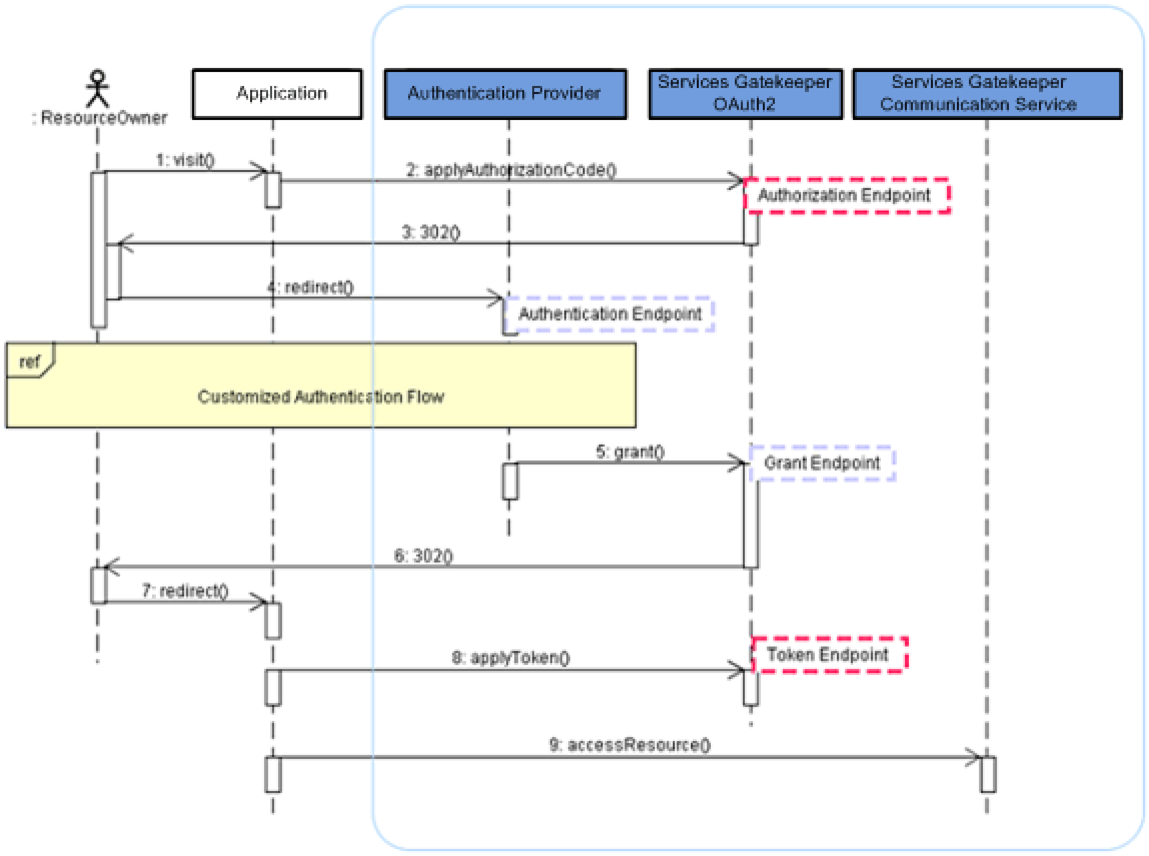
This section describes the available OAuth endpoints, the steps involved in obtaining an access token, and how applications use access tokens to access a REST resource in Services Gatekeeper.

The following endpoints are available in the OAuth Service:

- Authorization endpoint
- Token endpoint
- Authentication endpoint
- Grant endpoint

Figure 4–1 demonstrates the end to end flow to obtain an access token and use the access token to access the resource.

Figure 4–1 OAuth Endpoints and Functional Responsibility



OAuth Access Flow In Services Gatekeeper

This procedure describes the OAuth access flow:

1. A resource owner visits an application website 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 using 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 these 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 you use 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 verifies 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 is 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",
"anonymousid":"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 can 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 using an HTTP authorization header in the request.

   For a MAC token, the application constructs the HTTP authorization header using a MAC key 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.'"';
```

   This example request contains 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":""}
}
```

The following example request contains 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-"
}
```