Oracle® Communications Services Gatekeeper

API Management Guide

Release 7.0

E81152-01

July 2018

ORACLE®

Oracle Communications Services Gatekeeper API Management Guide, Release 7.0

E81152-01

Contents

3 Managing Network Service Interfaces

4 Managing APIs for Partner Applications

5 Using Actions to Manage and Manipulate API Traffic

6 Creating Custom HTTP Processors

Preface

This document describes how to use the API management platform and the partner relationship management portals offered by Oracle Communications Services Gatekeeper (Services Gatekeeper) to develop applications for use by application developers. It includes a high-level overview of the application development process, including the login and security procedures, and a description of the interfaces and operations.

Audience

This book is intended for software developers who will integrate functionality provided by telecom networks into their applications.

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

The following documents provide information related to creating applications that interact with Services Gatekeeper:

- *Oracle Communications Services Gatekeeper Concepts*
- *Oracle Communications Services Gatekeeper Partner Relationship Manager Developer's Guide*
- *Oracle Communications Services Gatekeeper Communication Service Guide*

1

Understanding API Management with the PRM Portals

This chapter provides an overview of Oracle Communications Services Gatekeeper application programming interface (API) management platform and its partner relationship management (PRM) portal applications.

About the Services Gatekeeper API Management Platform

You use the Services Gatekeeper API Management platform to create applications that subscribe to APIs for the services you expose. Through these applications, you can provide network quality of service (QoS) control, messaging, call control, big data analytics to internal developers, partners, and third-party developers.

The Services Gatekeeper API management platform processes all requests for the APIs associated with the services it supports. This processing included default actions you can take on messages, and options to create your own processing. For example, you can:

- Normalize all incoming requests to a unified format for processing the requests.
- Customize the message process flow as necessary.
- Regulate the use of your network resources and communication web services.
- Provide an API proxy for the services to expose by specifying the network address for the service.
- Provide a documentation URL describing the service, for use by internal or third-party developers.

Services Gatekeeper supports the API management platform in both single-tier and multi-tier environments. By default, the API Management platform is deployed as a single layer, with the possibility to cluster nodes together. It can also be deployed in application-tier or service-tier clusters. See *Services Gatekeeper Concepts* for more information.

About the PRM Portals and Users

The Services Gatekeeper API platform supports the following web-based PRM portals that offer three different roles for managing APIs:

- Partner and API Management Portal

  You use the Partner and API Management Portal to:

  – Create and manage APIs for use in applications.

The APIs are configured from network service interfaces (created in Network Service Supplier Portal), communication service APIs, and Web service APIs provided by Service Gatekeeper.

- Review and approve applications that use the exposed APIs. These applications are created in Partner Portal.
- Manage partner groups and service level agreements.
- Configure rules as a chain or chains of actions and locate the actions in the application-initiated or service-initiated flow of the request, as appropriate.

Your network operators and enterprise customers work with Partner and API Management Portal. They create and manage APIs, approve partner applications, manage partner groups, and also manage partner and network service supplier accounts.

This document and the Online Help documentation refer to users of Partner and API Management Portal as *partner managers*.

- Network Service Supplier Portal

  Network service suppliers (NSSs) use Network Service Supplier Portal to provision network resources as network service interfaces. You then use Partner and API Management Portal to manage their network service interfaces, and use them to create APIs for partner applications.

  NSSs can be in your group, in another group in your company, or from a separate entity (company) entirely. They use Network Service Supplier Portal for this, and gain access by completing an online registration request. The NSSs receive an email notification from the partner manager, approving or deleting the authorization request.

  This document and the Online Help documentation refer to users of Network Service Supplier Portal as *network service suppliers* or NSSs.

- Partner Portal

  You use the Partner Portal to create applications. These applications represent services that you provide to your customers. You configure them from the network resources and communication web services running on the Services Gatekeeper.

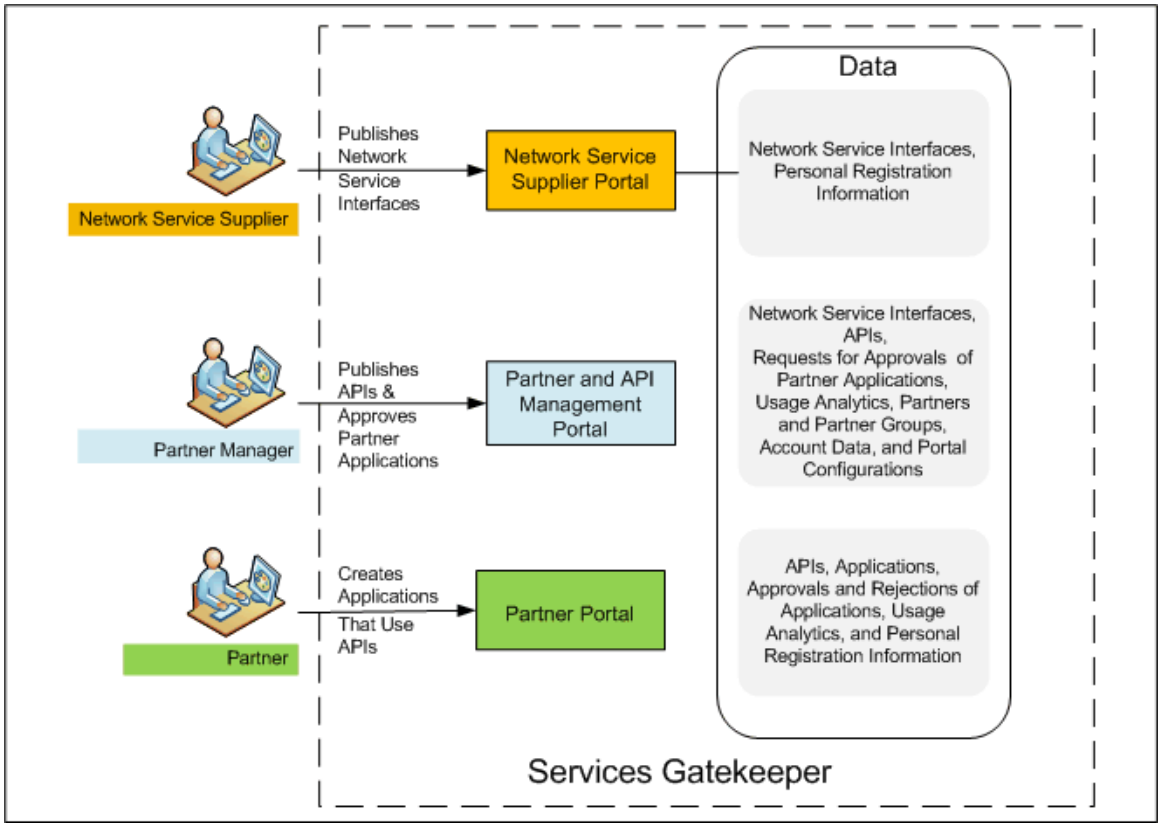
  Each partner application subscribes to one or more APIs exposed by Partner and API Management Portal. When active, a partner application can successfully handle associated HTTP requests and responses to maintain quality of service. Each contains logic targeted to improve customer satisfaction, such as setting the permissions for a request to exceed the quota limit.

  Application developers require authorization to use Partner Portal. Each application developer completes an online registration request displayed by Partner Portal. The application developer receives an email notification from the partner manager announcing that the request was approved or denied.

  This document and the Online Help documentation refer to users of Partner Portal as *partners*.

Figure 1–1 shows the users of the three portals and the data they create, access, and use in the Services Gatekeeper platform.

Figure 1–1 Services Gatekeeper PRM Portal Users and Data



How the API Management Platform Works

Services Gatekeeper uses the API Management platform to intercept and process the requests and responses in real-time, using *actions* that you specify. You configure these actions or chains of actions in the Partner and API Management Portal. You order the actions as necessary in the application-initiated or service-initiated flow of the traffic. When Services Gatekeeper receives a request message, it processes the incoming request and performs the actions that you have specified on it. For example, you can use actions to:

- Maintain security (such as verifying the service level agreement).
- Transform the message format as necessary (such as from JSON to XML format).
- Any other custom tasks you configure for the message flow.

You can manage endpoint routing by customizing actions, such as Groovy injection methods, or by using Java-based service provider interface. These actions can provide specific logic for interacting with third-party APIs, filtering or modifying field values, and so on. See "Configuring Actions Chains to Manage API Traffic" for more information.

About the Elements that Control the Quality of Service

The quality of service a Partner Portal application provides to the end user depends on:

- Application specifics in Partner Portal.
- Network service specific in the Network Service Supplier Portal.

- The API specifics in the Partner and API management Portals.

These factors determine quality of service:

- Service interfaces exposed by the network
- Maximum usage and throughput for the service exposed
- The API methods subscribed to in an application
- Service level agreements in effect for the API methods selected in an application
- Request limits and quotas for the partner group (to which an application belongs)
- Interceptors and action elements that act upon the request or response in real time

When an application developed using the PRM portals is in an active state, the API management platform receives the associated HTTP requests and proxies each request based on predefined rules set up in the portals.

About Developing APIs

The process required to provide your network services as APIs to be called in real time consists of the following tasks:

- Configuring Network Service Interfaces to Expose Your Services
- Configuring APIs to Expose Your Services For Use by Partner Applications
- Subscribing to APIs to Enhance Partner Applications

Configuring Network Service Interfaces to Expose Your Services

Network service suppliers create network service interfaces from the network resources that they want to expose. As a network service supplier, you control how partner managers (and therefore, partners) use your network services by specifying the throughput capacity for the network resource in each network service interface you create. This helps safeguard the associated networks from external attacks, and the resources from being overloaded.

Network service suppliers create these interfaces in Network Service Supplier Portal and Services Gatekeeper make these interfaces available in Partner and API Management Portal. Partner managers work offline with you to ensure that the network services interfaces are optimally configured for use in the network.

For example, your network group wants to market a Web service that permits applications or games to store and retrieve high scores for their games. Your network service supplier creates an interface for such a service in Network Service Supplier Portal under the name of High Score Game RESTful web service and makes it available to the network operator (partner manager). For each interface, the network service supplier provides the access URL for the interface and also information about the accessible methods of the interface.

Configuring APIs to Expose Your Services For Use by Partner Applications

As a partner manager, you use Partner and API Management Portal to create and expose APIs using the available network service interfaces and Services Gatekeeper communication services. In addition, you manage the different versions of the APIs and the life cycles of your client applications.

You exercise full control over the resource throttling and security processes by configuring elements (such as maximum usage, throughput) in the APIs you expose.

You can configure Services Gatekeeper to perform actions on both the API request and response traffic using the request and response actions chains.

Continuing with our example, you (as a partner manager) use the High Score RESTful Game web service network service interface to create and publish an API called High Score Game Notification API. You specify:

- The maximum usage and throughput for the API service exposed.
- Interceptors and action elements to act upon the request or response.
- Information about the accessible methods.

Subscribing to APIs to Enhance Partner Applications

Partners (or application developers) use Partner Portal to create applications that subscribe to one or more APIs. Before registering the application, partners collect all the information necessary, including:

- Name and description of the application.
- The time period when the application is active.
- The service to provide.
- The rate at which the application provides the service.

As a partner, you register an application by entering the appropriate information about the application. You can select the APIs that provide the services your application when you create the application, or later. Your partner manager publishes the list of APIs that are available to the applications.

For each API, you specify a desired number of requests that the application sends to the network and the minimum number of requests it receives from the network within an allotted time. By doing so for each API you include in the application, you can tailor the quality of services you provide to your customers.

When you have configured an application, you submit the application registration request to your partner manager for approval. When your partner manager approves the application, Partner Portal displays the application registration approval notification. Then, you access the application in Partner Portal and set a traffic user password. The application is then available for you to make API changes, and is ready for use.

In our example, an online gaming application company owns a game called Textrocks. In order to enhance the user experience, the online gaming application company wants to upgrade that game with the ability to query for high scores. Your partner is associated with that online gaming application company. Your partner sees the High Score Game Notification API displayed in Partner Portal. The partner clicks the API, opens the API description document, and upgrades the Textrocks software by using the required methods of the High Score Game Notification API. After the partner manger approves the application, the partner sets up the traffic password and the API is then ready for use.

How Services are Deployed Using PRM Portal Applications

Figure 1–2 Steps in the PRM API Development Process

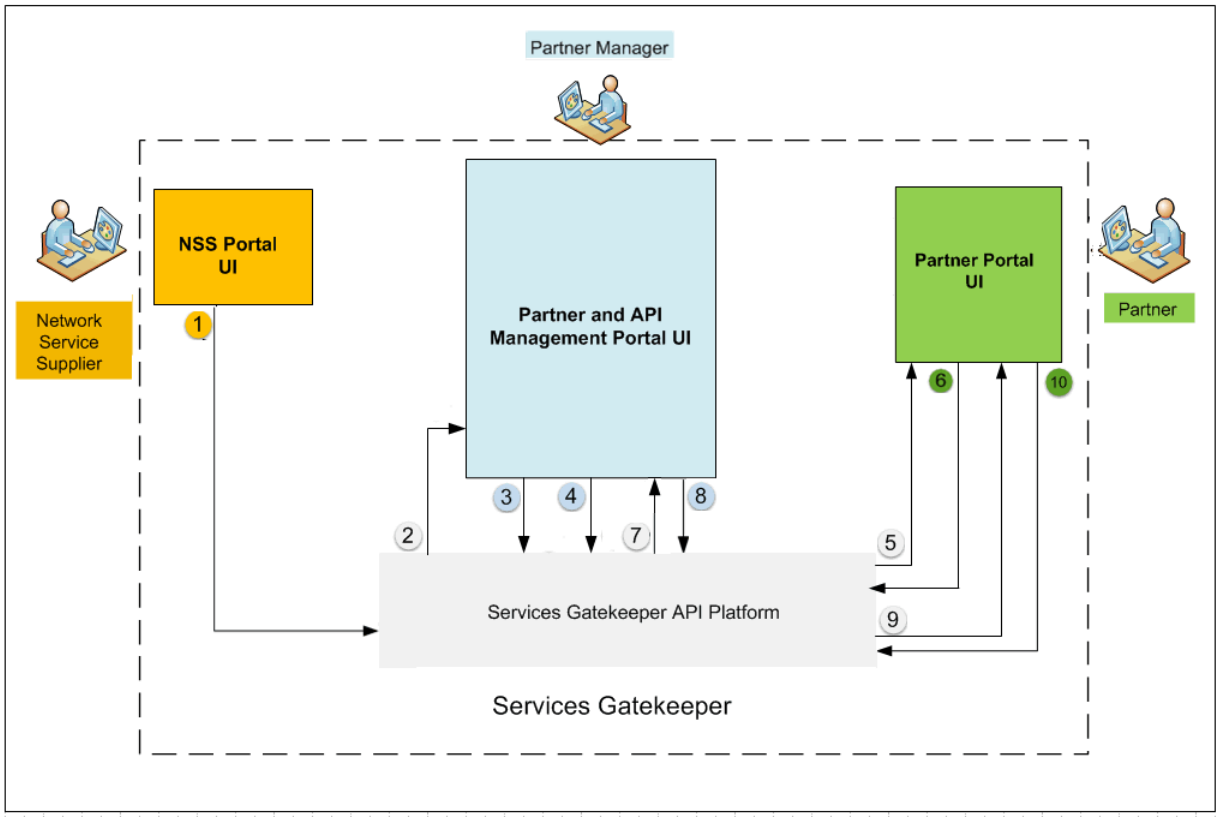


Figure 1–2 shows how the three PRM API portals deploy services in your network:

1. The network service supplier uses Network Service Supplier Portal to publish a network service interface.
2. Services Gatekeeper displays the network service interface in Partner and API Management Portal.
3. The partner manager uses the interface to create an API in Partner and API Management Portal.
4. The partner manager changes the status of the API to be published in Partner and API Management Portal.
5. Services Gatekeeper displays the API in Partner Portal.
6. The partner views the API in Partner Portal. The partner creates an application that subscribes to this API and specifies the desired request limit and quota. The partner submits the application to be registered for use.

   The partner can also create an application without an API and subscribe to them later.
7. Services Gatekeeper displays the application registration request in Partner and API Management Portal.
8. The partner manager reviews the application registration request and approves it.

The partner manager may also deny a request based on service level agreements and resource-related factors, such as the resource requests and quotas in effect.

9. Services Gatekeeper displays the approval (or denial) of the application registration request in Partner Portal.
10. If the application registration request is approved, the partner sets the traffic user password in the application. This password enables tracking traffic usage in the network.

    If the application registration request is rejected, the partner can change it and resubmit the application.

Understanding API Management Security

You need to consider these aspects of Services Gatekeeper API Management Security:

- Securing the Services Gatekeeper administration, portal, and managed servers. For information, see the *Services Gatekeeper Security Guide* in general and the "Deploying Services Gatekeeper in a Demilitarized Zone" chapter in particular.
- Securing the traffic between web-based users and applications and Services Gatekeeper. For details, see "Securing Network Traffic for APIs" in Services Gatekeeper Security Guide.
- Securing Services Gatekeeper APIs from unauthorized access. See "Securing the Services Gatekeeper API Methods" for details.
- Securing your API services (backend services) from unauthorized access. See "Securing the API Service" for details.
- Securing web traffic by creating white list of IP addresses allowed to communicate with your REST-based APIs. See "Protecting REST APIs with a White List of IP Addresses" in *Services Gatekeeper Security Guide* for details.

Using Cross-Origin Security with APIs

You can take advantage of browser validation features by using Cross-Origin Resource Security (CORS) to validate cross-origin resource requests. You do this per-API by configuring and adding the Services Gatekeeper CORS action in the API request action chain. You generally configure the allowed CORS origins, headers, and methods, and the action chain fails processing if the request message does not match the configuration. However, you also have to option to pass non-conforming request headers, which can then be configured to fail at the browser level.

See "Understanding the Default Actions" for a description of the options available. See *Partner and API Management Portal Online Help* for details about how to configure the CORS action.

About PRM Portal Service Level Agreements

Partner managers create partner groups and assign all partner accounts to a partner group.

When a partner manager creates a partner group, for example a partner group called *Platinum*, the partner manager sets up a service level agreement (SLA) for that partner group. A partner group SLA defines:

- A partner group's request limit for a service. That is, the number of requests per second allowed.

- The quota limit. That is, the number of requests the partner group can process.
- The number of days for processing the requests allowed for that group.

The quota limit is an integer with a maximum value of 2147483648 requests.

When creating an API, the partner manager can restrict the availability of that service to specific partner groups, or expose the API to all partner groups. Private APIs are only visible and available to the members of the groups you specify.

If the partner manager makes an API public, the API is visible and available in Partner Portal for all partner accounts.

Services Gatekeeper provides a default partner group called **sysdefault_sp_group**. When a partner manager creates or approves a partner account, Services Gatekeeper assigns that partner account to **sysdefault_sp_group**. This default service provider group has a blank SLA and therefore no request limits or quota allotments. Until a newly created partner account is assigned to a different partner group, the partner who owns that account has no APIs available and cannot successfully register an application.

Partner managers can create any number of uniquely named partner groups and change the group assignment for a partner account. At any given time, a partner account is assigned to one partner group and the partner is notified whenever there is a change to the group assignment. Partner managers also manage the partner accounts and partner groups they create and, when necessary, delete them.

If a partner manager assigns a partner account to a different partner group, the partner manager must reconcile any discrepancies between the allowances stipulated by the SLA of the new partner group and the usage requirements of the applications associated with the partner account. See "Group Assignments for Partners and SLAs".

About Extending the PRM API Portals

Partner managers can extend and customize portals by adding new pages to the portals, and creating a new navigation entry points to enter these new pages. For more information, see "About Customizing PRM Portals".

2

Developing Applications with the PRM Portals

This chapter provides an introduction to the setup of Oracle Communications Services Gatekeeper partner relationship management (PRM) portal applications and the application development process.

Required Software

Services Gatekeeper supports Network Service Supplier Portal, Partner and API Management Portal, and Partner Portal on Chrome 38.0.2125.111 m, Mozilla Firefox 24.8.1, and Internet Explorers 11 browsers. The portals included when you install Services Gatekeeper. See *Services Gatekeeper Getting Started Guide* or *Services Gatekeeper Multi-tier Installation Guide* for information about installing the portal applications.

Accessing the PRM Portals

As partners, network resource suppliers, and partner managers, you access the portals and do not interact directly with Services Gatekeeper.

You need a valid account and password to use each portal.

See "Administering the PRM Portals" for information about administering the portals.

Understanding the Partner Portal User

A WebLogic administrative user is required to start and use the Partner and API Management portal. You create this user partner manager user account when you install a default (single-tier) Services Gatekeeper implementation or multi-tier implementations using the Services Gatekeeper Administration Console. See *Services Gatekeeper System Administrator's Guide* for details on managing users. The new user must have a userlevel of **1000-Admin user**, and a type of **1-PRM OP user**.

Understanding the API Management Proxy Settings

These options determine the proxy settings that your APIs use. They are enforced in this order (the last one wins):

1. The Java operating system proxy settings (**http.proxyHost**, **http.proxyPort**, **https.proxyHost**, and **https.proxyPort**).

   See the Java documentation for instructions on how to configure these settings.

2. The proxy server that you enter in the **Network Proxy** field in each API.

   See *Services Gatekeeper Partner and API Management Portal Online Help* for information on using this field.

3. Creating a proxy object in a **Groovy** action in each API.

   See "Using Actions to Manage and Manipulate API Traffic" for details on the actions you can create for each API.

Starting the Partner and API Management Portal

Your Services Gatekeeper administrator provides you with the URL of the location where Partner and API Management Portal is installed in your environment. In addition, the administrator provides you with a valid account name and password that gives you access to Partner and API Management Portal.

The default URL to Partner and API Management Portal is:

`http://IP_address:port/portal/partner-manager/index/login.html`

Where *IP_address* is the IP address of the host system running Services Gatekeeper, and *port* is a port number to use. The default port is 8001.

> **Tip:** To request a user name and password for a partner manager account, to reset the password for your account or to reinstate your account, contact your Services Gatekeeper Administrator.

To access the Partner and API Management Portal:

1. Open a supported web browser and go to the URL provided to you for Partner and API Management Portal.
2. In the **User Name** field, enter the user name for your partner manager account.
3. In the **Password** field, enter the password for your partner manager account.
4. Click **Sign In**.

Starting the Network Service Supplier Portal

Your Services Gatekeeper administrator provides you with the URL of the location where Network Service Supplier Portal is installed in your environment.

The default URL to Network Service Supplier Portal is:

`http://IP_address:port/portal/service-supplier/index/ssLogin.html`

Where *IP_address* is the host system running Services Gatekeeper and *port* is a communication port to use. The default port is 8001.

When you obtain this URL, you can go to the website and do one of the following:

- If you received an email notification with a valid user name and password for Network Service Supplier Portal, the account was created for you. Sign in to the portal by entering the user name and password.
- If you do not have a valid user name and password for Network Service Supplier Portal, go to the URL and set up a network service supplier account. See "Setting Up a Network Service Supplier Account".

Setting Up a Network Service Supplier Account

To set up a network service supplier account:

1. Open a supported web browser and go to the URL provided to you for Network Service Supplier Portal.

2. Click **Create New Account**.
3. Enter the required information for all fields in the registration form that display asterisks next to them.

   The remaining information can be provided later.
4. Read and accept the terms and conditions.
5. Click **Register** to submit the registration request.

   Network Service Supplier Portal displays a message stating that the request is now pending approval and that an email notification is sent to the email address you entered in the registration form.
6. Click **OK**.

> **Note:** Your account name and password become valid only after your partner manager approves your registration request.

When you receive the email notification that your registration request has been approved, you can go to the same URL, enter the user name and password and sign in to use the Network Service Supplier Portal console.

If your registration request has been denied, consult with your partner manager and submit a new registration request to obtain a valid network service supplier account.

Starting the Partner Portal

Your Services Gatekeeper administrator provides you with the URL of the location where Partner Portal is installed in your environment.

The default URL to Partner Portal is:

`http://IP_address:port/portal/partner/index/partnerLogin.html`

Where *IP_address* is the host system running Services Gatekeeper, and *port* is a communication port to use. The default port is 8001.

When you obtain this URL, you can go to the website and do one of the following:

- If you received an email notification with a valid user name and password for Partner Portal, the account was created for you. Sign in to the portal by entering the user name and password.
- If you do not have a valid user name and password for Partner Portal, go to the URL and set up a partner account. See "Setting Up a New Partner Account".

Setting Up a New Partner Account

To submit a registration request:

1. Open a supported web browser and go to the URL provided to you for Partner Portal.
2. Click **Create New Account**.
3. Enter the required information for all fields in the registration form that display asterisks next to them.

   The remaining information can be provided later.
4. Read and accept the terms and conditions.

5. Click **Register** to submit the registration request.

   Partner Portal displays a message stating that the request is now pending approval and that an email notification is sent to the email address you entered in the registration form.

6. Click **OK**.

> **Note:** Your account name and password become valid only after your partner manager approves your registration request

When you receive the email notification that your registration request has been approved, you can go to the same URL, enter the user name and password and sign in to use the Partner Portal console.

If your registration request has been denied, consult with your partner manager and submit a new registration request to obtain a valid partner account.

About Developing Applications

The application development process consists of selecting APIs where the configuration for each API is based on a specific type of an interface and specifying how they are used in the application.

About the Types of Interfaces Used in an API

You can create APIs based on the following interface types:

- Existing URL
- Existing WADL/WSDL file
- Registered Network Service
- Existing Communication Service

About Registered Network Services

When you create an API based on a registered network service, that service interface is created in Network Service Supplier Portal.

A network service provider and the associated network operator, service provider, product manager, back-office personnel, or customer sales representative decide on the parameters for the network resources that are going to be provided for use as network service interfaces.

The network service supplier creates network service interfaces and saves them.

These network service interfaces are then displayed in Partner and API Management Portal for use by partner managers. See "Managing Network Service Interfaces" for more information.

About Developing Applications

The process of developing an application consists of the following steps.

1. A partner manager creates APIs in Partner and API Management Portal using the network service interfaces provided by the network service suppliers and communication services, exposed by Services Gatekeeper and configuring them as

required. The partner manager publishes the APIs and they are then displayed in Partner Portal.

See "Managing APIs for Partner Applications" for more information.

2. A partner associated with the partner manager creates an application in Partner Portal. You can subscribe to APIs to the application when you create it, or later. This application can issue HTTP requests to Services Gatekeeper. The partner selects the APIs to use and requests a desired number of requests the application sends to the network and the minimum number of requests it receives from the network within an allotted time.

   See "Managing Partner Applications" for more information.

3. The partner submits the application for approval. The status of the application is set to **pending**.

4. The partner manager reviews the application and, if it meets the requirements, accepts it. When the partner manager approves the application, its status is set to **active**.

   At this point the partner manager can change the desired number of requests the application sends to the network and the minimum number of requests it receives from the network within an allotted time.

5. The partner receives a notification that the application is approved and the application icon displays the state as **ACTIVE**.

   The partner can access the application to change the traffic user password and update the access token. The service level agreement (SLA) for the application is the SLA associated with the partner group to which the partner manager assigns the partner.

When an application is in the **ACTIVE** state, it can be marketed to customers.

About the Services Gatekeeper Communication Services

For multi-tier installations, Services Gatekeeper provides access to all of its communication services. When all the communications services are installed, the Partner Manager and Partner Portals display the available plugin instances. You can also install and use any of the communication services for single-tier installations. You must specify them in a custom install.

By default, when you install Services Gatekeeper, the following plugin instances are available:

- Plugin_px21_third_party_call_sip#wlng_nt_third_party_call_px21#6.0.0.0
- Plugin_px21_third_party_call_inap#wlng_nt_third_party_call_px21#6.0.0.0
- Plugin_px21_call_notification_sip#wlng_nt_call_notification_px21#6.0.0.0
- Plugin_px21_presence_sip#wlng_nt_presence_px21#6.0.0.0

The portals display the following types of services and interfaces as selections:

- CallNotification: For call notification, you can select from the following interfaces:
  - Interface: com.bea.wlcp.wlng.px21.plugin.CallDirectionManagerPlugin
  - Interface: com.bea.wlcp.wlng.px21.plugin.CallNotificationManagerPlugin
  - Interface: com.bea.wlcp.wlng.px21.callback.CallDirectionCallback
  - Interface: com.bea.wlcp.wlng.px21.callback.CallNotificationCallback

- Presence: For presence notification, you can select from the following interfaces:
  - Interface: com.bea.wlcp.wlng.px21.plugin.PresenceConsumerPlugin
  - Interface: com.bea.wlcp.wlng.px21.plugin.PresenceSupplierPlugin
  - Interface: com.bea.wlcp.wlng.px21.callback.PresenceNotificationCallback
- ThirdPartyCall: For third-party calls, you use
  - Interface: com.bea.wlcp.wlng.px21.plugin.ThirdPartyCallPlugin

If an operator creates another plugin instance in Services Gatekeeper, it is added to the existing set. The portals display the service type for the new plugin and interfaces.

Updating an Active Application

An active application is updated when the partner manager makes changes to the application or approves changes made by the partner who created and/or manages the application.

About Data Integrity During Updates to an Active Application

Services Gatekeeper maintains the integrity of application data in the following way:

1. A partner updates to an application through the **Applications** page in Partner Portal. Services Gatekeeper receives this update request.
2. Services Gatekeeper begins a WebLogic transaction to update the application. It locks all data associated with the application, such as its SLA, and short codes.
3. Until Services Gatekeeper ends the WebLogic transaction and releases the lock on the application data, no other user can update the application data.

Ways in Which an Active Application is Updated

An active application is updated in the following ways:

- A partner adds an API to, or deletes an API from an application. Partner Portal sends a notification to Partner and API Management Portal.

  The partner manager reviews the updated application. Before approving the application update, the partner manager may alter the number of requests the application sends to the network and the minimum number of requests it receives from the network within an allotted time.
- A partner is moved to a different partner group. The active applications come under the SLA associated with the destination partner group.

3

Managing Network Service Interfaces

This chapter describes how you can configure and manage network resources as network service interfaces. You do this by using the Oracle Communications Services Gatekeeper Network Service Supplier Portal, one of the partner relationship management (PRM) portal applications.

About Network Resources and Service Interfaces

In Network Service Supplier Portal, you configure network service interfaces using the network resources you want to provide to network operators. You expose these interfaces for the use of partner managers in Partner and API Management Portal.

Partner managers use these interfaces when they create APIs in Partner and API Management Portal. Partners use the APIs to create applications in Partner Portal. When the APIs are active and in use in partner applications, the interfaces provide the services of the network resources exposed by the network operator.

About the Network Service Interface Data

Before you begin configuring a network service interface in Network Service Supplier Portal, collect the following data about the network resource:

- Basic information, consisting of:
  - Name: a name to identify the service interface
  - Version: a version number to identify the service interface

    If you are updating an existing network service interface, specify the newer version number.
  - Description: a description to identify the service interface
- If you are updating an interface, provide the date and time when the older version should be deprecated.
- Access information, consisting of URLs for the following:
  - The network service interface
  - The WADL/WSDL file associated with this interface
  - Documentation for this interface
- Throughput capacity provided by this interface, in terms of maximum transactions per second (TPS)
- Security information to access the network interfaces, consisting of:

- Your choice of authentication and authorization, if any.

  Services Gatekeeper enables you to set up the network service interface with no security, text-based security, and OAuth.

Save an offline copy of this information for the interface.

About Interface Statuses

A network service interface can have on of the following statuses:

- ACTIVE

  When you create and save a network service interface in Network Service Supplier Portal, Services Gatekeeper sets the state of the interface to ACTIVE.

  Partner managers can subscribe to network service interfaces that are in an ACTIVE state.

- DEPRECATED

  A deprecated network service interface represents an older version of a current interface. When you update an existing interface, the updated version becomes the active version and the previous version becomes deprecated. You can also deprecate an existing interface in Network Service Supplier Portal, by selecting the icon adjoining the trash can icon within the interface icon. Services Gatekeeper asks you to specify the date and time when the interface should be deprecated.

  Tip: Maintain backward compatibility when you update an active interface.

  The data for a deprecated interface cannot be modified and deprecated interfaces are not available to new APIs.

  The service associated with a deprecated interface is available to APIs that subscribed to the interface and only for a designated period. After that period, the network service supplier can remove the deprecated interface.

Life Cycle Stages of a Network Service Interface

Each network service interface goes through the following stages:

1. As a network service supplier, you create an interface and submit it in Network Service Supplier Portal.
2. Services Gatekeeper displays a notification on the **MESSAGES** page of the associated Partner and API Management Portal.

   Partner managers with access to the Partner and API Management Portal can use this interface to create an API.
3. When the interface is active, you can do one of the following:
   - Update the interface, thereby deprecating the earlier version and creating an active interface version to be used in APIs created from this point onward.
   - Remove the interface.

     To remove an interface from active use, you delete the interface in Network Service Supplier Portal. If:

     - Any API is using the interface currently, Services Gatekeeper does not permit the removal of the interface.

A warning is seen in Network Service Supplier Portal.

– No API is using the interface currently, Services Gatekeeper removes the interface from the associated Partner and API Management Portal.

  A notification is seen in Partner and API Management Portal.

4

Managing APIs for Partner Applications

This chapter explains how to create, configure, and manage application programming interfaces (APIs) by using Oracle Communications Services Gatekeeper partner relationship management (PRM) portal applications.

About APIs for Partner Applications

An API for a partner application contains all the information required to use the interface. Service providers or partner managers create and manage APIs in Partner and API Management Portal, and application developers or partners use them in Partner Portal.

Services Gatekeeper enables you to publish an API based on any HTTP URI. To create an API, you select a network interface from the types of network interfaces or communication services the network supports, and configure the API. You can expose the API publicly to all partner groups or restrict it to selected partner groups. To assist partners in subscribing to the API, you also provide the URL to the location hosting the necessary documentation on the API.

In Partner Portal, partners subscribe to these APIs in the applications they create. When partner applications are being used, the APIs associated with the applications support traditional communication services, and for internet- or enterprise-based APIs from back-end third-party services.

About the API Data

Before you begin configuring an API in Partner and API Management Portal, collect the following data about the API:

- Basic information to identify the API, consisting of a unique name, and version number, and a short description. You can also provide a different name that is used in the access URI that your customers use to access the API (a context root).
- Type of interface, such as a URL for the existing Web service, an existing WADL/WSDL file, a registered network service, or an existing communication service. For the selected interface, the details about the interface, and the network security provided for it.
- URL to the documentation on this API.
- Type of exposure for the API, specifying whether the API interface uses the SOAP or REST protocol, its URL, encryption requirement, the exposed service resources and methods.

Configuring an API as public makes it accessible to all partners. Alternately, you can configure the API to be private and designate partner groups that are permitted to use the API.

- Type of security you want to use to protect access to Services Gatekeeper API methods, and type of security you want to use to protect the API service.
- Action chains to process incoming and outgoing traffic from and to the API.

For details about the API data to collect, see *Services Gatekeeper Partner and API Management Portal Online Help*.

About Naming Your APIs

The APIs you create can have two identifiers: a *display name* in the API **Name** field, and a different *context root* that identifies the API in its Access URL. The two identifiers allow you to create an API name to display in the PRM Portals that is different from the name seen by your customers. The identifiers are identical by default; if you fill in just one of these fields, that value is used for both identifiers. Once the API is created however, you cannot change these identifiers.

For example, assume that you want to name an WeatherAPI API within the PRM Portals, but want to present **WeatherSouthWestAPI** as the name in the access URI that users see. When creating the API in the Partner and API Management Portal, you could specify WeatherAPI in the **Name** field, and **WeatherSouthWestAPI** in the **Context Root** field. Within the PRM Portals, the API is named **WeatherAPI**, but users of the API access it using this URI:

```
http://IP_Address:port/WeatherSouthWestAPI
```

The object defining the API stores the name in the **apiName** field and the context root in the **apiDisplayName** field.

Note: An **apiName** can contain spaces.

About Presenting API Resources to Your Customers

The URI that you present to your customers (partners or their customers) has two components, a *context root* and a *path*. See "About Naming Your APIs" for details on the context root. This section deals with the second part of the service identifier, the *path*, which is often mapped to a different *service path*.

This resource mapping mechanism is designed to provide you with the flexibility to use a different internal URL structure than the one you present to customers. You can also potentially change the internal URL structure without your customers having to alter their request messages. This is particularly useful if your Services Gatekeeper implementation uses REST to communicate.

You present services or resources to customers by defining each of them in the API **Resource** table. If the URLs that users send to an API are different from the URLs of a specific resource or service, you can use the **Resources** table to map one to the other. Figure 4–1 shows the **Resource** table that each API uses with a sample resource in it.

Note: If you create an API from an existing WSDL or WADL file, the API **Resource Table** is automatically populated with the resources from the file.

Figure 4–1 The Services Gatekeeper API Resource Table

You fill in these fields for each resource:

- **Name** - an informal name to identify the resource.
- **Path** - The path given on the incoming request message (excluding the context root). This field takes variables; see "Using Variables and Wildcard Characters in the Request Path and Service Path" for details.
- **Method** (or **Verb**) - The request action. GET, POST, PUT, DELETE for REST request messages, or a SOAP Method to use.
- **Service Path** - The service path (resource) that combination of **Path** and **Method**/**Verb** maps to.
- **Service Verb** - The action to take on the resource. GET, POST, PUT, DELETE for REST request messages, or a SOAP Method to use.
- **Expose** - Allows you specify whether to expose or hide a resource from partners.

Using Variables and Wildcard Characters in the Request Path and Service Path

You have the option to use variables or wildcards in the resource path to map different URLs sent in with a request message to a different service path for a resource. At its simplest, you can map a request path to a different service path. Table 4–1 shows a simple mapping.

Table 4–1 A Simple Path Mapping Example

Path Value	Service Path Value	Result
/value1	/value2	/value1 is mapped to /value2

Note that if the request URI is not an exact match, it is not mapped. For example if the path value in Table 4–1 was **/value1/extra**, it would not be mapped to **/value2** because it does not exactly match **/value1**.

Table 4–2 shows more examples of mapping request message paths to service paths.

Table 4–2 Example Path Value Mapping With Wildcards and Variables

Path Value	Service Path Value	Result
/value1*	/value2	/value1 is mapped to /value2 /value1/value3 is mapped to /value2
/value1*	/value2*	/value1 is mapped to /value2 /value1/value3 is mapped to /value2/value3
/value1/{var1}/value3	/value2/{var1}/value4	/value1/123/value3 is mapped to /value2/123/value4
/value1/{var1}/{var2}	/value2/{var2}/{var1}	/value1/abc/123 is mapped to /value2/abc/123
/value1/{var1}/{var2}	/value2	/value1/abc/123 is mapped to /value2

Resource path wildcard and variable rules:

- The "*" wildcard character can only be used at the end of a path. That is, you cannot use this syntax: /*/value1
- You cannot change the HTTP method for the matched IP addresses with variables or wildcard characters.
- The **var***n* string is only an example; you can replace it with any string you want. Just remember to put variable strings inside curly brackets "{}", ensure that the strings are identical, and ensure that the values they match are identical.
- The variables can include more than one path layer separated by slash "/" characters. For example **{var1}** could map to **/value12/abc/123**.

About the Status of an API

Each of the APIs in the portals has an assigned status. The current state of an API indicates whether the API is available for use, is modifiable, or no longer in service.

A partner manager manages the status of an API from the time that the API is created to the time when the partner manager removes it from the portals. As the partner manager, you update the status of the API in the **Life Cycle** tab of the API page in Partner and API Management Portal.

- CREATED

  When a partner manager creates an API in Partner and API Management Portal, Services Gatekeeper stores the data on the API and assigns the status of the API as CREATED.

  APIs with CREATED status are in an unpublished state and are not visible in Partner Portal. They can be viewed in Partner and API Management Portal only and modified in that application.

  You can change the state of an API from CREATED to PUBLISHED. If a partner manager decides to discard a created API instead of publishing it, the API is removed.

- PUBLISHED

  A partner manager changes the status of a newly created API to PUBLISHED in Partner and API Management Portal. Then, Services Gatekeeper makes the API available to all partner groups or to designated partner groups, based on the API configuration.

  Partners can subscribe to the APIs when they create their applications.

  All modifications to a published API are performed in Partner and API Management Portal only. You can change the state of an API from PUBLISHED to:

  – DEPRECATED, when a newer version of the API is published

  – SUSPENDED, if necessary

Important: If you deprecate an API for which you are not providing a newer version, applications that currently use the API are affected. Check the **Applications** tab for the API to verify that the tab does not list any application.

If the **Applications** tab lists one or more applications, then, do the following before you change the status the API.

For each application:

1. Contact the partner who owns the application offline.
2. Ensure that your partner takes the required actions to safeguard the applications.

- DEPRECATED

  A deprecated API represents an older version of an API.

  Deprecated APIs are not available to new applications. A deprecated API is available to applications that subscribed to it until the end of the effective period set for the API. After that, applications can not access the API. It is the partner's responsibility to access all current applications that used the prior API and modify them so that they call the updated API.

  All calls to the prior version of the API are supported until the date when the API is suspended or removed from portal views. From then on, all calls to the prior API version fail and the request receive the 404 error response.

  You can change the state of an API from DEPRECATED to one of the following in Partner and API Management Portal:

  – SUSPENDED

  – PUBLISHED, when the API is required by partners and there is no other API with the same name in the system.

- SUSPENDED

  When a deprecated API reaches the final date set by the partner manager, Services Gatekeeper notifies all partners. Also, an API can be temporarily withdrawn from circulation by a partner manager in Partner and API Management Portal.

  Note: When a partner manager suspends a deprecated API that is still in use by applications, Services Gatekeeper displays a warning in Partner Manager. If the partner manager continues with the suspension of the API, the associated applications may be affected.

  In either scenario, the API is considered to be in a suspended state and the URL for the API is no longer valid. Calls made to a suspended API return a 404 error response.

About Temporarily Suspending APIs

At times, you may want to temporarily block the use of an API that you published and made available one, some, or all partner groups. This scenario occurs if there is an issue with an API and the resolution process for that issue requires you to disable the API temporarily. In such a situation, you can suspend the API temporarily and notify the partner groups whose applications are affected by this suspension.

About API Versioning

You have the option to create an alphanumeric string to use as a versioning number for the APIs that you create. Services Gatekeeper does not do anything with the string, but it can be useful for you to keep track of your APIs. You add the string to the **Version** field of the **Create API** screen as you create the API. Or leave this string blank to omit a version number. The string uses any of the [0-9a-zA-Z\\.-_].* characters.

The syntax for the API access URL:

IP_address:*port*/*context_root*(/*version_number*)

Where:

- *IP_address* **:***port* - The IP address and port number to use to connect to the system hosting Services Gatekeeper.
- *context_root* - the context root of the API. By default, this value is the name for the API that you entered in the **Name** field, but you can change it as needed.
- *version_number* - the optional version string that you entered for the API.

This example show an access URL with a context root of **apis/myapi** and no version number:

```
http://203.0.113.18:8001/apis/myapi
```

You have these options for specifying a particular API/version when calling an API:

- If there is no version, just send the access URL for the API. For example:

  ```
  GET apis/myapi HTTP/1.1
  Host: 203.0.113.18:5879
  http://203.0.113.18:8001/apis/myapi
  ```

  or with the final "/":

  ```
  GET apis/myapi HTTP/1.1
  Host: 203.0.113.18:5879
  http://203.0.113.18:8001/apis/myapi/
  ```

- Specify the version number after the context root. This example access URL matches **myapi** version 2:

  ```
  GET apis/myapi HTTP/1.1
  Host: 203.0.113.18:5879
  http://203.0.113.18/apis/myapi/2
  ```

- Specify the version number in the HTTP request **Accept** header. For example:

  ```
  GET apis/myapi HTTP/1.1
  Host: 203.0.113.18:5879
  Accept: 203.0.113.18/apis/myapi+json;version=2
  ```

Providing API Credentials to Partners

After a partner manager approves an API application registration, the Partner and API Management portal returns an application instance ID and authentication credentials to the requesting partner. The partner then uses the instance ID and credentials to sent traffic through Services Gatekeeper to the application. The credentials include both a **Traffic User Password** for basic authentication, and an **Access Token** for OAuth authentication.

The MBean attribute **DafExpireTime** has been added to the **OAuthCommonMBean** to control how long the **Access Token** is valid. The default value is 3600 seconds.

Creating APIs for Use in Partner Applications

To create an API, enter the details for the API on Partner and API Management Portal. The supported types of interfaces are:

- A WADL or WSDL file for the API containing some methods or resources defined for the API.
- The connection to a network interface that does not have a WADL or WSDL file.
- The connection to a network service from a set of network services maintained by Network Service Supplier Portal.
- The connection to an existing Services Gatekeeper communication service.

Use the **Create API** page of Partner and API Management Portal to enter the details about the API interface.

Manipulating HTTP Query Parameters in API Messages

You can change HTTP query parameters using these API components in this order:

- A request message
- The message request URL
- An API Resource Service Path
- Using a Custom Action
  - Using the **withoutQueryParameter** method to remove the query string
  - Using **withQueryString** to change the query string or **withQueryParameter** method to overwrite the query parameter value. These are methods to the **CalloutBuilder** class in the Actions Java API. See the "All Classes" section of the *Java API Actions Reference* documentation.

In other words, a query string in a request message can be changed by any of these components, but it is the last component to change it that sets the final string value.

See "Using Actions to Manipulate HTTP Query Parameters" for information on using actions to change your query parameters.

Securing Services Gatekeeper Methods and API Services

This section explains the types of security you need to consider when creating an API:

Securing the Services Gatekeeper API Methods

You use the **Exposed API Security** settings when creating an API to select a method for authenticating applications attempting to use methods in the Services Gatekeeper API. Used with all types of APIs. You have these options:

- TEXT - Requires the application to provide a user name and password. Can be used with OAuth. Requires that you set the **EnableSouthCookie** performance setting to **true**. See "Setting Actions System Performance Settings" for instructions.
- OAuth - Allows the application to access third-party resources. Can be used with TEXT-based security.
- AppKey - Requires the application to authenticate itself using an application key. Especially useful for applications that cannot otherwise identify themselves. By default Services Gatekeeper searches the **x-api-key** query parameter for the key first, then the header. You can change this behavior using the **AppKeyValidation** action in an action chain. See "Understanding Front and Middle Actions" for details on the **AppKeyValidation** action.

Securing the API Service

If you create an API using the **Existing URL** or **Existing WADL/WSDL File** selections you use one of the **Service Security** option to securing the API service (sometimes called a *backend service*). You have these **Service Security** options:

- None - Allows unauthorized access to the API service.
- Text - Text-based security relies on a username and password to protect the API service. You are asked to provide the user name and password for the account that monitors and manages the traffic. Can be used with OAuth security.
- OAuth - OAuth-based security allows you to provide access to third-party resources requiesYou are asked to provide:
  - Authorization URI, the URI to which the user will be sent for authentication and authorization.
  - Token URI, the URI to which the user will be sent to obtain a request token. This request token acts as a temporary token and authorizes the user to use the interface.
  - Client Redirect URI, the URI to which the user will be sent after a successful authentication.

If you are creating an API using the **Registered Network Service** selection, Services Gatekeeper takes this security setting and parameters from the interface you created using the Network Service Supplier Portal. If you are creating an API using the **Existing Communication Service** selection, Services Gatekeeper takes this setting from the underlying communication service plugin.

Configuring Actions Chains to Manage API Traffic

When end users use your partner applications, the applications generate requests and responses that call upon one or more of the subscribed APIs created in Partner and API Management Portal and supported by Services Gatekeeper.

You can set up actions to manage and manipulate the information in the requests and responses. See "Using Actions to Manage and Manipulate API Traffic" for information.

Understanding the API Back-end Server Configuration

You specify an access URL for each API that you manage using Partner and API Management portal. For multi-tier Services Gatekeeper implementations, the Partner and API Management portal provides back-end server configuration settings that you use to provide alternatives to the access URL in cases where:

- The traffic is mobile-originated (network to application). In this case the traffic must be directed to the network tier server.
- The back-end cluster has a public URL that is preferable to the access URL.
- The back-end server only supports SSL communication. You can provide an alternative that does not require it.

You specify Partner and API Management portal back-end server configuration settings by selecting the **Settings** in the Header bar, then Selecting **Configuration**, and then filling out the **back-end Server** section of **System Configuration** section.

If your configuration is not one above, the system fills the back-end server with the following default data. For:

- Multi-Tier installations: Access tier address and port
- Single-Tier installations: Node address and port

Updating APIs

At times, when an API has been in use, service providers and/or application developers may change some configured settings for the API. You can update an API by adding more resources to it or publish a newer version of the API. To modify an API, you select the API from the list of APIs in Partner and API Management Portal and make the necessary changes.

For details on updating an API in Partner and API Management Portal, see *Services Gatekeeper Partner and API Management Portal Online Help*.

About an API Status and Modifications to its Data

Partner managers can modify the configuration of an API in Partner and API Management Portal. However, the following restrictions apply and are based on the current status of the API:

- CREATED: Partner managers create APIs. They can modify all the fields in an API when it is in the created, unpublished state.

  Partners do not have access to APIs that are set to CREATED.

- PUBLISHED or DEPRECATED: Partner managers have access to published and deprecated APIs. Partners have access to APIs that are set to PUBLISHED. When an API is deprecated, some applications may be supported by a deprecated API, for a defined period, but the partner does not have access to the API.

  If the API is in a PUBLISHED or DEPRECATED state, partner managers can do the following:

  – Modify its description.

  – Modify the action chain, including add or remove actions, or change an action (for example, change the service level agreement to raise or lower a rate), or other action details.

  – Update the documentation link for the API.

  – Add more resources and expose more methods.

  – Change the encryption level by adding the HTTP or HTTPS setting.

- Modify a private API to make it public and available to all groups. For a private API, add more partner groups to increase its availability to intended customers.
- Edit the API action chain. For example, a partner manager can set the service level agreement to a very low rate.

Important: To enable you to maintain backward compatibility of active APIs, Services Gatekeeper permits the addition of resources, access settings, partner groups, and accessibility.

It does not permit any reduction to these elements in active APIs.

- SUSPENDED: Suspended APIs are not modifiable, by default.

  However, if there is some technical or business-related issue, the partner manager may temporarily block the API. See "About Temporarily Suspending APIs". Any change to a suspended API is made to resolve an issue. All changes fall within the above restrictions.

Suspending Applications from Using an API

You can suspend an application from using an API by using the Suspend button on the API **Applications** tab in the Partner and API Management Portal. This button toggles between **Suspend** and **Un-Suspend** states for the application. In the suspended state the application can not access the API. See the *Services Gatekeeper Partner and API Management Portal Online Help* for details.

Removing APIs

APIs that are in a CREATED state and not associated with any application can be removed from the portals. APIs that are in a SUSPENDED state and not in dispute can also be removed from the portals.

As a partner manager you access the **API List** page in Partner and API Management Portal, select the specific API icon, and click its trash icon (displayed within the API icon) to remove an API.

5

Using Actions to Manage and Manipulate API Traffic

This chapter explains how you use actions and action chains to manage Oracle Communications Services Gatekeeper API requests and responses, and change the information in those requests.

Configuring Actions Chains to Manage API Traffic

When your customers use your partner applications, the applications generate requests and responses that call upon one or more of the APIs created in Partner and API Management Portal.

You select and configure actions to filter and act on the request and response messages that contain calls to the APIs. You combine actions into request and response *action chains* (combinations of actions) that are processed in the order you arrange them. The sections in this chapter:

- Introduces the actions and explains how to configure them.
- Explains what the individual default actions do.
- Provides information for some common action processing tasks.
- Provides some troubleshooting information for action chains.

For information about creating you own custom actions, see “Creating Custom Actions for Your APIs” in *Services Gatekeeper Portal Developer’s Guide*.

About Action Chains

You implement actions on API request or response traffic using the **Actions** tab for each API. You drag and drop individual actions into the **Request** or **Response** action chain to make them take effect. When you drop an action, a pane appears with the action parameters for you to configure. Every action has at least one optional **Instance ID** parameter that you can use to create an informal alphanumeric version number to identify the action. Some of these parameters are optional and some required. If you get a fail message when you try to save your changes, the problem can be a missing parameter. See "Understanding the Default Actions" for details about the parameters for individual actions.

Note: Actions only affect traffic for APIs that you create using the Partner and API Management Portal. Use interceptors to affect request and response messages for Services Gatekeeper communication services.

You can use the actions provided by Services Gatekeeper, or create your own actions to act on the request (or response) traffic for an API. You can set up actions chains to take a wide range of real-time effect on the request or response messages, such as identity management, mapping to support data formats and protocol changes, authorization, logging, monitoring, and statistics. Some action parameters are optional and others required. If you get a fail message when you try to save your changes, the problem can be a missing parameter.

The sequence of actions in the action chain depends on the direction of the message flow. It is either application-initiated and traveling to the network, or server-initiated and traveling to the application. Some actions (such as identity management) are valid only in the request flow and some only in the response flow. Other actions (such as supporting protocol changes, validations) are common in that they are applicable in either direction. Services Gatekeeper does not allow you add an invalid action to an action chain.

When you position an action incorrectly in a chain, Partner and API Management Portal prompts you to ensure that the sequence of actions is valid for that direction.

Note: Actions only affect traffic for APIs that you create using the Partner and API Management Portal. Use interceptors to affect request and response messages for Services Gatekeeper communication services.

Actions in Application-Initiated Flows

When an application sends a request that contains a call to an API subscribed to by the application, Services Gatekeeper processes it using all actions configured for it in order. Services Gatekeeper can receive and handle incoming HTTP requests such as SOAP, REST, or XML-RPC. Services Gatekeeper checks the incoming request and performs preconfigured tasks such as enforcing policy (associated with the service level agreement), translates the message as necessary (such as from JSON to XML format), and so on. In addition, it processes the action with any custom tasks you added to suit your requirements. Finally, Services Gatekeeper forwards the outbound request message to the server.

Actions in Server-Initiated Flows

Server-initiated flows occur when, for example, an application sets up a notification for an event. The server listens for the event and sends a notification to the application. The notification comes in to Services Gatekeeper as a request from the server and contains a call to an API subscribed to by the application. Services Gatekeeper receives the request from the server and processes it according to the response action chain. The final step in the action chain would be to forward the outbound request message to the application in the format required by that application.

When the application responds to this notification, the API proxy processes that response according to the tasks preconfigured for that sequence of the action chain. Finally, the response for the server-initiated request is sent back to the server.

Understanding Front and Middle Actions

Actions for both chains are divided into *front actions* and *middle actions*. Use front actions as major filters on the incoming requests. For example, you can use **Throttling** to regulate the usage of the API and regulate traffic based on specific partner groups. (API management already provides throttling based on the application and based on the network service.)

Use middle actions to act on the content of the request or response in real time. For example, use **Callout** to perform an HTTP GET operation against the Request URL and store the response as required.

About Action Statuses

You only change the action statuses if you create your own actions using the instructions in "Creating Custom Actions for Your APIs" in *Services Gatekeeper Portal Developer's Guide*.

An action can have one these statuses:

- ACTIVE

  The Oracle-supplied actions, and any new actions you create in Partner and API Management Portal have a state of the ACTIVE.

  Any API can use an action the ACTIVE state.

- DEPRECATED

  A deprecated action represents an older version of a current interface. When you update an existing action, the updated version becomes the active version and the previous version becomes deprecated. You can also deprecate an existing action in Partner and API Management Portal, by selecting the icon adjoining the trash can icon within the interface icon. Services Gatekeeper asks you to specify the date and time when the interface should be deprecated.

  Tip: Maintain backward compatibility when you update an active action.

  The data for a deprecated action cannot be modified, and deprecated actions are not available to new APIs.

  The service associated with a deprecated action is available to APIs that subscribed to the interface and only for a designated period. After that period, the network service supplier can remove the deprecated interface.

Setting Actions System Administration Settings

This section explains the administration settings specific to Actions. See *Services Gatekeeper System Administrator's Guide* for information about general Services Gatekeeper administration settings.

Setting Actions System Performance Settings

You use the Actions system performance settings to ensure that your actions do not inadvertently include code that adversely affects Services Gatekeeper performance. The Actions system performance settings are listed here with the default value in parenthesis:

- **KeepNorthSession** Boolean (false) - If true, this setting keeps the session open after the request is complete
- **EnableSouthCookie** Boolean (false) - If true, allows applications to communicate with backend servers using cookies. You must set this to true if your Actions use basic (text-based) authentication.
- **UseSession** Boolean (false) - If true, allows an application to use a session when communicating with backend servers.
- **MaxTotalConnections** Integer (4000) - Sets the maximum total allowed network connections. You can also set this option from the system WebLogic server start up script; see "Using the WebLogic Startup Script to Set Action System Performance Settings" for details.
- **SocketTimeoutMs** Integer (30000) - The time, in milliseconds, that socket waits for activity before closing down. You can also set this option from the system WebLogic server start up script; see "Using the WebLogic Startup Script to Set Action System Performance Settings" for details.
- **ConnectTimeoutMs** Integer (30000) - The time, in milliseconds, that a connection waits for activity before closing down. You can also set this option from the system WebLogic server start up script; see "Using the WebLogic Startup Script to Set Action System Performance Settings" for details.
- **ReuseAddress** Boolean (true) - If true, directs Services Gatekeeper to ignore the TCP **time_wait** state. This improves performance by allowing Services Gatekeeper to close the socket immediately after the connection is closed, instead of continuing to wait for late packets.

You have these options to change these settings:

- Edit the **DafConfigurationsMBean** directly. For details on **DafConfigurationsMBean** see the "All Classes" section of the Actions Java API Reference documentation.
- Use the Services Gatekeeper Administration Console. See "Using the Administration Console to Set Action System Performance Settings" for details on using the Administration Console.
- See "Using the WebLogic Startup Script to Set Action System Performance Settings" for details on setting these Actions system performance parameters as startup script options.

Using the Administration Console to Set Action System Performance Settings

To configure the Actions performance settings using the Administration Console:

1. Start the Administration Console.

   See "Starting and Using the Administration Console" in *Services Gatekeeper System Administrator's Guide* for details.

2. Click **Lock & Edit**.
3. Navigate to **OCSG**, then *admin_server_name* (**Server1** by default), then **Container Services**, then **DafGeneralInformation**.

   The **Configuration and Provisioning on** *server_name* page appears with the performance settings.

4. Change the settings as necessary.
5. Click **Release Configuration**.

6. Restart the administration sever to make your changes take effect.

Using the WebLogic Startup Script to Set Action System Performance Settings

You can change these Actions system performance settings as command line options:

- oracle.sdp.daf.max_total_connections
- oracle.sdp.daf.socket_timeout_ms
- oracle.sdp.daf.connect_timeout_ms

This example sets the maximum total number of connections to 3000:

```
% startWebLogic.sh -Doracle.sdp.daf.max_total_connections=3000
```

Setting Actions White and Black Lists

You use these "white" and "black" lists to control the individual methods and packages that software developers are allowed to use in actions. These are the default lists:

- Black list of methods: **exit** and **setProperty**.
- Black list of packages:
  - **java.lang.Thread;java.lang.Runnable**
  - **java.lang.ClassLoader**
  - **java.lang.Class**
  - **java.io**
  - **java.net**
  - **groovy.lang.GroovyShell**
  - **groovy.util.Eva**
  - **groovy.io**
  - **groovy.net**
- White list of methods: **java.net.InetAddress.getLocalHost()**
- White list of packages: **null**

To view or change the white and black lists using the Administration Console:

1. Start the Administration Console.

   See "Starting and Using the Administration Console" in *Services Gatekeeper System Administrator's Guide* for details.

2. Click **Lock & Edit**.

3. Navigate to **OCSG**, then *admin_server_name* (**Server1** by default), then **Container Services**, then **DafGeneralInformation**, then **Operations**.

   The **Configuration and Provisioning on** *server_name* page appears with the performance settings.

4. Select an operation from the **Select An Operation** menu. The choices are:
   - **loadBlackListMethod** - Add Java/Groovy methods to the black list.
   - **loadBlackListPackage** - Add Java/Groovy packages to add to the black list.
   - **loadWhiteListMethod** - Add Java/Groovy methods to the white list.

- **loadWhiteListPackage** - Add Java/Groovy packages to the white list.
- **retrieveBlackListMethod** - Show the list of the Java/Groovy methods on the black list.
- **retrieveBlackListPackage** - Show the list of Java/Groovy packages on the black list.
- **retrieveListAll** - Show the contents of all white and black lists.
- **retrievePerformanceSets** - Show the Actions performance settings (set using **Attributes**).
- **retrieveWhiteListMethod** - Show the list of all items on the white lists.
- **retrieveWhiteListPackage** - Show the list of all packages on the white lists.

Understanding the Default Actions

This section lists the default actions provided by Services Gatekeeper. You configure these actions in the **Actions** tab for each API in the Partner and API Management Portal, or by using MBeans. See the "All Classes" of the Actions Java API Reference for the appropriate MBeans.

All actions have at least one parameter, **Instance Id**, which you can use to create an identifier, or version number for the action. The instance id is captured in EDR data for each action. This is useful, for example, if you create multiple custom actions and need to identify which EDR fields came from a specific action. Instance id values **0-7** are reserved for Oracle internal use. See "Understanding EDR Fields for API Management" in *Services Gatekeeper Administrator's Guide* for details on the actions EDR fields.

Note: Do not confuse the actions instance id with the **instanceid** field that Services Gatekeeper maps to clients to use SLA enforcement with OAuth.

You invoke these actions on requests passing between the application and the network service (backend service):

- appKeyValidation
- Callout
- CORS
- Groovy
- Json2Xml
- RateLimit
- SchemaValidation
- Throttling
- Xml2Json
- XSLT

appKeyValidation

Authenticate an application and give it access to an API. Typically used when the application cannot identify itself by other means. Confirms that an application has permission to access a specific API using an application key. You identify the application key, and you can also test whether that key exists in a header, or in a query parameter inside the header.

This action probes the x-app-key parameter for the API key in the request/response (unless you specify a different parameter). Configure **appKeyValidation** with these parameters:

- **Headerkey** - (String) Specifies the header to check for the application key.
- **QueryParameter** - (String) Specifies the application key to check for.
- **UseHeader** - (Boolean) **True** directs the action to confirm that the application key exists on the header. **False** does not use this test. Can be used with **UseQueryParameter**.
- **UseQueryParameter** - (Boolean) **True** directs the action to confirm that the application key exists in the query parameter. **False** does not use this test. Can be used with **UseHeader**.

Callout

A REST Call-out. Performs an HTTP GET operation against the **RequestUrl** and puts the response into the value of the **storeResponse** field for use by another action. You can use Callout to change attributes of the incoming request. This feature is also available to use in a Groovy script or custom action.

Set up a proxy or business service callout by:

1. For the **Request URL** parameter, enter the remote URL for the callout in the **Value** column.
2. For the **Store Response** parameter, enter the name of variable on the context in which the response is stored in the **Value** column.

The content of the message is constructed using the values of variables in the message context. The message content for outbound messages is handled differently depending upon the type of the target service.

This example uses the **createCallout** method to obtain a value for **myattribute** at the **myurl** URI:

```
context.createCallout().withRequestUrl("http://myurl.com")
.withRequestMethod("GET").build().send("myattribute");
```

You can then use the value for **myattribute** in a consecutive action:

```
HttpResponse response = context.getAttribute("myattribute");
and response has getStatus(), getHeaders and getBodyAsType(...)
```

CORS

Web pages are free to imbed some resources, such as images, from outside their own domain. Exceptions to this are fonts and AJAX (XMLHttpRequest) requests, which are usually restricted to the same domain that the parent web page itself is in. The "cross-domain" AJAX requests in particular are forbidden because they represent glaring security risks.

Note: The CORS action allows you to configure cross-origin resource sharing (CORS) to safely access third-party resources for your APIs. The default settings are empty, so CORS messages are not processed.

At a minimum, you need allow **Support Preflight Requests** or **Allow Simple Requests** and the appropriate origin, header, and method parameters so that this action passes CORS traffic.

This is a security-based action, so put it as early in your middle action chain as you can. After you add this action to an action chain, configure it to process the CORS messages that your implementation uses.

Also, having this early in the action chain allows you to take advantage of the **Allow Non Verified Requests** option which cancels action chain processing if the action fails. This can prevent unnecessary processing if this action fails.

See the CORS specification at the W2C Cross-Origin Resource Sharing website:

https://www.w3.org/TR/2014/REC-cors-20140116/

The configuration parameters give you the options to:

- Allow preflight requests, simple requests, or both.
- Decide whether to require credentials when calling the API.
- Allow or disallow all origin URIs, resource headers, or methods to call the API.
- Create "white lists" of origins URIs, resource headers, or methods that are allowed to call the API.
- Decide whether to support credentials in the request.
- Decide whether to continue action chain processing if the CORS action does not pass validation.
- The maximum time a user agent is allowed to cache results.

Table 5–1 lists the CORS configuration options.

Table 5–1 CORS Configuration Options

Option	XML Representation	Data Type	Description
Support Preflight Passthrough	<supportPreflightPassthrough>	Boolean	Allows (true) or disallows (false) the API to send reflight requests to the backend service.
Support Preflight Requests	<supportPreflightRequests>	Boolean	Allows (true) or disallows (false) preflight requests for resources. No default setting.
Support Simple Requests	<supportSimpleRequests>	Boolean	Allows (true) or disallows (false) the API to process CORS simple requests.
Supports Credentials	<supportsCredentials>	Boolean	Allows (true) or disallows (false) the use of credentials when calling this API.
Allowed Origins	<allowedOrigins>	List of Strings	A "white list" of URIs that can act as origins for CORS requests for this API. Can be used with **Return Wild Card Allowed Origins**.
Allow Any Origins	<allowAnyOrigins>	Boolean	Allows (true) or disallows (false) any URI that can act as origins for requests to this API

Table 5–1 (Cont.) CORS Configuration Options

Option	XML Representation	Data Type	Description
Return Wildcard Allowed Origins	<returnWildCardAllowedOrigins>	Boolean	Allows (true) or disallows (false) the ability to use the "*" (star) wildcard character in a lists of origins, allowing requests from ANY origin. Use Carefully. This setting becomes invalid if **Supported Credentials** is true. See **Allowed Origins**. This example allows all domains that end in us.mydomain.com to submit requests: `*.us.mydomain.com`
Allowed Methods	<allowedMethods>	List of Strings	The list of methods allowed to call this API.
Allow Any Method	<allowAnyMethod>	Boolean	Allows (true) or disallows (false) any method to call this API.
Exposed Headers	<exposedHeaders>	List of Strings	The list of headers (other than simple response headers) that can be exposed to a resource.
Allowed Header	<allowedHeaders>	List of Strings	The list of headers allowed in a preflight request
Allow Any Header	<allowAnyHeader>	Boolean	Allows (true) or disallows (false) any header to be allowed in a preflight request.
Maximum Age	<maxAge>	Long	The time limit, in seconds, that a user agent is allowed to cache the result of the request.
Allow Non-Verified Requests	<allowNonVerifiedRequests>	Boolean	Allows (true) or disallows (false) action chain processing to continue if the request does not pass validation for the header, origin, or method validation.

This example CORS configuration allows all CORS features and responds to the origin with any requests it receives:

```
<corsActionConfig>
  <allowAnyHeader>true</allowAnyHeader>
  <allowAnyMethod>true</allowAnyMethod>
  <allowAnyOrigin>true</allowAnyOrigin>
  <supportPreflightRequests>true</supportPreflightRequests>
  <supportSimpleRequests>true</supportSimpleRequests>
</corsActionConfig>
```

This example allows any header or method, supports both simple and preflight requests, but limits CORS requests to only the **yourdomain.com** and **mydomain.com** domains:

```
<corsActionConfig>
  <allowAnyHeader>true</allowAnyHeader>
  <allowAnyMethod>true</allowAnyMethod>
  <allowedOrigins>yourdomain.com</allowedOrigins>
  <allowedOrigins>mydomain.com</allowedOrigins>
  <supportPreflightRequests>true</supportPreflightRequests>
  <supportSimpleRequests>true</supportSimpleRequests>
</corsActionConfig>
```

This example allows any header or method, but only from a derivative of **mydomain*.com**, and that supports only simple requests.

```
<corsActionConfig>
  <allowAnyHeader>true</allowAnyHeader>
  <allowAnyMethod>true</allowAnyMethod>
  <returnWildCardAllowedOrigins>
  <allowedOrigins>mydomain*.com</allowedOrigins>
  <supportPreflightRequests>false</supportPreflightRequests>
  <supportSimpleRequests>true</supportSimpleRequests>
</corsActionConfig>
```

Groovy

Use a Groovy language script to change any aspect of a request or response message. You can add Groovy code to change the message content, destination, status, and so on. The script can be as simple or complex as your implementation requires. This option requires knowledge of the Groovy programming language.

You use the Actions Java API to obtain content from the API request and response traffic to act on in the Groovy action. Start by looking through the **HttpContext** class for information about extracting elements from messages.That class uses the **HttpMessage**, **HttpRequest**, and **HttpResponse** classes and some of its own methods to retrieve elements from messages that you can then manipulate. These classes are available from the "All Classes" section of the *Actions Java API Reference*.

See "Common Actions Programming Tasks" and "Creating Custom Actions for Your APIs" in *Services Gatekeeper Portal Developer's Guide* for some Groovy code examples.

Prohibited Components in Groovy Actions

Your Groovy actions are validated to prevent security vulnerabilities. These Java and Groovy interfaces and methods, and any class that inherits them are prohibited in Groovy actions.

- **java.lang.Thread**
- **java.lang.Runnable**
- **java.lang.ClassLoader**
- **java.lang.Class**
- **java.io**
- **java.net**
- **groovy.lang.GroovyShell**
- **groovy.util.Eval**
- **groovy.io**
- **groovy.net**
- **exit**
- **setProperty**

Json2Xml

This action takes text formatted for the JSON protocol from the body of the incoming request or response body, and translates it into XML format in the body of the outgoing request or response. Table 5–2 lists the parameters for this action:

Table 5–2 Json2Xml Action Parameters

Parameter Name	Description
Instance Id	(Optional) A value that you add that is included in the event EDR. It identifies the action for debugging.
Default Name Space	(Optional) An identifier (usually a URI). This value is added to the first XML tag as the default namespace.
Root Tag	(Optional) Encloses the translated XML in a tag named for the value of this parameter. Used to avoid multiple roots in the translated XML.

For example, this JSON formatted text:

```
{
  "SendSms": {
    "message":"my message 2",
    "senderName‟ : "senderName",
    "addresses‟ : "tel:123456",
    "@myattribute‟ : "foo‟
  }
}
```

Is translated in this XML format:

```
<SendSms myattribute="foo">
  <addresses>tel:123456</addresses>
  <senderName>senderName</senderName>
  <message>my message 2</message>
</SendSms>
```

See the "Xml2Json" action to translate XML input to JSON.

RateLimit

Restricts access to a URL for HTTP-to-HTTP communication. You set the allowable number of accesses for a configurable time period for a specific URL.

You set **rate** and **timePeriodInMs** (time period in milliseconds) parameters. **rate** sets the number of request messages that can be sent to the URL in the time period set by **timePeriodInMs**. For example, a **rate** of 100 and a **timePeriodInMs** of 60000, allows 100 request messages per minute.

You can further refine access to the URL by using multiple **RateLimit** actions. For example you can set an overall rate limit of 1000 messages per day, and also ensure that the URL is protected during peak hour by adding another **RateLimit** of 1 per minute.

SchemaValidation

Services Gatekeeper validates an incoming request or outgoing response that accesses a web service API, based on the schema provided by you for the API.

Provide values for the first XSD in the fields under the unit numbered **0**. The first in the list is the main XSD/WADL/WSDL. The other entries are referenced from the first entry in the list.

1. For the **Content** parameter, enter the actual XML Schema XSD content. Paste in a Schema in the **Value** column.

2. For the **Name** parameter, enter the reference to the entry in **Content**. Use this name from another action or Groovy action to retrieve the schema you entered for **Content**.

To add more units, click the **-** sign next to **Un....** and repeat.

> **Note:** If you add the SchemaValidation action to a flow but you do not provide a schema definition for the web service API, Services Gatekeeper returns an error.

Throttling

Change a partner group name, rate (requests/second), quota period (days), and quota (requests per quota period) specified in the message. The data you are changing comes from the appropriate SLA.

Provide values for the first partner group under the fields for the unit numbered **0**:

1. For the **Group Name** parameter, enter the name of the partner group.
2. For the **Quota** parameter, enter the number of requests per quota period allowed.
3. For the **Quota Period** parameter, enter the number of days in the quota period allowed.
4. For the **Rate** parameter, enter the number limit for the number of requests per second allowed.

To add more throttle units, right-click the **-** sign next to **Un....** and repeat.

Xml2Json

This action takes text formatted for the XML protocol from the body of the incoming request or response body, and translates it into JSON format in the body of the outgoing request or response. Table 5–3 lists the parameters for this action.

Table 5–3 Xml2Json Action Parameters

Parameter	Description
Instance ID	A value that you add that is included in the event EDR. It identifies the action for debugging.

For example, this XML formatted text:

```
<SendSms myattribute="foo">
  <addresses>tel:123456</addresses>
  <senderName>senderName</senderName>
  <message>my message 2</message>
</SendSms>
```

Is translated into this JSON formatted text:

```
{
  "SendSms": {
    "message":"my message 2",
    "senderName" : "senderName",
    "addresses" : "tel:123456",
    "@myattribute" : "foo"
  }
}
```

See the "Json2Xml" action to translate JSON input into XML format.

XSLT

Use an Extensible Stylesheet Language script to change the XML-formatted body of the message.

DAF Callout Callback

For asynchronous `send()` calls, two actions are generated for each ServiceCallOut, allowing the response to be accessed other than in the subsequent action. In addition to the current `oracle.sdp.daf.Callout.Callback` class, an additional DAF (Dynamo Application Framework) callback interface called `oracle.sdp.daf.action.api.CalloutCallbackHandler` is shown in the following example:

```
package oracle.sdp.daf.action.api;

/**
 * Used for in-action callouts' callback processing.
 */
public interface CalloutCallbackHandler {
  /**
   * Called on callout completion.
   * @param context Context.
   * @throws ActionProcessingError Exception during callback
   * @since 7.0.0.0
   */
  void process(HttpContext context) throws ActionProcessingError;
}
```

To use it, the `oracle.sdp.daf.action.api.CalloutBuilder` interface has been extended to have two more methods for callback handler setting and retrieval. The following example illustrates the extensions.

```
package oracle.sdp.daf.action.api;
...
/**
 * Callout builder. Used to build a callout HTTP request.
 */
public interface CalloutBuilder extends MessageBuilder {
  /**
   * Adds callback handler.
   * @param callbackHandler The callback for this builder
   * @return The builder
   * @since 7.0.0.0
   */
  default CalloutBuilder withCallback(CalloutBuilderCallbackHandler
callbackHandler) {
    // return unmodified builder for those classes which don't implement this
    return this;
  }

  /**
   * Gets the callback handler associated with this builder.
   * @return The callback handler
   * @since 7.0.0.0
   */
```

```
  default CalloutBuilderCallbackHandler getCallbackHandler() {
    // return null for those classes which don't implement this
    return null;
  }
...
```

Limitation

The implementation of the DAF Callout Callback assumes that DAF Action could contain only one asynchronous callout or send call. The presence of multiple asynchronous callouts or `send()` calls within the same DAF action could lead to unknown results.

Example of Callback

The following example demonstrates how to implement the callback class and also how to retrieve the response from a context attribute that was previously set in an asynchronous call.

```
package example;

public class MyCallback implements CalloutCallbackHandler {
  @Override
  public void process(HttpContext context) {
    System.out.println("---MyCallback process call");
    System.out.println("---MyCallback context.getAttribute(): " +
context.getAttribute("TOKENADMIN_GET_RESPONSE"));
  }
}
```

The following example shows how it would be added in Groovy code using extra `.withCallback(callback)` syntax:

```
final String groovyCode1 =
  "try {\n"
  + "example.MyCallback callback = new example.MyCallback();\n"
  + "    String queryString = context.clientRequest.getQueryString();\n"
  + "    String apics_appkey = (String)context.clientRequest.queryParameters.apics_appkey\n" + "\n"
  + "System.out.println(\"Calling TokenAdmin with apics_appkey = \" + apics_appkey);\n"
  + "      // Get existing data from TokenAdmin\n"
  + "      context.setAttribute(\"CALLOUT_STATUS\", \"TOKENADMIN_GET\");\n"
  + "
((oracle.sdp.daf.action.api.ExternalCalloutBuilder)(context.createCallout().withRequestUrl('"
  + "http://localhost:" + getPort() + ROOT_CONTEXT_PATH + "/callout/callouturl')"
  + ".withQueryParameter(\"apics_appkey\", apics_
appkey).withRequestMethod(\"GET\").withHeader(\"Content-Type\","
  + "\"text/plain\").withCallback(callback))).build().send(\"TOKENADMIN_GET_RESPONSE\");\n"
  + "\n"
  + "}\n" + "catch (Exception e) {\n"
  + "    e.printStackTrace();\n"
  + "    throw e;\n" + "}";
```

Exception Handling

The following are exception handling considerations:

- The DAF Action interface provides the following entry point method

  ```
  void process(HttpContext context) throws ActionProcessingError;
  ```

The method throws `ActionProcessingError` to indicate that the processing flow should be cancelled.

- `CalloutCallback` code, which is available to HTTP AsyncClient and implements `FutureCallback<HttpResponse>)`, on request failure, at least for the case `java.net.ConnectException:Connection refused`, invokes `public void failed(Exception e)` method and through it a `sendErrorResponseToApp(errorCode)` call.

  The Javadoc output for `sendErrorResponseToApp(errorCode)` further illustrates:

```
package oracle.sdp.daf;
...
public final class CalloutCallback implements FutureCallback<HttpResponse> {
...
  /**
   * Send an error to the app - something went wrong. If this is within an
   * action, keep going, otherwise set the error flag on the utilcontext so
   * that no actions will be processed.
   *
   * @param statusCode
   *          HTTP status code to return in the response
   */
  private void sendErrorResponseToApp(int statusCode) {
   ...
      internalHttpContext.getClientResponse().withStatus(statusCode);
      if (callback != null) {
        // this is the southbound call
        internalHttpContext.setInternalError(true);
      }
      internalSystemContext.getActionChainManager().continueActionChain(
          internalHttpContext.getSouthboundResponse(), internalHttpContext);
   ...
```

  In the case of a southbound call, the call above to `internalHttpContext.setInternalError(true)` is analyzed inside `oracle.sdp.daf.ActionChainManager` and it's `doResponseChain(...)` method, which are shown here:

```
package oracle.sdp.daf;
...
public final class ActionChainManager {
...
  private void doResponseChain(HttpResponse response,
      InternalHttpContext internalHttpContext) {
...
        if (!internalHttpContext.isInternalError()) {
...
```

- When doing an asynchronous `send()` and a long invocation on the south side leads to a timeout, an HTTP asynchronous client will potentially throw `java.net.SocketTimeoutException`. An example of a `java.net.SocketTimeoutException` stack trace for an HTTP asynchronous client is shown here:

```
java.net.SocketTimeoutException
    at
org.apache.http.nio.protocol.HttpAsyncRequestExecutor.timeout(HttpAsyncRequestE
xecutor.java:376)
    at
org.apache.http.impl.nio.client.InternalIODispatch.onTimeout(InternalIODispatch
.java:92)
    at
```

```
org.apache.http.impl.nio.client.InternalIODispatch.onTimeout(InternalIODispatch
.java:39)
    at
org.apache.http.impl.nio.reactor.AbstractIODispatch.timeout(AbstractIODispatch.
java:175)
    at
org.apache.http.impl.nio.reactor.BaseIOReactor.sessionTimedOut(BaseIOReactor.ja
va:263)
    at
org.apache.http.impl.nio.reactor.AbstractIOReactor.timeoutCheck(AbstractIOReact
or.java:492)
    at
org.apache.http.impl.nio.reactor.BaseIOReactor.validate(BaseIOReactor.java:213)
    at
org.apache.http.impl.nio.reactor.AbstractIOReactor.execute(AbstractIOReactor.ja
va:280)
    at
org.apache.http.impl.nio.reactor.BaseIOReactor.execute(BaseIOReactor.java:104)
    at
org.apache.http.impl.nio.reactor.AbstractMultiworkerIOReactor$Worker.run(Abstra
ctMultiworkerIOReactor.java:588)
    at java.lang.Thread.run(Thread.java:745)
```

Some `oracle.sdp.daf.CustomHttpRequestRetryHandler` logic will attempt to retry:

```
[INFO ] pRequestRetryHandler  The maximum number of retry times is 1, current
execution count is 1
[INFO ] pRequestRetryHandler  isRquestSent is false
```

If it fails again, logic will route through the `public void failed(Exception e)` method mentioned above.

- As previously mentioned, the Callout callback handler interface has the same ability as DAF Action to throw ActionProcessingError and, in that case, action chain processing will be cancelled.

DAF Support of HTTP Methods

The following HTTP methods provide support of DAF (Dynamo Application Framework) features:

- PATCH
- HEA
- TRACE
- CONNECT
- OPTION

PATCH Method

You can use the PATCH method to update partial resources. For example, you might want to update only one field of a resource. Using PUT to update the complete representation of a resource could be cumbersome.

Supporting the PATCH method allows a PATCH request to pass through DAF. The Portal is updated with the new PATCH method in the resource table.

DAF treats the PATCH method the same as other HTTP methods like POST and PUT, meaning that the PATCH method is transparent to DAF. DAF treats a PATCH request as follows:

1. Transfers the request from the client to the backend service.
2. Transfers the response from the backend service to the client.

HEAD Method

The HTTP HEAD method corresponds to a GET request but without the response body. The HEAD method use cases are:

- Provides a faster way to check the headers
- Provides a LastModified / ContentLength check to decide whether to re-download a given resource
- Provides ability to log JavaScript-based content appearances

DAF treats the HEAD method the same as other HTTP methods like POST and PUT, meaning that the HEAD method is transparent to DAF. DAF treats a HEAD request as follows:

1. Transfers the request from the client to the backend service.
2. Transfers the response from the backend service to the client.

TRACE Method

The TRACE method, which is used for debugging, echoes input back to the user.

You must do the following to set up TRACE in your environment:

1. Enable TRACE support in WebLogic to avoid error 501. You can enable TRACE support in the following ways:
   a. To enable TRACE manually, in the WebLogic console, select your domain -> Configuration -> WebApplication, click the **Http Trace Support Enabled** box and select **Save**.
   b. To enable TRACE in Java, set the `HttpTraceSupportEnabled` flag in the **WebAppContainerMBean**.

   DAF treats the TRACE method the same as other HTTP methods like POST and PUT, meaning that the TRACE method is transparent to DAF. DAF treats a TRACE request as follows:

   1. Transfers the request from the client to the backend service.
   2. Transfers the response from the backend service to the client.

CONNECT Method

Use the CONNECT method to create an HTTP tunnel.

In this mechanism, the client sends an HTTP CONNECT request to the OCSG server to forward the TCP connection to the desired destination. The server proceeds to make the connection on behalf of the client. Once the server establishes the connection, an HTTP 200 response is sent to the client. The OCSG server continues to proxy the TCP stream by other HTTP methods to and from the client. Note that only the initial connection request is HTTP. After that, the server simply proxies the established TCP connection.

DAF treats the CONNECT method the same as other HTTP methods like POST and PUT, meaning that the CONNECT method is transparent to DAF. DAF treats a CONNECT request as follows:

1. Transfers the request from the client to the backend service.
2. Transfers the response from the backend service to the client.

OPTION Method

Use the OPTION method to list available interfaces.

You can make three types of OPTION requests:

1. Preflight request for CORS action. The CORS action responds directly. No change needed.
2. Request to list available interfaces for an API. You must add a ListResourcesAction object to the request to handle this.
3. Other types of requests are passed through DAF to the backend service.

Configure CORS parameters as shown in Table 5–4:

Table 5–4 CORS Parameters

CORS Parameter	Value
Allow Any Origin	False
Allowed Methods	OPTION
Allowed Origins	www.google.com
Support Preflight Request	true

HTTP Header Filter

Some headers in an incoming (northbound) client request cannot be transferred to the back-end service, and the corresponding response headers cannot be sent back to the client. The following list of headers are filtered from a client request when transferring the request to the back-end service, and from the back-end response when the reply is returned to the client.

Note: This filter list does not apply to a Groovy or dynamic action with the following APIs when sending to a back end or client:

```
oracle.sdp.daf.action.api.MessageBuilder#withHeader(String, String)
addHeader(String, String)
withHeaders(Map)
```

- proxy-authorization
- authorization
- content-length
- transfer-encoding
- cookie
- connection
- set-cookie

- host
- ocsgoauthbearer
- ocsgoauthmac
- anonymous
- ocsgproxy-authorization
- ocsgsoapheader
- ocsgappkeyheader

Common Actions Programming Tasks

This section provides more detailed information about tasks that you will probably perform on the request and response messages that action chains.

Also see "Creating Custom Actions for Your APIs" in *Services Gatekeeper Portal Developer's Guide* for some Groovy code examples.

Printing and Changing Message Content

To print the contents of a message in the *request* action chain, you use the **getClientRequest** operation to the **httpContext** class, with the **getBodyAsType** operation to **messageBuilder** class. For example:

```
println httpContext.getClientRequest().getBodyAsType(String.class)
```

Assume that a RESTful request message contains:

```
{
  "test" : {
    "bob" : "123"
  }
}
```

You could use this Groovy script to return the value **123**:

```
def body =
context.clientRequest.getBodyAsType(org.codehaus.jackson.JsonNode.class);
println body.get("test").get("bob").getTextValue()
```

To print the contents of a message in the *response* action chain, you use the **getSouthBoundResponse** operation to the **httpContext** class, with the **getBodyAsType** operation to the **messageBuilder** operation. For example:

```
println httpContext.getSouthboundResponse().getBodyAsType(String.class)
```

In a Groovy script you could use:

```
HttpResponse httpResponse = (HttpResponse) context.getAttribute("myattribute");
```

And then:

```
def responseBody =
httpResponse.getBodyAsType(org.codehaus.jackson.JsonNode.class);
```

To change a request action chain message value, you use:

- The **withBodyAsObject(Object)** operation. Once changed, you then read only the changed value using the **getBodyAsObject(Object)** operation. Both are in the messageBuilder class.
- The **withBodyAsStream(input stream)** operation. Once changed, you then read only the changed value with the **getBodyAsStream** operation. Both are in the **messageBuilder** class.

See the "All Classes" section of the *Actions Java API Reference* for details on all of these operations and classes.

Using Actions to Manipulate HTTP Query Parameters

During processing by Services Gatekeeper, you can set encoded HTTP query parameters for a request message using any of these components. The components are processed in this order so the last in order makes the final changes:

1. The URL in the original request message.
2. The API Service URL that you specify when you create or update the API.
3. The API resource Path that you set in the **Resources** table when you create or update the API
4. An action. You can use either the default Groovy action, or a custom action that you create for this purpose. There are several purpose-built methods in the Actions Java API Reference for this purpose. See "Using a Groovy or Custom Action to Manipulate Query Parameters" for details on these methods. Also see "Creating Custom Actions for Your APIs" in *Services Gatekeeper Portal Developer's Guide* for information on how to create a custom action.

> **Note:** Services Gatekeeper requires encoded query parameters. If you add or change any query parameters, ensure that they are encoded, using no blank spaces or "&" characters. REST-based URLs are encoded by default, but if you add or change any values be sure they are encoded.

Using a Groovy or Custom Action to Manipulate Query Parameters

The Groovy action within an API is the logical place to make changes to request query parameters, because it is the last processing performed for the message. You can use these operations from the **CalloutBuilder** class for the Actions Java API to manipulate query parameters:

- Return all query parameters by using **getQueryParameters**.
- Return a query string of parameters by using **getQueryString**.
- Return a single parameter for a key by using **getQueryParameter**.
- Add or overwrite a query parameters using **withQueryParameter**.
- Add or overwrite a query parameter string by using the **withQueryString**.
- Send a response message without a specific query parameters by using **withoutQueryParameter**.
- Ignore all query parameters in the original request message using the **ignoreAllRequestQueryParameter** flag.

For details on these methods, see the **CalloutBuilder** class in the "All Classes" section of *Actions Java API Reference*.

Groovy Query Manipulation Code Examples

These examples use these values for query parameters listed in Table 5–5.

Table 5–5 Example HTTP Query Parameters (Key=Value Pairs)

Request Message	API Service URL	API Resource Service Path
reqQueryStr1=reqV1	NA	NA
NA	suQueryStr2=suV2	NA
NA	NA	pathQueryStr3=spV3
reqQueryStr4=reqV4	reqQueryStr4=suV4	reqQueryStr4=spV4
NA	suQueryStr5=suV5	suQueryStr5=spV5
reqQueryStr6=reqV6	NA	reqQueryStr6=spV6

Example 5–1 changes the **pathQueryStr3** parameter value set by the API Service Path in Table 5–5 to **actionV3_1**. The full list of query parameters after processing are listed in Table 5–6.

Example 5–1 Using withQueryString to Change a Parameter

```
context.getSouthboundCallout().withQueryString("pathQueryStr=actionV3_1");
```

Table 5–6 HTTP Query Parameters After the QueryString Operation

Key	Value
reqQueryStr1	reqV1
suQueryStr2	suV2
pathQueryStr3	**actionV3_1**
reqQueryStr4	spV4
suQueryStr5	spV5
reqQueryStr6	spV6

Example 5–2 shows how to change a string of HTTP query parameters using **withoutQueryParameter**. Table 5–7 shows the new values after the **withoutQueryParameter** operation. The **reqQueryStr4** key/value pair has been removed. Notice that because **withoutQueryParameter** (**pathQueryStr3**) is processed *before* **withQueryParameters**, **pathQueryStr3** maintains its value.

Example 5–2 withoutQueryParameter Operation

```
context.getSouthboundCallout().withoutQueryParameter("pathQueryStr3");
context.getSouthboundCallout().withQueryString("pathQueryStr3=actionV3_1");
context.getSouthboundCallout().withQueryParameter("reqQueryStr4=actionV4_2");
context.getSouthboundCallout().withoutQueryParameter("reqQueryStr4");
```

Table 5–7 HTTP Query Parameters After the withoutQueryParameter Operation

Key	Value
reqQueryStr1	reqV1
suQueryStr2	suV2

Table 5–7 (Cont.) HTTP Query Parameters After the withoutQueryParameter Operation

Key	Value
pathQueryStr3	actionV3_1
suQueryStr5	spV5
reqQueryStr6	reqV1

Example 5–3 shows how **ignoreAllRequestQueryParameters** affects the list of query parameters. As Table 5–8 shows, it removed all of the query parameters set by the original request message (all **reqQueryStr***n* key/values from Table 5–5).

Example 5–3 ignoreAllRequestQueryParameters Operation

```
context.getSouthboundCallout().ignoreAllRequestQueryParametersb(true);
```

Table 5–8 HTTP Query Parameters after ignoreAllRequestQueryParameters Operation

Key	Value
suQueryStr2	suV2
pathQueryStr3	spV3
reqQueryStr4	spV4
suQueryStr5	spV5
reqQueryStr6	spV6

Using Actions to Translate Between REST and SOAP

This section explains how to map between SOAP and REST communication using the tools in an actions chain.

REST to SOAP Translation

You use these general steps to translate a REST message to a SOAP message through an actions chain:

1. Use the **Json2Xml** action to translate data into the JSON format.
2. Use the **SchemaValidation** action to verify the data. You must create your own schema file for this action.
3. Use the **XSLT** action to complete the transition to a SOAP format. You create this script.

These actions translate the data to the appropriate formats, but they cannot compensate for the fundamental difference between SOAP and REST communication. REST is resources-based, and depends on methods (GET, POST, PUT, DELETE) to act on resources. SOAP is a much more flexible standard, and not limited to methods. You must convert the REST tasks to tasks that your SOAP components can use by creating an XSLT script for the **XSLT** action.

During the translation you must think about how to translate the three main components of a REST message into a format that a SOAP-based program can use:

- The request URI
- The HTTP method (GET, POST, PUT, DELETE)
- The message body

Example 5–4 shows code snippets for a simple REST to SOAP translation example. It shows the actions used to translate the x-1 and y-2 key-value pairs from REST format to XML format that SOAP can use.

Example 5–4 A Simple REST to SOAP Translation Using Actions

Original REST Input:

```
{
  "envelope": {
      "x":1,
      "y":2
  }
}
```

After Json2Xml action:

```
<envelope>
   <y>2</y>
   <x>1</x>
</envelope>
```

After XSLT action (final XML output):

```
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cal="http://www.parasoft.com/wsdl/calculator/">
<soapenv:Header/>
<soapenv:Body>
<cal:add>
<cal:x>1</cal:x>
<cal:y>2</cal:y>
</cal:add>
</soapenv:Body>
</soapenv:Envelope>
```

This the **XSLT** action script used in Example 5–4 to translate the JSON data to XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/">

    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cal="http://www.parasoft.com/wsdl/calculator/">
      <soapenv:Header/>
      <soapenv:Body>
        <cal:add>
          <cal:x><xsl:value-of select="envelope/x"/></cal:x>
          <cal:y><xsl:value-of select="envelope/y"/></cal:y>
        </cal:add>
      </soapenv:Body>
    </soapenv:Envelope>
  </xsl:template>

</xsl:stylesheet>
```

SOAP to REST Translation

This task is a reverse of the "REST to SOAP Translation" procedure, but it can be significantly harder because of the nature of REST and SOAP communication. REST communication is based on a small number of methods (such as GET, POST, PUT, and DELETE). SOAP is a much more flexible standard. In order to make SOAP communication work for REST, you must translate all SOAP actions into one of the REST methods in the **XSLT** action. This translation depends entirely upon your implementation.

You use these general steps to translate a SOAP message to the REST format in an actions chain:

1. Use the **XSLT** action to translate data into XML format.
2. Use the **SchemaValidation** action to verify the data. You must create your own schema file for this action.
3. Use the **Xml2Json** action to complete the transition to a format that a REST program can use.

During the translation you need to think about how to translate the three main components of a REST message into a format that a SOAP-based program can use:

- The request URI
- The HTTP method (GET, POST, PUT, DELETE)
- The message body

Transfering Data from Request Chain to Response Chain

The following example shows you how you can transfer data from a request to a response, using a Request and a Response Groovy action (serializable is required on multi-tier):

Request:

```
Object obj = new Object();
context.setAttribute("key", obj)
```

Response:

```
Object obj = (Object) context.sgetAttribute("key")
```

Converting JSON and XML

The Xml2Json action converts incoming XML body content to JSON body content. The following example illustrates:

```
XML Body Content In:
<SendSms myattribute="foo">
 <addresses>tel:123456</addresses>
 <senderName>senderName</senderName>
 <message>my message 2</message>
</SendSms>

JSON Body Content Out:
{
"SendSms": {
 "message":"my message 2",
 "senderName" : "senderName",
 "addresses" : "tel:123456",
 "@myattribute" : "foo"
```

```
 }
}
```

The Json2XML action transforms JSON to XML. The following example illustrates:

```
JSON Body Content In:
{
"SendSms": {
 "message":"my message 2",
 "senderName" : "senderName",
 "addresses" : "tel:123456",
 "@myattribute" : "foo"
 }
}

XML Body Content Out:
<SendSms myattribute="foo">
 <addresses>tel:123456</addresses>
 <senderName>senderName</senderName>
 <message>my message 2</message>
</SendSms>
```

Table 5–9 describes the three Json2XML optional configuration attributes:

Table 5–9 Json2XML Optional Configuration Attributes

Attribute	Description	Data Type
Instance ID	ID of this action instance	String
Default Name Space	A URI that is added to the first XML tag as the default namespace. The URI format is not enforced.	String
Root Tag	A value that is used to create an enclosing tag to hold the converted XML. Used to avoid multiple roots in the transformed XML	String

Accessing the Customized Data Store

OCSG provides the ability to read and delete data from the customized data store, the database table `PRM2_CUSTOMIZEDDATA` shown in Figure 5–1:

Figure 5–1 PRM2_CUSTOMIZEDDATA Table

#	Column Name	Data Type	Length	Not Null	Auto Generated	Auto Increment	Default	Key	Description
1	THEKEY	VARCHAR	1,024	☑	☐	☐		☐	
2	DATA	VARCHAR	4,000	☐	☐	☐		☐	
3	STORED_TS	BIGINT	19	☑	☐	☐		☐	

Currently, only a key that is a simple string and a value that is a string are supported.

The class `oracle.ocsg.daf.store.CustomizedDataHelper` enables you to easily perform create, retrieve, update, and delete (CRUD) operations on the data store. This class is contained in **wlng.jar**, which is under the directory **{OCSG_INSTALL}/ocsg/server/lib/wlng**.

This class includes the following Java APIs:

- `public CustomizedDataHelper getInstance()`

  Gets the singleton `CustomizedDataHelper` instance.

  Returns the singleton instance.

- `public CustomizedData storeCustomizedData(CustomizedData data) throws StorageException`

  Stores the `CustomizedData` instance.

  Parameters:

  – `data` The `CustomizedData` record to store.

  Returns:

  The previous value associated with `theKey` element. or null if there was no mapping for `theKey`.

  Throws:

  `StorageException` when storing the data fails

- `public CustomizedData retrieveCustomizedData(final String key) throws StorageException`

  Retrieves the `CustomizedData` specified by `key`.

  Parameters:

  – `key` A string to match `theKey` element of the CustomizedData record to be retrieved.

  Returns:

  The matched `CustomizedData` record, if found; otherwise, null.

  Throws:

  `StorageException` when retrieval fails

- `public CustomizedData deleteCustomizedData(final String key) throws StorageException`

  Deletes the `CustomizedData` record specified by `key`.

  Parameters:

  – `key` A string to match `theKey` element of the CustomizedData record to be deleted.

  Returns:

  The deleted `CustomizedData` instance, if found; otherwise null.

  Throws:

  `StorageException` - When failing to delete the record.

- `public CustomizedData updateCustomizedData( String key, CustomizedData data) throws StorageException`

  Updates the `CustomizedData` record specified by `key`.

  Parameters:

  – `key` A string to match `theKey` element of the CustomizedData record to be updated.

  – `data` The data to update.

Returns:

The old `CustomizedData` record.

Throws:

`StorageException` - When failing to update the record.

- `public List<CustomizedData> retrieveCustomizedDataItems(String keySearchString, SearchStringOperation operation) throws StorageException`

  Gets a list of stored `CustomizedData` items whose `theKey` element matches the search string.

  Parameters:

  `keySearchString` The search string to match `theKey` of `CustomizedData` records.

  `SearchStringOperation` One of the following enumerators that specify how the value of keySearchString should be evaluated:

  - `SearchStringOperation.STARTSWITH`: The key starts with the search string. Null or empty ("") matches all records.
  - `SearchStringOperation.ENDSWITH`: the key ends with the search string. Null or empty ("") matches all records.
  - `SearchStringOperation.CONTAINS`: the key contains the search string. Null or empty ("") matches all records.
  - `SearchStringOperation.LIKES`: the key likes the search string.

  Returns:

  A list of `CustomizedData` instances that match the search criteria. If no records match the search string, the list will be empty.

  Throws:

  `StorageException` - When an error occurs executing the query.

- `public int deleteCustomizedDataItems(String keySearchString, SearchStringOperation operation) throws StorageException`

  Deletes the list of stored `CustomizedData` items whose `theKey` element matches the search string.

  Parameters:

  `keySearchString` The search string to match `theKey` element of `CustomizedData` records.

  `SearchStringOperation` One of the following enumerators that specify how the value of keySearchString should be evaluated:

  - `SearchStringOperation.STARTSWITH`: The key starts with the search string. Null or empty ("") matches all records.
  - `SearchStringOperation.ENDSWITH`: the key ends with the search string. Null or empty ("") matches all records.
  - `SearchStringOperation.CONTAINS`: the key contains the search string. Null or empty ("") matches all records.
  - `SearchStringOperation.LIKES`: the key likes the search string.

  Returns:

The list of `CustomizedData` instances that match the search criteria. If no records match the search string, the list will be empty.

Throws:

`StorageException` - When error occurs executing the query.

Examples

Assume that Table 5–10 contains the data that we have in our PRM2_CUSTOMIZEDDATA table:

Table 5–10 Sample Data in PRM2_CUSTOMIZEDDATA Table

theKey	Data	Stored_TS
search1String1	value1	1,492,756,468,312
search2String2	value2	1,492,756,468,313
searchString3	value3	1,492,756,468,314
searchString4	value4	1,492,756,468,315

The following examples illustrate how this data is affected by various operations.

The first example gets the rows in which the key starts with the specified prefix:

```
List<CustomizedData> dataList =
CustomizedDataHelper.getInstance().retrieveCustomizedDatas("searchString",
SearchStringOperation.STARTSWITH);
```

Result: Returns the third and fourth rows.

The second example deletes the rows in which the key ends with the specified post-fix.

```
int deletedRowNum =
CustomizedDataHelper.getInstance().deleteCustomizedDatas("String4",
SearchStringOperation.ENDWITH);
```

Result: `deletedRowNum` = 1 and the fourth row is removed.

The third example retrieves the rows in which the key contains the specified string:

```
List<CustomizedData> dataList =
CustomizedDataHelper.getInstance().retrieveCustomizedDatas("String",
SearchStringOperation.CONTAINS);
```

Result: Retrieves all four rows.

The fourth example deletes the rows in which the key likes the specified string.

```
int deletedRowNum1 =
CustomizedDataHelper.getInstance().deleteCustomizedDatas("searchString",
SearchStringOperation.LIKES);
```

Result: deletedRowNum1= 0 and no rows are deleted.

The fifth example gets rows when the search string is null or empty.

```
List<CustomizedData> allDataList1 =
CustomizedDataHelper.getInstance().retrieveCustomizedDatas(null,
SearchStringOperation.STARTSWITH);
```

Result: Retrieves all four rows

The sixth example deletes rows when the search string is null or empty.

```
int deletedRowNum1 = CustomizedDataHelper.getInstance().deleteCustomizedDatas("",
SearchStringOperation.ENDSWITH);
```

Result: `deletedRowNum1` equals 4, and all four rows are deleted

Configuring Chunking for Back-end Services

You can control the `chunked` setting for southbound callout in Transfer-Encoding, regardless of the incoming request. You can configure the setting globally and also at the request or API level.

> **Note:** The maximum southbound chunked callout size is 4097 bytes.

Global Configuration

You can set the `SouthBoundChunkedSetting` item in the `DafGeneralInformation` MBean to globally configure the chunked handling method once for all. It accepts the following three values, which are case insensitive:

- `PassThrough`, which is the default. In this case, the southbound callout will be chunked, or not, in accordance with the incoming request.
- `ForceChunk`, which forces Services Gatekeeper to choose chunked when sending southbound callout.
- ForceNoChunk, which forces Services Gatekeeper to not chunk when sending southbound callout.

Per Request or Per API Configuration

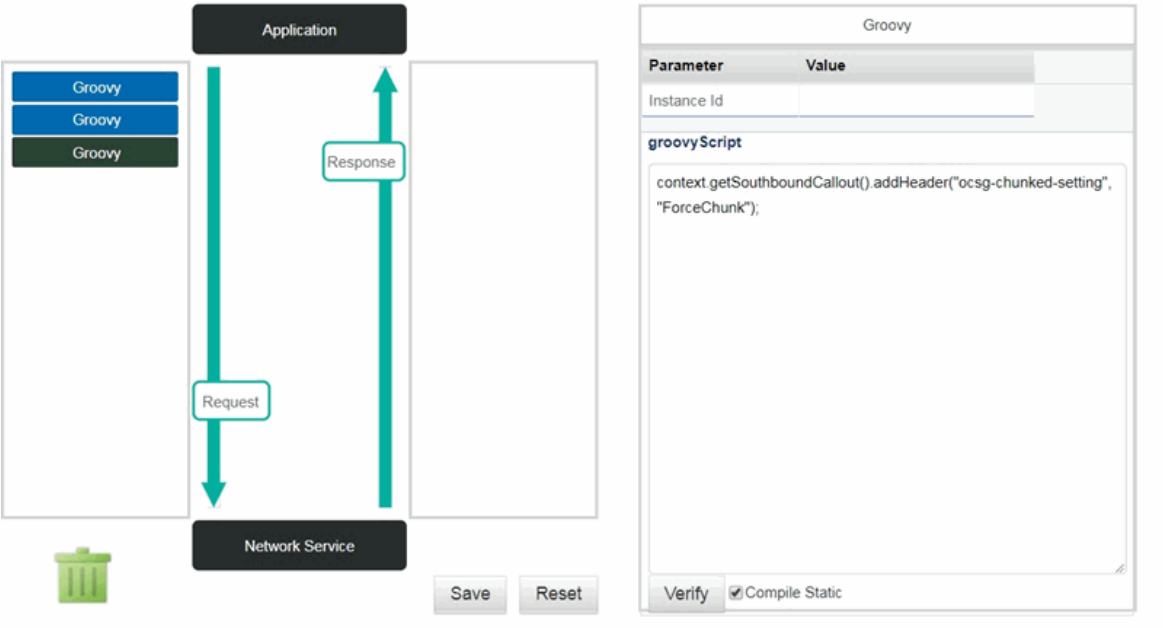
You can also configure chunking on a per request or per API basis to obtain more granularity. You can set `chunked` either with a special header, `ocsg-chunked-setting`, in the incoming request or use a Groovy action to add a special header, also called `ocsg-chunked-setting`, for a specified API. In both cases, post-processing uses this header to override the global setting.

The order of precedence is the Groovy action, followed by the traffic request, and then the global MBean setting.

The `ocsg-chunked-setting` header takes the same three values as the `SouthBoundChunkedSetting` item in the `DafGeneralInformation` MBean: `PassThrough`, `ForceChunk`, and `ForceNoChunk`.

In Figure 5–2, the Groovy script uses the `addHeader()` method to add the header. You can also use the `withHeader()` method.

Figure 5–2 Groovy Script Adding Chunking Header



For the Groovy action, instead of using a String to set the special header, you can use the following predefined constant or enums in class `oracle.sdp.daf.configurations.SouthBoundChunkedSetting`:

- `public static final String CHUNKED_HEADER = "ocsg-chunked-setting"`
- enum
  - `PassThrough`
  - `ForceChunk`
  - `ForceNoChunk`

The following example illustrates a Groovy setting:

```
context.getSouthboundCallout().withHeader(oracle.sdp.daf.configurations.SouthBound
ChunkedSetting.CHUNKED_HEADER,
oracle.sdp.daf.configurations.SouthBoundChunkedSetting.ForceChunk.toString())
```

Understanding the Troubleshooting Action Information in EDRs

Event Data Records (EDRs) contain information to specific to the action chain that you specify. See the **ReqAction** (Request Action) and **RspAction** (Response Action) EDR fields in "Understanding EDR Fields for API Management" in *Services Gatekeeper Administrator's Guide* for details.

6

Creating Custom HTTP Processors

This chapter explains how you can create custom HTTP processors for Oracle Communications Services Gatekeeper (OCSG).

Creating a Custom HTTP Processor

To write custom processors that perform authentication, assertion or other security related tasks that need to execute in the WebLogic security provider, the OCSG security provider has been refactored to make it lightweight for developing new authentication or assertion mechanisms.

The framework also allows the API to subscribe the available HTTP processors with the values for the configuration parameters. Each API can subscribe to more than one processor, and the same processor also can subscribe to multiple different configuration values.

There are two types of processors: *out-of-the box* and *custom.* There are also two scopes: *global* and *API*.

Currently, the out-of-the-box processors are:

- **ShieldHttpProcessor** (global): used to block IP addresses that have been blocked by the threat protection framework.
- **AnonymousHttpProcessor**: used for anonymous APIs
- **AppKeyHttpProcessor**: used for validating APPKeys
- **BasicAuthHttpProcessor**: handles basic authorization header
- **CORSProcessor**: handles CORS requests, always present unless API is anonymous
- **OauthProcessor**: handles Bearer and MAC OAuth tokens (only OCSG generated tokens)
- **SoapHttpProcessor**: handles `usernameToken` in WSSE Header

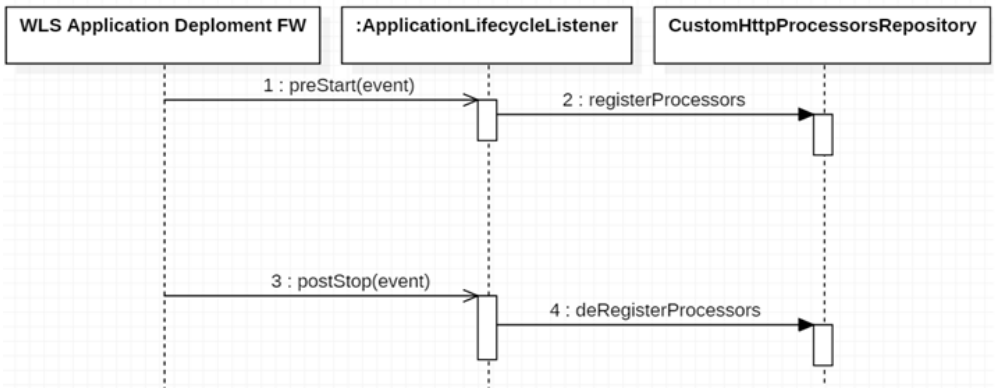
Deploying and Undeploying Custom HTTP Processors

Out-of-the-box HTTP processors do not need to register or unregister, while custom HTTP processors must.

Custom HTTP processor *must* extend the `ApplicationLifecycleListener` in its EAR. And the ear *must* be deployed to AT (cluster) in OCSG ENV.

The sequence invocations are repeated on each AT node as Figure 6–1 illustrates:

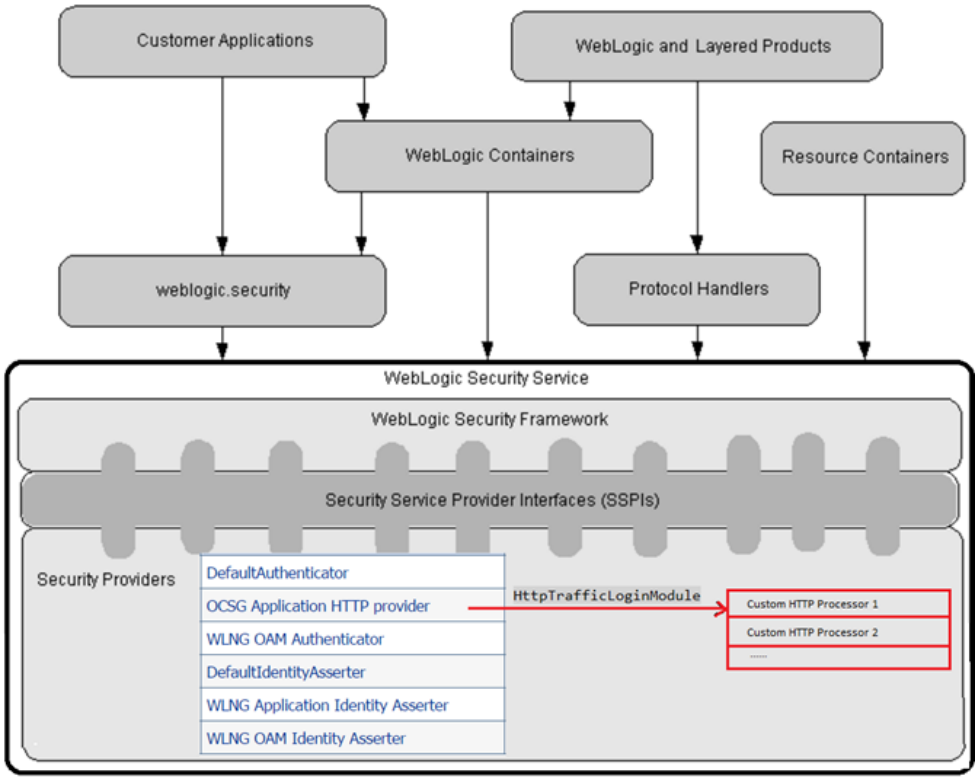
Figure 6–1 HTTP Processor Deployment/Undeployment Sequence



HTTP Processor Runtime Architecture

As Figure 6–2 shows, all of the HTTP processor instances are contained in the OCSG Application HTTP Provider:

Figure 6–2 HTTP Processor Runtime Architecture



In runtime, the processors are invoked from `HttpTrafficLoginModule`. The custom and out-of-the-box chains are invoked in that order. Both chains produce a set of principals that are merged. The custom chain is created based on a URL that maps to an API that might or might not have custom HTTP processors attached to it. The out-of-the-boxchain is created based on the API auth setting (`NONE`, `TEXT`, `OAUTH`, `APPKEY`).

When a request comes to the OCSG Application HTTP provider it is processed by four chains in the following order:

1. Out-of-the-box Global processors
2. Custom global processors
3. Custom per-API processors
4. Out-of-the-box per API processors

Each chain can be broken if a processor return `true` in `isDone`. If that happens, the next chain is executed.

If a processor throws a runtime exception, the next processor in the chain is executed.

If a processor throws `HttpProcessorViolationException`, all execution is aborted and the overall login or assertion fails.

Custom Processor EDRs

Each processor's execution is monitored by a tracker, and when all processors have executed, the tracker logs the execution to EDR map.

Each processor's execution has one of the following results:

- Success (no exception)
- Exception (if a runtime exception is thrown)
- Deny (if `HttpProcessorViolationException` is thrown)

If a login or assertion fails for any reason, the DAF error servlet fires an EDR.

> **Note:** This means that an unsuccessful login for other services will not generate this EDR. The reason is that even if the HTTP provider fails, OAM or SMPP could still be successful - so the provider itself cannot decide whether the EDR should be logged.

A login or assertion failure occurs when a security constraint is not met; that is, when the principal `TrafficUser` is not returned from any of the processors.

> **Tip:** In an EDR, you can track the number of principals each processor adds to the subject, and if all add 0 principals, you can be sure that a failed login occurred.

Example EDRs

Example 6–1 shows a sample EDR for the success outcome.

Example 6–1 Success

```
[04-21 03:52:20:DEBUG EdrInternalPublisher.java] *** EDR:
ServiceName = null
ContainerTransactionId = null
Method = null
Position = before
ServiceProviderId = partner
TransactionId = 57a211e0-032b-4730-a722-8935da86c9d0_IDX_1
State = ENTER_AT
Class = oracle.ocsg.daf.trafficlogger.SingleTierTrafficLogger
ApplicationId = weather
```

```
Processors = "seq=0, name=ShieldHttpProcessor, status=Success, new_principals=0",
"seq=1, name=JsonTokenProcessor, status=Exception,
code=NullPointerException:null", "seq=2, name=AppKeyHttpProcessor, status=Success,
new_principals=2", "seq=3, name=CORSProcessor, status=Success, new_principals=0"
TsBeAT = 1492782740932
HttpMethod = POST
Timestamp = 1492782740932
Direction = south
Source = method
URL = /ECHOServer/1/echo
ServiceProviderGroup = gold
AppInstanceId = appkey_Password1
ServerName = Server1
ReqMsgSize = 17
```

Example 6–2 shows a sample EDR for the exception outcome.

Example 6–2 Exception

```
[04-21 03:09:19:DEBUG EdrInternalPublisher.java] *** EDR:
ServiceName = null
ContainerTransactionId = null
Method = null
Source = null
Position = after
AccessUrl = http://10.88.42.23:8001/ECHOServer/1
https://10.88.42.23:8002/ECHOServer/1
TransactionId = 5d8a0a62-a9bb-4306-84dd-e2e3770564eb
Class =
com.bea.wlcp.wlng.security.providers.authentication.account.http.HttpTrafficLoginM
odule
Processors = "seq=0, name=ShieldHttpProcessor, status=Success, new_principals=0",
"seq=1, name=JsonTokenProcessor, status=Exception,
code=NullPointerException:null", "seq=2, name=AppKeyHttpProcessor, status=Success,
new_principals=0", "seq=3, name=CORSProcessor, status=Success, new_principals=0"
Timestamp = 1492780149789
ServerName = Server1
ApiId = ECHOServer
```

Example 6–3 shows a sample EDR for the deny outcome.

Example 6–3 Deny

```
[04-21 03:06:30:DEBUG EdrInternalPublisher.java] *** EDR:
ServiceName = null
ContainerTransactionId = null
DenyCode = 39
Position = after
AccessUrl = http://10.88.42.23:8001/ECHOServer/1
https://10.88.42.23:8002/ECHOServer/1
Method = POST
TransactionId = 86a3e481-4313-42e7-b1dc-f3b3e67a63a0
Class = ErrorServlet
Processors = "seq=0, name=ShieldHttpProcessor, status=Success, new_principals=0",
"seq=1, name=JsonTokenProcessor, status=Deny, code=39"
Timestamp = 1492779990517
URL = /ECHOServer/1/echo/
Source = Exception
ServerName = Server1
ApiId = ECHOServer
```

Implementing a Custom HTTP Processor

The framework provides an abstract class `AbstractHttpProcessor`, which you should use to implement customized HTTP processors.

> **Note:** There is an `HttpProcessor` interface available but you should never implement this interface directly; always extend `AbstractHttpProcessor`.

The `HttpProcessor` interface has some default methods, such as `update(T)`, `destroy()`, `getConfiguration()`. The general type `T` in the `HttpProcessor` is the configuration class of the processor implementation.

To Implement a Custom HTTP Processor:

1. You must use `HttpProcessorAnnotation` on the class level to specify the name for end-users, the configuration class name and the description of the processor.
2. The implementation must extend `weblogic.application.ApplicationLifecycleListener`

   Invoke `CustomHttpProcessorsRepository.INSTANCE.registerProcessor(Class)` to register itself in the `preStart(ApplicationLifecyclevEent)` method

   Invoke `CustomHttpProcessorsRepository.INSTANCE.deRegisterProcessor(String)` to register itself in the `postStop(ApplicationLifecycleEvent)` method
3. The configuration for the processor is optional.
4. The processor class implementation must invoke the constructor without parameters because in the framework, processor instances are initiated as `Class.forInstance()`.
5. The processor class must extend `AbstractHTTPProcessor` and implement

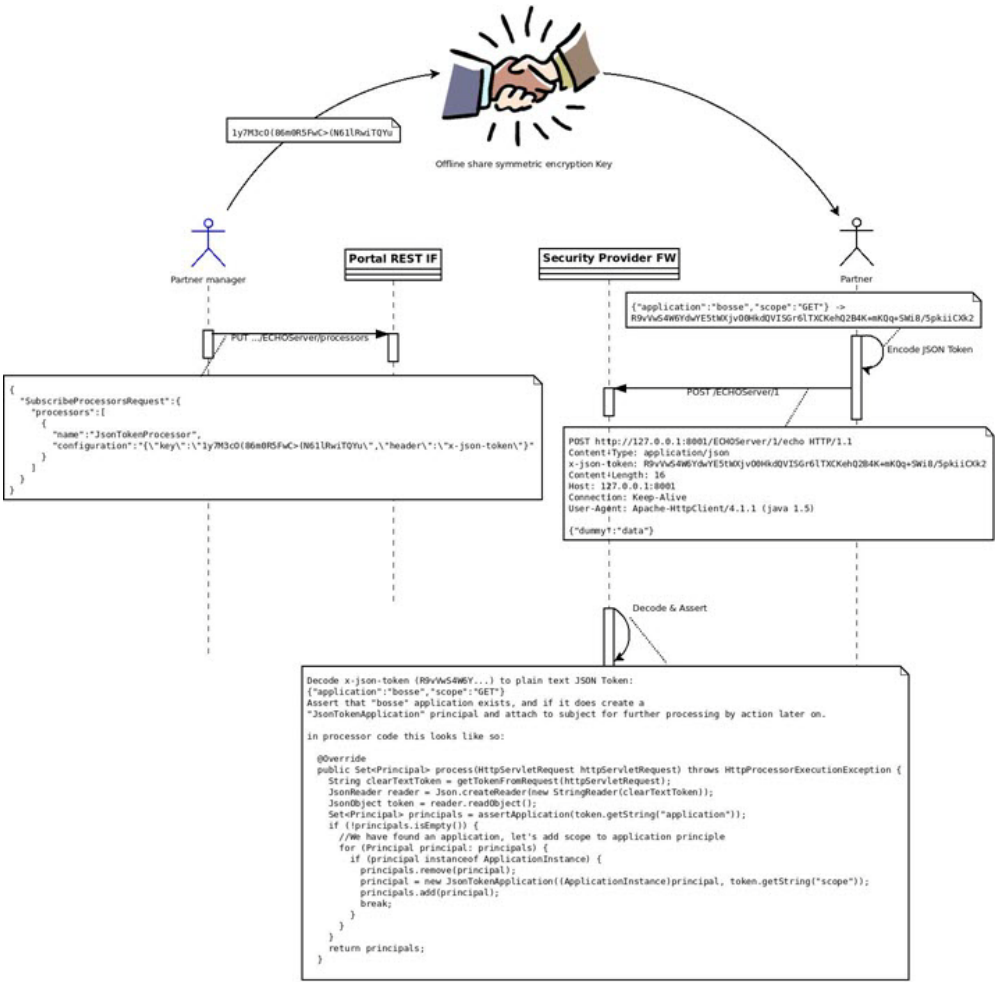
   ```
   Set<Principal> process(HttpServletRequest httpReq) throws
   HttpProcessorViolationException
   ```
6. Oracle strongly recommends that you implement the package as a WebLogic application (EAR file).

Example: A Symmetric Key Encrypted JSON Token

Figure 6–3 provides an overview use case to illustrate the concepts of custom HTTP processors. This is the overall flow:

1. Develop the custom processor EAR.
2. Deploy the custom processor EAR.
3. Come up with a header name (example: `x-json-token`) and encryption key (example: 1y7M3...) and share it with a partner offline.
4. Subscribe the processor to an API.
5. The partner application invokes the API.

Figure 6–3 Example HTTP Processor Architecture



OCSG includes a custom processor that is a deployable EAR file that has a life cycle class defined in **weblogic-application.xml**, shown in Example 6–4.

Example 6–4 weblogic-application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-application
 xmlns="http://xmlns.oracle.com/weblogic/weblogic-application"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-application
http://xmlns.oracle.com/weblogic/weblogic-application/1.4/weblogic-application.xsd
">
   <listener>
     <listener-class>
       oracle.ocsg.security.ProcessorLifecycleListener
     </listener-class>
   </listener>
</weblogic-application>
```

The listener class is responsible for registering, and deregistering all custom `HttpProcessors` that you want to want to add to OCSG.

Example 6–5 ProcessorLifecycleListener

```
/**
* Listener class for HTTP processor to register.
*/
public class ProcessorLifecycleListener extends ApplicationLifecycleListener {
  List<Class<? extends HttpProcessor>> processorClasses;

  public ProcessorLifecycleListener() {
    processorClasses = new ArrayList<>();
    processorClasses.add(JsonTokenProcessor.class);
  }

  /**
  * Register the Processor(s)
  */
  public void preStart(ApplicationLifecycleEvent evt) {
    registerProcessors(processorClasses);
  }

  /**
  * Unregister the Processor(s)
  */
  public void postStop(ApplicationLifecycleEvent evt) {
    deRegisterProcessors(processorClasses);
  }

  void registerProcessors(List<Class<? extends HttpProcessor>> processorClasses) {
    List<Class<? extends HttpProcessor>> registeredProcessors = new
      ArrayList<Class<? extends HttpProcessor>>();
    for (Class<? extends HttpProcessor> processorClass : processorClasses) {
      try {
        CustomHttpProcessorsRepository.INSTANCE.registerProcessor(processorClass);
        registeredProcessors.add(processorClass);

      } catch (Exception e) {
        // If we have exception, let's rollback all processors we already
        // registered before throwing exception.
        deRegisterProcessors(registeredProcessors);
        throw new RuntimeException(e);
      }
    }
  }

  void deRegisterProcessors(List<Class<? extends HttpProcessor>> processorClasses)
  {
    for (Class<? extends HttpProcessor> registeredProcessor : processorClasses) {
      try {
        CustomHttpProcessorsRepository.INSTANCE.deRegisterProcessor
             (registeredProcessor.getName());
      } catch (Exception ex) {
        ex.printStackTrace();
        //keep going
      }
    }
  }
}
```

Note: At this stage they are not associated with any API and will not be in the execution path until this association is made.

To enable a `HttpProcessor` for a given API you must PUT the processors plus configuration to that API using the Portal REST interface:

Example 6–6 Processor Configuration REST PUT Payload

```
{
 "SubscribeProcessorsRequest":{
   "processors":[
     {
      "name":"JsonTokenProcessor",
      "configuration":"{\"key\":\"1y7M3cO(86m0R5FwC>(N61lRwiTQYu\",\"header\":
        \"x-json-token\"}"
    }
   ]
 }
}
```

The configuration above maps to the configuration class `JsonTokenProcessorConfig`, shown in Example 6–7:

Example 6–7 JsonTokenProcessorConfig

```
public class JsonTokenProcessorConfig implements HttpProcessorConfig {
  private String key;
  private String header;

  public String getKey() {
    return key;
  }

  public void setKey(String key) {
    this.key = key;
  }

  public String getHeader() {
    return header;
  }

  public void setHeader(String header) {
    this.header = header;
  }
}
```

The next request to the `ECHOServer` API invokes `JsonTokenprocessor` which looks like Example 6–8:

Example 6–8 JsonTokenProcessor

```
@HttpProcessorAnnotation(name = "JsonTokenProcessor", configBean =
    JsonTokenProcessorConfig.class)
  public class JsonTokenProcessor extends
     AbstractHttpProcessor<JsonTokenProcessorConfig> {
    private JsonTokenProcessorConfig configuration;

    @Override
```

```
    public Set<Principal> process(HttpServletRequest httpServletRequest) throws
          HttpProcessorExecutionException {
      String clearTextToken = getTokenFromRequest(httpServletRequest);
      JsonReader reader = Json.createReader(new StringReader(clearTextToken));
      JsonObject token = reader.readObject();
      Set<Principal> principals =
            assertApplication(token.getString("application"));
      if (!principals.isEmpty()) {
        //We have found an application, let's add scope to application principle
        for (Principal principal: principals) {
          if (principal instanceof ApplicationInstance) {
            principals.remove(principal);
            principal = new JsonTokenApplication((ApplicationInstance)principal,
                token.getString("scope"));
            principals.add(principal);
            break;
          }
        }
      }
      return principals;
    }
    private String getTokenFromRequest (HttpServletRequest httpServletRequest) {
      return AES.getInstance(getConfiguration().getKey()).
        decrypt(httpServletRequest.getHeader(getConfiguration().getHeader()));
    }
    @Override
    public JsonTokenProcessorConfig getConfiguration() {
      return configuration;
    }

    @Override
    public void init(JsonTokenProcessorConfig configuration) throws
        HttpProcessorConfigException {
      this.configuration = configuration;
    }
}
```

Optional Custom HTTP Processor Configuration

The framework provides an optional interface for the configuration of the processors. The `HttpProcessorConfig` interface is used to mark that its subclass is the configuration type of the processors. Example 6–9 illustrates:

Example 6–9 HttpProcessorConfig

```
/* Copyright (c) 2017, Oracle and/or its affiliates. All rights reserved. */
package oracle.ocsg.security.provider;

import java.io.Serializable;

/**
 * Interface of a HTTP processor configuration, its implementation must be a
 * mean type.
 */
public interface HttpProcessorConfig extends Serializable {
}
```

A custom HTTP processor configuration class must meet the following criteria:

1. The implementation must implement `HttpProcessorConfig`.

2. The configuration for the processor is optional.
3. The configuration class must invoke the constructor without parameters.
4. Plain Old Java Object (POJO) style is strongly recommended.
5. It can be exchanged with JSON strings.

Example 6–10 shows a sample custom processor configuration.

Example 6–10 Example Custom Processor Configuration

```
package oracle.ocsg.security;

import oracle.ocsg.security.provider.HttpProcessorConfig;

/**
 * Sample HTTP processor configuration.
 */
public class SampleHttpProcessorConfig implements HttpProcessorConfig {
  /**
   * generated serial version UID.
   */
  private static final long serialVersionUID = 8390799967874999370L;
  private long intervals;
  private int maxTryTimes;
  /**
   * Constructor without parameters MUST be there.
   */
  public SampleHttpProcessorConfig() {
  }
  public SampleHttpProcessorConfig(final long intervals, final int maxTryTimes) {
    this.intervals = intervals;
    this.maxTryTimes = maxTryTimes;
  }
  public long getIntervals() {
    return intervals;
  }
  public void setIntervals(long intervals) {
    this.intervals = intervals;
  }
  public int getMaxTryTimes() {
    return maxTryTimes;
  }
  public void setMaxTryTimes(int maxTryTimes) {
    this.maxTryTimes = maxTryTimes;
  }
}
```

Custom HTTP Processor MBean

The custom HTTP processor MBean is `oracle.ocsg.security.provider.processor.management.ProcessorConfigurationMBean`. Example 6–11 shows the MBean operations that are exposed:

Example 6–11 Exposed MBean Operations

```
/**
 * List all the HTTP processor subscriptions for all the APIs.
 * <p><strong>Scope</strong>: Cluster</p>
 *
```

```
     * @return The HTTP processor subscriptions on all the APIs.
     * @throws ManagementException When it fails to get the subscriptions.
     */
    Map<String, String> listProcessorSubscriptions() throws ManagementException;

    /**
     * Gets the JSON string of the HTTP processor subscription for the specified API
    context path.
     * <p><strong>Scope</strong>: Cluster</p>
     *
     * @param contextPath The API context path.
     * @return The JSON content of the API JSON subscription.
     * @throws ManagementException When it fails to get the subscription.
     */
    String getProcessorSubscription(String contextPath) throws ManagementException;
```

7

Managing Partner Applications

This chapter described how you can configure and manage partner applications by using Oracle Communications Services Gatekeeper API management platform and its partner relationship management (PRM) portal applications.

About Applications

Partners create their applications by using Partner Portal. Partners can subscribe APIs to their applications when they create them or later. Partner managers supply the APIs for partner applications.

The following topics explain how to manage partner applications:

- Life Cycle of an Application
- Application States and Notification Entries
- Data Integrity During Updates to Applications

Life Cycle of an Application

An application goes through the following stages:

1. A partner creates an application and submits it in Partner Portal. The application state is set to CREATE PENDING APPROVAL.
2. As the partner manager, you review the application in Partner and API Management Portal and do one of the following:
   - Approve the application.

     The application state is set to ACTIVE. The partner sees the approval on the **Messages** page of his Partner Portal.
   - Reject the application.

     The application is returned to the partner. The partner sees the rejection on the **Messages** page of his Partner Portal.
3. When the application is active, the application is updated in one or both of the following ways:
   - The partner edits the application and submits it in Partner Portal. The application state is set to UPDATE PENDING APPROVAL.
   - As a partner manager you update the API.

4. As a partner manager, you approve or reject the updates made by the partner to the application. If the automatic approval of applications is enabled, Services Gatekeeper approves or rejects the updated application.
5. When a partner decides to delete an application, the partner submits a request in Partner Portal.

   As the partner manager, you review the application in Partner and API Management Portal and do one of the following:

   - Approve the deletion. The application is deleted from Partner Portal.
   - Reject the deletion. The application continues to display in an active state in Partner Portal.

Application States and Notification Entries

Services Gatekeeper uses notifications to alert the users of the PRM portals of events associated with application-related requests and responses.

When a partner registers or updates an application in Partner Portal, the partner manager receives a corresponding notification in Partner and API Management Portal. The partner waits to receive the results of the review before attempting further updates on the application. When the partner manager reviews a notification and approves or rejects the request, the partner receives a notification in Partner Portal. The partner is now able to take further action on the application. When a partner deletes an application, the application is removed from Partner Portal, unless the partner manager rejected the deletion.

All notifications for a partner manager are displayed on the **WORKFLOW** page of the Partner and API Management Portal. All notifications for a partner are displayed on the **MESSAGES** page of the Partner Portal.

When a network service supplier applies for a network service supplier account, the partner manager receives the registration request in Partner and API Management Portal. When the network service supplier signs in to Network Service Supplier Portal and registers, updates or deletes a network service interface, partner manager receives a corresponding notification in Partner and API Management Portal. However, the partner manager reviews the notification and is not required to approve or delete the notification.

Data Integrity During Updates to Applications

Services Gatekeeper maintains data integrity by disallowing partner actions if the partner manager is currently changing the application.

At times, a partner may log in to Partner Portal and access the application submitted for approval at the same time as when the partner manager is reviewing the application request in Partner and API Management Portal.

In order to maintain data integrity of applications, Services Gatekeeper takes the following precautions. For:

- Newly-created applications

  When a partner creates an application and submits it, Services Gatekeeper displays the approval request for the application from Partner Portal with a CREATE PENDING APPROVAL notification in Partner and API Management Portal.

  - The partner cannot update that newly-created application the period when it is under review by the partner manager. Services Gatekeeper locks the application data.
  - The partner cannot submit another request for approval by the partner manager.
  - The partner can delete the newly-created application during the period when the partner manager is reviewing it.

    When the partner manager completes his review of that newly-created application, Services Gatekeeper deletes the partner manager's approval or rejection of that application.

    The deleted application is no longer available in Partner Portal or Partner and API Management Portal.

- Updating active applications

  If the partner manager is reviewing an active application at the current time:

  - The partner cannot update the application during the period when it is under review by the partner manager. Services Gatekeeper locks the application data.
  - The partner cannot submit another request for approval by the partner manager.
  - The partner can delete the application during the period when the partner manager is reviewing it.

    When the partner manager completes his review of that updates, Services Gatekeeper deletes the partner manager's approval or rejection of the updates to that application.

    The deleted application is no longer available in Partner Portal or Partner and API Management Portal.

- Deleting active applications

  When a partner accesses an active application and deletes it, Services Gatekeeper removes the application from Partner Portal. It displays the deletion request for the application from Partner Portal with a DELETE PENDING APPROVAL notification in Partner and API Management Portal.

  Partners cannot delete an active application when the application is in Partner and API Management Portal with an DELETE PENDING APPROVAL notification.

Collecting Information About Application Traffic with EDRs

See "Managing EDRs, CDRs, and Alarms" in *Services Gatekeeper System Administrator's Guide* for more information collecting information about applications, and action chain processing in particular using EDRs.

8

Managing Partners and Partner Groups

This chapter describes how you can create and manage partner accounts, network service supplier accounts, and partner groups in Oracle Communications Services Gatekeeper application programming interface (API) management platform.

Overview of Accounts and Roles

Services Gatekeeper supports and manages accounts for partners, network service suppliers, partner applications, partner groups, and partner managers. Services Gatekeeper oversees all of these accounts and their activities and stores all the data in its database.

This is how the roles operate:

- Partner Manager

  You create partner manager accounts in Services Gatekeeper. Each partner manager account owner manages a set of partners and partner groups. Partner managers create, approve, manage and delete partner accounts in Partner and API Management Portal.

- Partner

  A partner account is created when a self-registration request is approved by a partner manager or by Services Gatekeeper (if automatic registration approval is enabled). Alternately, that account could be created by a partner manager using Partner and API Management Portal.

  Partners are assigned to partner groups by the associated partner manager.

  Partners create applications and manage them under the supervision of the partner manager.

- Network Service Supplier

  A network service supplier account is created when a self-registration request is approved by a partner manager. Alternately, that account could be created by a partner manager using Partner and API Management Portal. Partner managers approve, manage and delete network service supplier accounts in Partner and API Management Portal.

  Network service suppliers are not assigned to any group.

  Network service suppliers create interfaces and manage them in Network Service Supplier Portal.

The service accounts operate in the following way:

- Partner Applications

Partner applications are created by partner, assigned to partner groups by the associated partner managers who manage all changes to the applications.

Services Gatekeeper assigns a Traffic User account and password for each partner application. The partner who created the application can change the password used by that Traffic User account.

Every application belongs to the specific partner group to which the partner account belongs.

- Partner Groups

  Partner groups are created and managed by partner managers in Partner and API Management Portal. Services Gatekeeper assigns a specific service level agreement to each partner group.

  Services Gatekeeper maintains a default partner group called **sysdefault_sp_grp**. The default partner group contains a blank service level agreement.

  Services Gatekeeper assigns a newly-created partner account to **sysdefault_sp_grp**. The partner manager must assign a partner to a different partner group before the partner can create an application.

About the Registration Review

Service Gatekeeper requires valid account user name and passwords before it allows access to Network Service Supplier and Partner Portals. You can apply for an account on the login page of the Network Service Supplier and Partner Portals.

When you complete the registration form Services Gatekeeper stores that registration request temporarily until the request is approved. It displays each registration request it receives as a **Partner registration request** or **Network Supplier registration request** task in Partner and API Management Portal. The registration request must be approved in Partner and API Management Portal before the owner of the account can access the respective portal.

After partner managers review of a registration request, Services Gatekeeper sends an email notification to the email address provided in the registration request.

- If the registration request is approved, the email notification sent to the partner or network service supplier states that the registration request has been approved. The email recipient can access the respective portal.
- If the registration request is denied, the email notification states that the registration request is denied.

  The email recipient must resubmit the partner or network service supplier registration request with the correct entries. At this point, the partner or network service supplier may contact you to ascertain the reasons.

You can automate the registration process by customizing the system configuration for the Partner and API Management Portal.

Managing Accounts

Whether the individual has a partner or network service supplier account, the account data consists of the following information on the owner of the account:

- General information, such as the user name, password, first and last name of the account owner, the email address associated with the owner, and a telephone number.

- Company data, such as the company name, its URL, street address, city, state or province name, and the country name.
- Primary and secondary contact information, such as the first and last name, the email address, a telephone number, and the time when the contact person is available.

Setting Up Accounts in Partner and API Management Portal

When individuals enter requests to become partners or network service suppliers, Service Gatekeeper displays the requests as notifications in your workflow table. Review each request. If:

- You approved the request, Services Gatekeeper sends an email notification to the email address on the registration request. It includes the account information with your list of partners and network service supplier accounts it displays on your Partner and API Management Portal.
- You denied the request, Services Gatekeeper takes no action. The person who submitted the registration request contacts you offline to resolve any issue with the registration entries.

Creating Partner Accounts in Partner and API Management Portal

You can actively create partners accounts by collecting the required information from your partner and creating the account in Partner and API Management Portal. Services Gatekeeper sends an email notification to the email address on the registration request. It includes the account information with your list of partners and network service supplier accounts it displays on your Partner and API Management Portal.

Managing Accounts

Accounts have the following status in Services Gatekeeper:

- **registered**, indicating that the account needs approval by the partner manager. The individual whose account is in the **registered** state cannot log in to the associated portal application.
- **active**, indicating that the account has been approved by the partner manager. A partner or network service supplier with an **active** account can log in to the associated portal and perform his tasks.

You can access a partner or network service supplier account and view the account details, reset the password (if the account is active), or delete the account.

You can access a partner account and assign the partner to a different partner group (if the account is active).

Managing Partner Groups

Services Gatekeeper provides a default partner group called **sysdefault_sp_grp**. By default, when a partner account is activated (that is, the status is active in Services Gatekeeper, the account is assigned to **sysdefault_sp_grp**, the default partner group.

You can create as many partner groups as you require in Partner and API Management Portal. Partner group names are not case-sensitive and they must be unique. When you create a partner group, enter a unique name, the maximum number of requests per second allotted to the partner group, and the number of requests allowed within the specified number of days for the partner group.

When the account status is **active**, you can reassign a partner account to a different partner group.

You can update a partner group by adding or removing partners from the group. Additionally, the partner group is modified if you alter the API data associated with the partner group.

Group Assignments for Partners and SLAs

When you assign a partner to a different partner group, the service level agreement associated with the current partner group becomes invalid. APIs that are not compatible or covered by the SLA associated with the destination partner group are suspended. Partner and API Management Portal displays a dialog box informing you of all the applications that are affected when the partner account moves to the destination group when the change in partner group assignment could affect the following elements in the service level agreement:

- Rate
- Quota
- Guarantee
- Expiration Date

For example, a service provider (partner) currently has an application A that subscribes to an API called "SendSMS" and requires 100 throughput per second (TPS) for that API. However, the new partner group's SLA supports 80 TPS for the API.

The dialog box displays three options from which the partner manage can make the following adjustment:

- Expand the SLA to accommodate the requirements of the affected application. In our example case, the SLA for the new group is expanded to support the required 100 throughput per second.

  Such an adjustment might result in adding an SLA requirement currently missing in the new partner group's SLA.

- Modify the parameters in the current application to fit the new partner group's SLA. In our example case, modify the application to support 80 TPS only.

  Such an adjustment might result in deleting the SLA requirement currently missing from the new partner group's SLA.

- Cancel the support for the SendSMS API (Cancel the assignment? Partner belongs to the old group?)

Deleting Partner Groups

You can delete a partner group if it has no members and there are no applications associated with the partner group.

9

Administering the PRM Portals

This chapter describes how to manage the Oracle Communications Services Gatekeeper Partner Management Portal, Partner Portal, and Network Service Supplier Portal.

Resetting Passwords

Partner managers can reset passwords for an accounts.

To reset a password:

1. The account owner (partner or network services supplier) requests for the password to be reset. See "Requesting for a Network Service Supplier or Partner Password to be Reset".
2. The partner manager receives a notification requesting the password to be reset for the account. The partner manager resets the password.
3. The partner or receives an email containing the link to the URL where he can reset his password.

Note: The URL:

- Expires after a specified interval, such as 6 hours or one day.
- Remains active until it is accessed. When the person resetting the password accesses the URL, it is disabled and no longer accessible.
- After the user resets the password successfully, no further emails are sent to the user.

Requesting for a Network Service Supplier or Partner Password to be Reset

Network Service suppliers and partners can request that their account passwords be reset. If you are a network service supplier or partner, do the following:

1. Access the login page for the respective (Network Service Supplier or Partner) portal.
2. Click the **Forgot Password** link on the main login page for the portal.
3. Provide the required information.
4. Submit the request.

Your request for your password to be reset is displayed in Partner and API Management Portal.

Resetting Passwords in Partner and API Management Portal

To reset account passwords for a network service supplier or partner account, you do the following in Partner and API Management Portal:

1. Sign in to Partner and API Management Portal.
2. Access the **Partner & Network Supplier List** page, by clicking **Partners**.
3. In the table, locate the user name entry for the partner or network service supplier.
4. Right-click the row and select **Reset Password**.
5. In the **Reset Password Confirmation** dialog box, check the name.
6. Click **OK**.

An email notification is sent to the email address of the account holder. The email contains the link to the Web page where the password can be reset by the network service supplier or partner.

Resetting Passwords in Network Service Supplier or Partner Portal

Network service suppliers and partners who have requested a password reset, receive an email notification when the partner manager approves their request. This email notification contains the link to the page where you can reset the password for your account.

Note: The URL is valid for one day.

1. Open the email notification and click the URL link in the notification.

   The password reset page for your portal is displayed.
2. In the **Security Answer** field, input the answer you provided for the **Security Question** you selected at the time you registered for a partner or network service supplier account.
3. In the **New Password**' and **Confirm Password** fields, input your new password.

The password is reset.

About Customizing PRM Portals

You can customize the pages of the three portals in many ways. For example, you can do one or more of the following:

Caution: The online help that Services Gatekeeper supplies with your PRM Portals provides support for the default configuration only.

Oracle recommends the following:

- If you add a page to a module, ensure that you have provided online help support appropriate for that custom page.
- If you add a page to a module, ensure that you have provided online help support appropriate for the custom module (and its pages).

- Add new pages to a portal and provide a new navigation entry point on the left or top menu to enter these new pages.
- Add other options to current pages in a portal. For example, you can create a new blacklist action and have it appear in the **Actions** tab of the **APIs** pages in Partner and API Management Portal. Or add a new source for API exposure to expose any data as soap or rest API.
- Add new functionality to existing pages, such as adding revenue sharing settings for applications, add logic to provide a rolling average of the API usage invocation in the Dashboard of Partner and API Management Portal and so on.
- Remove existing functionality such as the need for registration confirmation in Partner Portal, or the Groovy action, or remove all sources for API creation except the one coming from the Network Service Supplier Portal source in Partner and API Management Portal.
- Modify existing functionality by changing the workflow for a certain task in a portal, such as creating a network service interface in Network Service Supplier Portal.
- Add new pages to the existing portal and provide a new navigation entry point on the left or top menu to enter these new pages.

For information on how to add pages to a module and how to add a module, see "Extending Portals" in *Services Gatekeeper Portal Developer's Guide*.