Oracle® Communications Services Gatekeeper

Accounts and SLAs Guide

Release 7.0

E95427-01

July 2018

ORACLE®

Oracle Communications Services Gatekeeper Accounts and SLAs Guide, Release 7.0

E95427-01

Contents

7 Managing Sessions

8 Defining Service Provider Group and Application Group SLAs

Preface

This document describes service provider and application provisioning for Oracle Communications Services Gatekeeper.

It covers:

- An overview of the administration model
- Managing service provider groups
- Managing service provider accounts
- Managing application groups
- Managing application accounts
- Managing application instances

Audience

The book is intended primarily for telecom operators who perform service provider and application provisioning tasks.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

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

- *Oracle Communications Services Gatekeeper Alarms Handling Guide*
- *Oracle Communications Services Gatekeeper Application Developer's Guide*
- *Oracle Communications Services Gatekeeper Communication Service Reference Guide*

- *Oracle Communications Services Gatekeeper Concepts*

1

About Service Level Agreements and Accounts

This chapter describes the Service Level Agreements (SLAs) and accounts that Oracle Communications Services Gatekeeper uses to define and enforce access to Services Gatekeeper and to the underlying telecommunication network.

About SLAs

An SLA is an XML document that defines and enforces access to Services Gatekeeper or, in the case of node SLAs, to the underlying network nodes. The core element in an SLA is a contract element whose child elements specify the details of the access. Examples of these details are the dates in which access is permitted, the number of requests to be processed within a certain time period, and the denial of the use of certain methods or parameter values to an application.

SLAs are associated with application service providers through a tiered system of accounts that are organized into account groups.

If you are using the Services Gatekeeper API management features through the Partner Manager and API Management Portal, Partner Portal, and Network Service Supplier Portals, most of the required SLA creation and management is transparent to you. See "Using Partner and API Management Portal to Manage SLAs and Accounts" for more information. If you are using the Service Gatekeeper features, then you need to follow the instructions in this book to create and manage SLAs.

You can load an SLA into a Services Gatekeeper implementation from the Administration Console console using the operations provided in the **ApplicationSLAs** section of the **AccountService** container service. See "Managing SLAs" for information about the **AccountService** operations to load and manage SLAs. These SLA load operations offer options for loading SLAs to all geo-redundant locations if your implementation uses this feature.

As an alternative, you can manage SLAs through external management systems integrated with Services Gatekeeper using the Services Gatekeeper Partner Relationship Management interfaces. For information about the Partner Relationship Management interfaces, see “Partner Relationship Management Interfaces” in *Services Gatekeeper Portal Developer’s Guide*. You can also manage SLAs programmatically using MBeans. For information about the **ServicelevelAgreement** MBean, see "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

System SLAs

System SLAs have static XML Schema Documents (XSDs). The enforcement logic is already in place for these types of SLAs, except for the subscriber SLA, which requires custom subscriber policy logic. The following types of SLAs are system SLAs:

- Global node SLA
- Service provider group SLA
- Application group SLA
- Service provider node SLA
- Subscriber SLA

Service provider group SLAs and application group SLAs define service provider and application access to Services Gatekeeper. See "Defining Service Provider Group and Application Group SLAs" for information about these types of SLAs and "Sample SLA XML File" for a complete example.

Node SLAs define access to the underlying network, either by service provider group or for Services Gatekeeper as a whole regardless of the service provider whose requests are being processed. See "Defining Global Node and Service Provider Group Node SLAs" for information about node SLAs.

Subscriber SLAs define subscriber policy and manage subscriber access. The implementor defines the subscriber policy logic using the Platform Development Studio. For more information, see “Using Policies to Manage Subscribers” in *Services Gatekeeper Extension Developer’s Guide*.

Custom SLAs

Custom SLAs are enforced by customized logic for which the XSDs are developed using the Platform Development Studio. For information on custom SLAs and an example of one, see “Custom Service Level Agreements” in *Services Gatekeeper Extension Developer’s Guide*.

The following SLAs are custom SLAs:

- Custom global SLAs
- Custom service provider group SLAs
- Custom application group SLAs

The custom logic that operates on data is provided in the SLAs and on data that comes with the request. The data in the SLA is formatted according to the SLA XSD, and the enforcement logic uses the Document Object Model (DOM) representation of the XSD to get the data in the SLA. The request can provide information such as application ID, application instance, service provider ID, application group ID, and service provider group ID.

Custom SLAs can be associated with application groups and service provider groups or be global. A custom global SLA is valid for all requests, regardless of which application group or service provider group the application that triggered the request.

About Accounts

Each service provider and application that uses the Services Gatekeeper application-facing interfaces must establish an account.

Service provider accounts, application accounts, and application instances have states that can be changed using the Administration console. This enables the administrator to take them out of service temporarily.

Accounts

There are two types of accounts:

- Service Provider Account: Every service provider that has access to the operator's Services Gatekeeper facilities has a service provider account.
- Application Account: Every application that has access to the operator's Services Gatekeeper facilities has an application account. Each application account is associated with a service provider account.

See "Managing Service Provider and Application Accounts" for information about creating and managing accounts.

Account Groups

For the purposes of SLA enforcement, accounts are organized into account groups. There are two types of account groups:

- Service provider group
- Application group

When you load an Application Group SLA or Service Provider Group SLA, you provide the identifier of the group with which the SLA is associated along with the SLA.

Following is an example of how a Services Gatekeeper administrator might use system SLAs to create three different service provider groups:

- Bronze
- Silver
- Gold

Each of these service provider groups would be associated with different SLAs with different privileges regarding maximum throughput and Quality of Service (QoS parameters. When a new service provider is provisioned, the service provider is associated with the appropriate group based on, for example, the amount of network traffic the service provider is projected to generate. If the outcome is not what was predicted, the service provider account can be reassigned to another service provider group that has a different QoS parameter defined in its service provider SLA.

See "Managing Groups" for information about creating and managing groups.

Application Instances

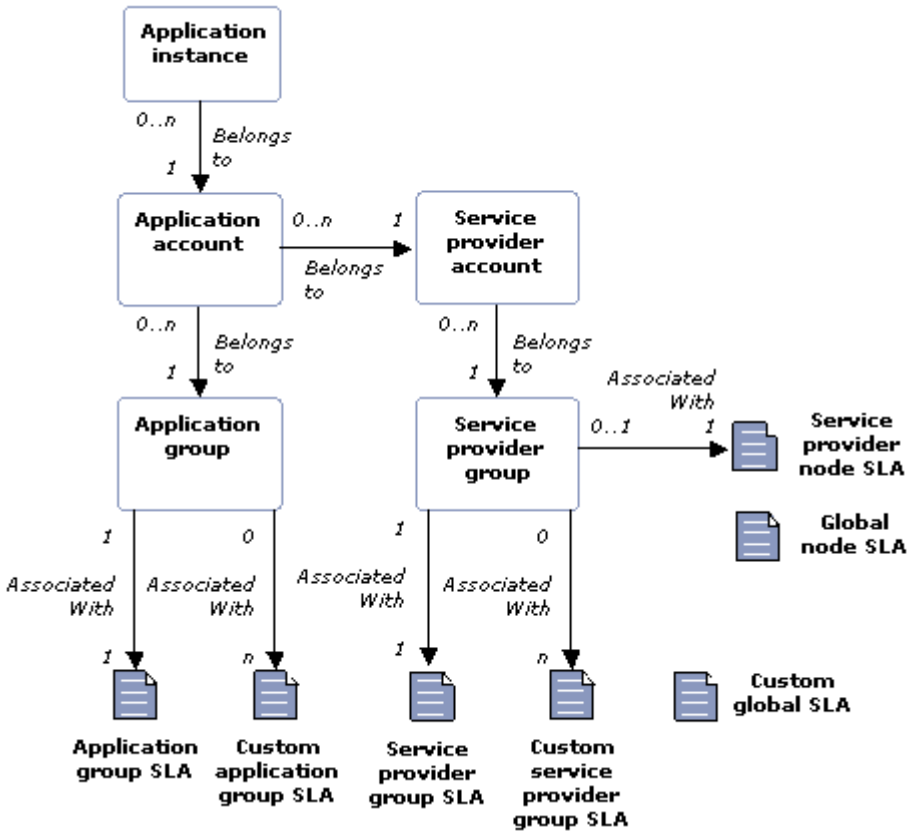
In the context of SLAs in Services Gatekeeper, an application instance is an application service provider. Each application instance has a user name and password or certificate. Each application instance is associated with a service provider account / application account combination. An application instance must be in the ACTIVATED state to send traffic through Services Gatekeeper.

See "Managing Application Instances" for more information about creating and managing application instances.

Associations Among SLA Types, Account Types, and Group Types

Figure 1–1 illustrates the associations among the different account types, group types, and SLA types.

Figure 1–1 Service provider and application model with SLAs



Note the following relationships:

- An *application instance*, which uniquely identifies an account, belongs to an *application account* and a *service provider account*.
- An *application account* belongs to a *service provider account* and an *application group*.
- A *service provider account* belongs to a *service provider group*. A service provider account can have zero or more application accounts associated with it.
- An *application group* is associated with an *application group SLA* and zero or more *custom application group SLAs*.
- *A service provider group* is associated with a service level agreement on the service provider-level and may also be associated with a *service provider node SLA* and zero or more *custom service provider group SLAs*.
- Zero or more custom global SLAs can be defined,. These are not associated with any special group or account, but are valid for all accounts and groups
- A *global node SLA* is not associated with any special group or account, but is valid for all accounts and groups.

In light of the relationships between applications, groups, and SLAs, you configure them in the following order:

1. SLAs
2. Service provider groups
3. Service provider accounts
4. Application groups
5. Application accounts

The process steps are listed in the section, "Typical SLA and Account Workflow".

Note that the IDs of service provider groups, application groups, application instances, and service provider accounts must be unique. The combination of service provider account and application account for an application instance must be unique.

The separation of accounts and groups provides a flexible way of defining service level agreements. Groups allow the Services Gatekeeper administrator to offer tiers of service at both the service provider and application level.

Application group SLAs and service provider group SLAs can be managed and enforced either locally in one site or across geo-redundant sites.

Typical SLA and Account Workflow

In a typical SLA and account workflow, you complete the following steps in that order:

1. Create service provider and application groups that will correspond to the levels of service to be defined in the SLAs. See "Managing Groups" for more information.
2. Define the set of service classes, expressed as SLAs, at both the service provider group level and the application group level. See "Defining Service Provider Group and Application Group SLAs" for more information.
3. Define the set of node service classes, expressed as SLAs, at both the service provider node level and global node level. See "Defining Global Node and Service Provider Group Node SLAs" for more information.
4. Associate the service provider group SLAs and application group SLAs with the groups. See "Managing SLAs".
5. Load the global node SLA and service provider group node SLAs. See "Managing SLAs" for more information.
6. Create a service provider account and associate it with the appropriate service provider group. See "Managing Service Provider and Application Accounts" for more information.
7. Create an application account and associate it with the appropriate application group. See "Managing Service Provider and Application Accounts" for more information.
8. Create an application instance, see "Managing Application Instances" for more information.
9. Depending on which communication service is being used, provision communication service-specific data using the operation and management for relevant communication service.
10. Distribute the credentials to be used by the application to the service provider.

Using Partner and API Management Portal to Manage SLAs and Accounts

Partner and API Management Portal, along with a companion GUI, Partner Portal act as "front-office" offerings provide a friendlier workflow, insulating you from the intricacies of Services Gatekeeper administration. Use the menu selections and/or input fields on well-defined pages of the two GUIs to set up your SLAs and manage your service provider accounts.

Network operators and service provider managers use Partner and API Management Portal to manage:

- Global node SLAs
- Service provider group SLAs
- Service provider group node SLAs
- Service Provider accounts
- Application accounts

The **Actions** tab available to each API on the Partner and API Management Portal also provides these SLA capabilities:

- The ability to add access limitations for specific API endpoint
- The ability to authenticate API traffic using a an API key inside message headers

Network operators authorize service providers to act as partners with access to Partner Portal and create applications using these SLAs. For more information on how to use these GUIs, see *Services Gatekeeper Partner and API Management Portal Online Help* and *Services Gatekeeper Partner Portal Online Help*.

2

Managing Application Instances

This chapter describes how application instances are managed and provisioned in Oracle Communications Services Gatekeeper:

- Summary of Tasks Related to Application Instances
- About Application Instance States
- Web Services Security

You also need to configure application password encryption. For details, see "Encrypting Application Passwords" in *Services Gatekeeper System Administrator's Guide*.

See **ApplicationInstanceMBean** in the "All Classes" section of the *Services Gatekeeper OAM Java API Reference* or information for details on this MBean.

Before registering application instances, service provider and application accounts must have been created. See "Managing Service Provider and Application Accounts" for more information.

Summary of Tasks Related to Application Instances

Table 2–1 lists the tasks related to application instances and the operations you use to perform those tasks. The operations belong to **ApplicationInstanceMBean**. For detailed information on this MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*..

Table 2–1 Tasks Related to ApplicationInstanceMBean

Task	ApplicationInstanceMBean Operation
Get information about the number of application instances	countApplicationInstances
Add, remove and get information about an application instance	addApplicationInstance removeApplicationInstance getApplicationInstance
List registered application instances	listApplicationInstances
Define additional properties for an application instance	setApplicationInstancePassword setApplicationInstanceProperties setApplicationInstanceReference setApplicationInstanceState

About Application Instance States

Application instances have states. Two states are possible:

- **ACTIVATED**
- **DEACTIVATED**

In **ACTIVATED** state, the application using this application instance ID is in normal operation.

In **DEACTIVATED** state, the application using this application instance ID is not allowed to send traffic through Services Gatekeeper. The account is still valid and the state can be transitional by the Services Gatekeeper Administrator using the setApplicationInstanceState operation of the **ApplicationInstanceMBean**.

Web Services Security

When Web Services Security is being used as authentication for the application instance, its credentials must be set up, as described in Table 2–2.

Table 2–2 Credential Mapping

ApplicationInstance	UserNameToken	X.509	SAML
ApplicationInstanceName	userName	CN	CN
Password	password	Not applicable	Not applicable

In addition, if X.509 or SAML are being used, the certificates and private keys must be provisioned in the WebLogic Server keystore. For example, **ImportPrivateKey** utility can be used to load the key and digital certificate files into the keystore. See "ImportPrivateKey" in *Oracle Fusion Middleware Command Reference for Oracle WebLogic Server* for a description of **ImportPrivateKey**.

3

Managing SLAs

This chapter describes how the different types of Service Level Agreements (SLAs) are managed and provisioned in Oracle Communications Services Gatekeeper (OCSG).

It contains the following sections:

- Introduction to SLA types
- Summary of Tasks Related to SLAs

Introduction to SLA types

There are two different kinds of SLAs:

- System level SLAs
- Custom SLAs

System level SLAs have static XSDs that are already defined in Services Gatekeeper, while custom SLAs provides the possibility to use a custom XSD. Any given service provider group or application group can have only one system SLA associated with it, while they can have many custom SLAs. Create custom SLAs when additional SLA enforcement logic is required that is not provided by default in Services Gatekeeper. You must create the enforcement logic for a Custom SLA. See *Services Gatekeeper Extension Developer's Guide* for information on how to develop Custom SLAs.

When SLAs are loaded into memory, they are stored in the SLA repository. SLAs can be loaded into the repository in two ways:

- The contents of the SLA is provided as a parameter in the operation that loads the SLA.
- The SLA is stored in a file and the URL to that file is provided as a parameter in the operation that loads the SLA.

SLAs are retrieved from the SLA repository using the retrieve operations.

The SLA loaded into the SLA repository is the one being enforced. Changes to the file after the SLA is loaded into the repository are not automatically reflected in the active version.

Table 3–1 outlines the different SLA types, the IDs and their scope.

Table 3–1 SLA Types

SLA Type	ID(s)	Description
Application Group	application system:geo_application	System SLA. Defines how application can use Services Gatekeeper. See "Application Group SLAs" for a summary of related management operations. For information on creating these SLAs, see "Defining Service Provider Group and Application Group SLAs". The SLA type ID: ■ **application** indicates that the SLA is loaded to and enforced in the network tier cluster it is loaded. ■ **system:geo_application** indicates that the SLA is loaded to and enforced across all network tier clusters in a geo-redundant configuration.
Service Provider Group	service_provider system:geo_service_provider	System SLA. Defines how service providers can use Services Gatekeeper. See "Service Provider Group Node SLAs" for a summary of related management operations. For information on creating these SLAs, see "Defining Service Provider Group and Application Group SLAs". The SLA type ID: ■ **service_provider** indicates that the SLA is loaded to and enforced in the network tier cluster it is loaded. ■ **system:geo_service_provider** indicates that the SLA is loaded to and enforced across all network tier clusters in a geo-redundant configuration.
Global Node	global_node	System SLA. Defines how Services Gatekeeper is allowed to use the underlying telecom network nodes. See "Global Node SLAs" for a summary of related management operations. For information on creating these SLAs, see "Defining Service Provider Group and Application Group SLAs". This SLA type is loaded and enforced locally in the network tier cluster it is loaded.

Table 3–1 (Cont.) SLA Types

SLA Type	ID(s)	Description
Service Provider Node	service_provider_node	System SLA. Defines how Service Providers are allowed to use the underlying telecom network nodes. See "Service Provider Group Node SLAs". For information on creating these SLAs, see "Defining Service Provider Group and Application Group SLAs". This SLA type is loaded and enforced locally in the network tier cluster it is loaded.
Subscriber	subscr	System SLA. Defines classes of application services that can be associated with subscribers in the context of Services Gatekeeper. Using the Platform Development Studio, an operator or integrator may create a subscriber-centric policy mechanism. The specifics of this mechanism are covered in "Using SLA Policies to Manager Subscribers" in *Services Gatekeeper Extension Developer's Guide*. This SLA type is loaded and enforced locally in the network tier cluster on which it is loaded.
Custom	Defined at load time	Custom SLA. A custom SLA is defined by an XSD, that must be created and loaded. A custom SLA type ID is associated with the XSD, and this type is the one being referenced when loading the custom SLAs. In addition to the SLA, the enforcement logic that operates on the data in the SLA must be created. See "Creating Custom Runtime SLAs" in Services Gatekeeper *Services Gatekeeper Extension Developer's Guide*. Just like system SLAs, the custom SLAs are associated with service provider groups and application groups. In addition there is a custom global SLA, that does not take into consideration the originator of the request, but affects all requests. Custom SLAs are loaded and enforced locally in the network tier cluster it is loaded.

Note: The prefix system is reserved, and should not be used by custom SLAs. For backward compatibility, there is a set of SLA types without this prefix.

Summary of Tasks Related to SLAs

The tasks related to the management of SLAs comprise the following:

- Managing Service Provider and Application Group System SLAs
- Managing Node SLAs

Managing Service Provider and Application Group System SLAs

The management of Service Provider and Application Group system SLAs comprises managing the following:

- Service Provider Group System SLAs
- Application Group SLAs

You use the methods of the **ServiceLevelAgreementMBean** MBean to manage Service Provider and Application Group system SLAs. For information on the methods and

fields of the **ServiceLevelAgreementMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Service Provider Group System SLAs

Table 3–2 describes the tasks related to managing Service Provider Group system SLAs and the **ServiceLevelAgreementMBean** MBean methods to use.

Table 3–2 Tasks Related to Managing Service Provider Group System SLAs

Task	ServiceLevelAgreementMBean Method to Use
Associate Service Provider Group with SLA	**loadServiceProviderGroupSlaByType** **loadServiceProviderGroupSlaFromUrlByType**
View Service Provider Group SLA	**retrieveServiceProviderGroupSlaByType**

Application Group SLAs

Table 3–3 describes the tasks related to managing Application Group system SLAs and the **ServiceLevelAgreementMBean** MBean methods to use.

Table 3–3 Tasks Related to Managing Application Group SLAs

Task	ServiceLevelAgreementMBean Method to Use
Associate Application Group with SLA	**loadApplicationGroupSlaByType** **loadApplicationGroupSlaFromUrlByType**
View Application Group SLA	**retrieveApplicationGroupSlaByType**

Managing Node SLAs

The management of Node SLAs comprises managing the following:

- Global Node SLAs
- Service Provider Group Node SLAs

You use the methods of the **ServiceLevelAgreementMBean** MBean to manage Node SLAs. For information on the methods and fields of the **ServiceLevelAgreementMBean**, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Global Node SLAs

Table 3–4 describes the tasks related to managing Global Node system SLAs and the operations you use to perform those tasks.

Table 3–4 Tasks Related to Managing Global Node SystemSLAs

Task	ServiceLevelAgreementMBean Method to Use
Associate Global Node with SLA	**loadGlobalSlaByType** **loadGlobalSlaFromUrlByType**
View Global Node SLA	**retrieveGlobalSlaByType**

Service Provider Group Node SLAs

Table 3–5 describes the tasks related to Service Provider Group Node SLAs and the operations you use to perform those tasks.

Table 3–5 Tasks Related to Managing Service Provider Node SLAs

Task	ServiceLevelAgreementMBean Operation to Use
Associate Service Provider Node with SLA	**loadServiceProviderGroupSlaByType** **loadServiceProviderGroupSlaFromUrlByType**
View Service Provider Node SLA	**retrieveServiceProviderGroupSlaByType**

Subscriber SLAs

Subscriber SLAs are a feature that can be developed using the Platform Development Studio. Table 3–6 describes the tasks related to managing Subscriber SLAs and the operations you use to perform those tasks.

Table 3–6 Tasks Related to Managing Subscriber SLAs

Task	Operation to Use
Associate Subscriber with SLA	**loadGlobalSlaByType** **loadGlobalSlaFromUrlByType**
View Subscriber SLA	**retrieveGlobalSlaByType**

Custom SLAs

Custom SLAs are a feature that can be developed using the Platform Development Studio. This section describes the management of Custom SLAs.

You use the methods of the **ServiceLevelAgreementMBean** MBean to manage custom SLAs. For information on the **ServiceLevelAgreementMBean** methods and fields, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Custom XSDs

Table 3–7 describes the tasks related to managing Custom XSDs and the operations you use to perform those tasks.

Table 3–7 Tasks Related to Managing Custom XSDs

Task	Operation to Use
Set up Custom XSD	**setupCustomSlaXSDDefinition** **setupCustomSlaXSDDefinitionFromUrl**
View Custom XSD	**retrieveCustomSlaXSDDefinition**
Count Custom XSD	**countCustomSlaXSDDefinition**
List Custom XSD	**listCustomSlaXSDDefinition**

Custom Application Group SLAs

Table 3–8 describes the tasks related to managing Custom Application Group SLAs and the operations you use to perform those tasks.

Table 3–8 Tasks Related to Managing Custom Application Group SLAs

Task	Operation to Use
Associate Custom Application Group with SLA	**loadApplicationGroupSlaByType** **loadApplicationGroupSlaFromUrlByType**
View Custom Application Group SLA	**retrieveApplicationGroupSlaByType**
Count Custom Application Group SLA	**countApplicationGroupsByType**
List Custom Application Group SLA	**listApplicationGroupsByType**

Custom Service Provider Group SLAs

Table 3–9 describes the tasks related to managing Custom Service Provider Group SLAs and the operations you use to perform those tasks.

Table 3–9 Tasks Related to Managing Custom Service Provider Group SLAs

Task	Operation to Use
Associate Custom Service Provider Group with SLA	**loadServiceProviderGroupSlaByType** **loadServiceProviderGroupSlaFromUrlByType**
View Custom Service Provider Group SLA	**retrieveServiceProviderGroupSlaByType**
Count Custom Service Provider Group SLA	**countServiceProviderGroupsByType**
List Custom Service Provider Group SLA	**listServiceProviderGroupSlaTypes** **listServiceProviderGroupsByType**

Custom Global SLAs

Table 3–10 describes the tasks related to managing Custom Global SLAs and the operations you use to perform those tasks.

Table 3–10 Tasks Related to Managing Custom Global SLAs

Task	Operation to Use
Associate Custom Global with SLA	**loadGlobalSlaByType** **loadGlobalSlaFromUrlByType**
View Custom Global SLA	**retrieveGlobalSlaByType**
Count Custom Global SLA	**countGlobalSlaTypes**
List Custom Global SLA	**listGlobalSlaTypes**

Mapping of Deny Codes to HTTP Responses

Services Gatekeeper provides a mapping mechanism to allow you to assign a specific HTTP response for a given deny code during SLA validation. Table 3–11 describes the mapping.

Table 3–11 Deny Code to HTTP Response Mapping

From	To
errorCode in DenyCodes	■ HTTP status code ■ HTTP content type header ■ HTTP body

SLA validation uses the mapper to resolve which HTTP response to send. Validation no longer uses static HTTP responses. If no mapping exists for a deny code, an HTTP response is generated from the built-in list of deny code definitions.

Services Gatekeeper provides a REST API to manage these mappings along with a corresponding MBean interface to provide troubleshooting capabilities for the administrator. The OCSG database provides cluster-wide data storage and a cache layer. The REST API consumes and produces both JSON and XML content types:

- application/xml
- application/json

You can manage mappings using both `DafConfigurationsMBean` and the Partner Manager REST API.

Table 3–12 illustrates sample REST interface requests.

Table 3–12 Example REST Requests

Operation	Curl Example
Get a mapping	`curl -u api-admin:weblogic1 http://127.0.0.1:8001/prm_pm_rest/services/prm_pm/services/partner_manager/denycode/mappings/34`
Remove a mapping	`curl -u api-admin:weblogic1 -X DELETE http://127.0.0.1:8001/prm_pm_rest/services/prm_pm/services/partner_manager/denycode/mappings/34`
Add a mapping	`curl -u api-admin:weblogic1 http://127.0.0.1:8001/prm_pm_rest/services/prm_pm/services/partner_manager/denycode/mappings -H 'Content-Type: application/json' --data-binary '{"mappingItem":{"denyCode":34,"httpStatus":403,"httpContentType":"application/json","httpBody":"{\"message\":\"Forbidden123\"}"}}' --compressed`
List mappings	`ccurl -u api-admin:weblogic1 'http://127.0.0.1:8001/prm_pm_rest/services/prm_pm/services/partner_manager/denycode/mappings/34'`

The following sections describe in detail the operations you can perform using the REST API.

List Deny Code Definitions

When no mapping exists for a deny code, a default HTTP response from this list is returned. To list the built-in deny code definitions, issue an HTTP GET to the URL in the following example:

```
GET prm_pm_rest/services/prm_pm/services/partner_manager/denycode/definitions
```

The response returns a list of the deny code definitions in descending order.

JSON Example

```
{"denyCodeDefinitionItems":
  {"items": [
    {
      "name": "NO_APP_INSTANCE",
      "denyCode": 1,
      "httpStatus": 400,
      "httpContentType": "text/plain",
      "httpBody": "Application instance does not exist"
    },
    {
      "name": "APP_INST_DEACTIVATED",
      "denyCode": 2,
      "httpStatus": 400,
      "httpContentType": "text/plain",
      "httpBody": "Application instance is not active"
    },
    ...
    ]
  }
}
```

XML Example

```
<denyCodeDefinitionItems>
  <denyCodeDefinitionItem>
    <denyCode>1</denyCode>
    <httpBody>Application instance does not exist</httpBody>
    <httpContentType>text/plain</httpContentType>
    <httpStatus>400</httpStatus>
    <name>NO_APP_INSTANCE</name>
  </denyCodeDefinitionItem>
  <denyCodeDefinitionItem>
    <denyCode>2</denyCode>
    <httpBody>Application instance is not active</httpBody>
    <httpContentType>text/plain</httpContentType>
    <httpStatus>400</httpStatus>
    <name>APP_INST_DEACTIVATED</name>
  </denyCodeDefinitionItem>
  ...
```

List Mappings

To list the mappings of deny codes to HTTP responses, issue an HTP GET to the URL shown in the following example:

```
GET prm_pm_rest/services/prm_pm/services/partner_manager/denycode/mappings
```

Parameters

Table 3–13 describes the request's optional parameters:

Table 3–13 Parameters for GET List Mappings

Name	Type	Description
skip	Positive Integer	Specifies how many mappings to skip. For example, set skip to 0 to obtain the full list. For the list 0, 1, 2, 3, 4, 5, 6, setting skip to 2 would return items 2, 3, 4, 5, 6. Default is 0.

Table 3–13 (Cont.) Parameters for GET List Mappings

Name	Type	Description
limit	Positive Integer	For a full list with no items omitted, specify 0. For the list 0, 1, 2, 3, 4, 5, 6 specifying limit of 5 would return 0, 1, 2, 3, 4. Default is 0.

Returns a standard response of 200 with the list of MappingItem items ordered by deny code in ascending order

JSON Example

```
{"mappingItems":
  {"items": [
    {
      "denyCode": 34,
      "httpStatus": 403,
      "httpContentType": "application/json",
      "httpBody": "{\"message\":\"Forbidden123\"}"
    },
    {
      "denyCode": 35,
      "httpStatus": 403,
      "httpContentType": "application/json",
      "httpBody": "{\"message\":\"Forbidden123\"}"
    },
    {
      "denyCode": 37,
      "httpStatus": 403,
      "httpContentType": "application/json",
      "httpBody": "{\"message\":\"Forbidden123\"}"
    }]
  }
}
```

An empty list would look like this:

```
{
  "mappingItems":{
    "items":[
    ]
  }
}
```

XML Example

```
<mappingItems>
  <mappingItem>
    <denyCode>34</denyCode>
    <httpBody>{"message":"Forbidden123"}</httpBody>
    <httpContentType>application/json</httpContentType>
    <httpStatus>403</httpStatus>
  </mappingItem>
  <mappingItem>
    <denyCode>35</denyCode>
    <httpBody>{"message":"Forbidden123"}</httpBody>
    <httpContentType>application/json</httpContentType>
    <httpStatus>403</httpStatus>
  </mappingItem>
  <mappingItem>
    <denyCode>37</denyCode>
```

```
    <httpBody>{"message":"Forbidden123"}</httpBody>
    <httpContentType>application/json</httpContentType>
    <httpStatus>403</httpStatus>
  </mappingItem>
</mappingItems>
```

An empty list would look like this:

```
<mappingItems/>
```

Add a Mapping

To add a mapping or overwrite an existing mapping, issue a POST command to the following URL with a mapping item:

```
POST prm_pm_rest/services/prm_pm/services/partner_manager/denycode/mappings
```

JSON Payload Example

```
{"mappingItem":
  {
    "denyCode": 36,
    "httpStatus": 403,
    "httpContentType": "application/json",
    "httpBody": "{\"message\":\"Forbidden123\"}"
  }
}
```

XML Payload Example

```
<mappingItem>
  <denyCode>36</denyCode>
  <httpBody>{"message":"Forbidden123"}</httpBody>
  <httpContentType>application/json</httpContentType>
  <httpStatus>403</httpStatus>
</mappingItem>
```

Returns a status code 201 with the location header pointing to the end of the created mapping as shown in the following example:

```
http://127.0.0.1:8001/prm_pm_rest/services/prm_pm/services/partner_
manager/denycode/mappings/prm_pm/services/partner_manager/denycode/mappings/34
```

Get a Mapping

To get an existing mapping item, issue a GET to the URL in the following example, replacing `{denyCode}` with the deny code for which you want the mapping:

```
http://127.0.0.1:8001/prm_pm_rest/services/prm_pm/services/partner_
manager/denycode/mappings/{denyCode}
```

Returns a standard HTTP response with a status code 200 and the payload. Produces the added MappingItem item, including any automatically populated properties like `httpContentType`.

JSON Example

```
{"mappingItem":
  {
    "denyCode": 36,
    "httpStatus": 403,
    "httpContentType": "application/json",
    "httpBody": "{\"message\":\"Forbidden123\"}"
```

```
  }
}
```

XML Example

```
<mappingItem>
  <denyCode>36</denyCode>
  <httpBody>{"message":"Forbidden123"}</httpBody>
  <httpContentType>application/json</httpContentType>
  <httpStatus>403</httpStatus>
</mappingItem>
```

Remove a Mapping

To remove a mapping, issue an HTTP Delete request with the following URL, replacing `{denyCode}` with the deny code for the mapping that you want to remove.

Returns a standard HTTP response with status code 200 and the payload. Produces the removed MappingItem item or Content-Length of 0 to indicate that there was no item to remove.

JSON Example

```
{"mappingItem":
  {
    "denyCode": 36,
    "httpStatus": 403,
    "httpContentType": "application/json",
    "httpBody": "{\"message\":\"Forbidden123\"}"
  }
}
```

XML Example

```
<mappingItem>
  <denyCode>36</denyCode>
  <httpBody>{"message":"Forbidden123"}</httpBody>
  <httpContentType>application/json</httpContentType>
  <httpStatus>403</httpStatus>
</mappingItem>
```

Data Definitions

The REST API uses the following data definitions.

Table 3–14 describes the definition of a deny code mapping to an HTTP response. All properties are mandatory:

Table 3–14 DenyCodeDefinitionItem Data Definition

Property Name	Description	Type
denyCode	A non-negative integer code that specifies the reason for denial in SLA enforcement.	Int
httpBody	Text that is used in HTTP response body.	String
httpContentType	Statically set to plain text and used in the Content-Type header in an HTTP response.	String
httpStatus	Status code used in HTTP response.	Int

Table 3–14 (Cont.) DenyCodeDefinitionItem Data Definition

Property Name	Description	Type
name	A more friendly identifier for this deny reason. Has a one to one relationship with denyCode.	String

Table 3–15 describes the definition of a MappingItem item.

Table 3–15 MappingItem Item

Property Name	Description	Type	Mandatory
denyCode	A non-negative integer code that identifies this deny reason in SLA enforcement.	Int	true
httpStatus	HTTP response status code.	Int	true
httpBody	The text that is sent in the HTTP response body. If not present the following applies: ■ No content will be returned in the HTTP response for this deny code. ■ No content type header will be returned in the HTTP response for this deny code.	String	false
httpContentType	Used in Content-Type header in an HTTP response. If not present, it will be set to default value of text / plain	String	false

Error Handling

You can receive the following error codes when using the REST APIs:

- 400 - Omitting a mandatory parameter
  - JSON example:

```
{
  "SOAPException":{
    "message":"Mandatory parameter missing: httpStatus"
  }
}
```

  - XML example:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:SOAPException xmlns:ns2="http://ocsg.oracle/portal/ws/common">
 <message>Mandatory parameter missing: httpStatus</message>
</ns2:SOAPException>
```

- 400 - Using illegal parameter value
  - JSON example:

```
{
 "SOAPException":{
 "message":"Mandatory parameter cannot be negative: denyCode"
 }
}
```

  - XML example:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:SOAPException xmlns:ns2="http://ocsg.oracle/portal/ws/common">
 <message>Mandatory parameter cannot be negative: denyCode</message>
</ns2:SOAPException>
```

- 400 - When API user doesn't have a partner manager access role
  - JSON Example:

```
{
  "SOAPException":{
    "message":"The login user type is invalid!"
  }
}
```

  - XML Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:SOAPException xmlns:ns2="http://ocsg.oracle/portal/ws/common">
 <message>The login user type is invalid!</message>
</ns2:SOAPException>
```

- 404 - Attempting to reach a non-existing endpoint
- 405 - Attempting to use an unsupported HTTP operation
- 500 - An internal error happens. Use incidentID to correlate with the error l (default.log)
  - JSON example:

```
{"incidentID":"E-8b4cc44c49534f72a14f5b026103a145"}
```

  - XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<internalServerException>
 <incidentID>E-700cf6bd27a940fc8476d9cfbeabe2ab</incidentID>
</internalServerException>
```

MBean Interface

An interface uses primitive types to assert that it can be managed through the WebLogic console. In results, JSON formatted Strings encapsulate a complex structure in the primitive String type.

> **Note:** Gatekeeper *attributes* are read-only values, while *operations* are executable functions.

The following interface is in DafGeneralInformation:

```
public interface DenyCodeMappingsOperations {
  /**
 * Use this operation to find out the available deny codes which you can
 * re-configure mappings for. If no mapping is configured for a specific deny
 * code, the response in this listing applies.
 *
 * @return Array of deny code definitions. Each definition is in JSON format.
 * @throws ManagementException If deny deny codes could not be fetched from
 * storage.
 */
```

```
String[] getDenyCodeDefinitions() throws ManagementException;

/**
* Use this operation to view the existing deny code to HTTP mappings.
*
* @param offset Offset to use when listing mappings.
* @param limit Limit the number of results to retrieve.
* @return Array of mappings, ordered by deny code (ascending). Each Mapping is in
* JSON format.
* @throws ManagementException If mappings could not be retrieved from Storage.
*/
String[] getMappings(int offset, int limit) throws ManagementException;

/**
* Use this operation to view the existing deny code to HTTP mappings.
*
* @param denyCode Deny code to show mapping for.
* @return Mapping in JSON format.
* @throws ManagementException If mapping could not be retrieved from Storage.
*/
String getMapping(int denyCode) throws ManagementException;

/**
* Use this operation to specify what HTTP response to generate for a given deny
* code.
* @param denyCode Numeric value of deny deny code.
* @param httpCode HTTP status code to use for this deny code.
* @param contentType The content type that matches content in httpBody parameter.
* @param httpBody HTTP Body to use for this deny code.
* @return Saved mapping in JSON format, eg {"denyCode": 40,"httpCode":
* 400,"httpBody": "DEFAULT"}
* @throws ManagementException If mapping could not be saved to Storage.
*/
String setMapping(int denyCode, int httpCode, String contentType, String
httpBody) throws ManagementException;

/**
* Use this operation to remove an existing mapping.
*
* @param denyCode Deny code to remove mapping for.
* @return Removed mapping in JSON format or null if no mapping was found.
* @throws ManagementException If mapping could not be removed from Storage.
*/
String removeMapping(int denyCode) throws ManagementException;

}
```

SLA Enforcement

In SLA enforcement, you can do the following when an SLA Exception is thrown:

```
throwSLAException(DenyCodes.API_STATE_INVALID);
```

EDR Details

When a request is denied, the deny code is added to the EDR. See the highlighted DenyCode in this example:

```
[03-30 03:42:07:DEBUG EdrInternalPublisher.java] *** EDR:
ContainerTransactionId = null
```

```
Method = null
HttpStatusCode = 401
TsAfAT = 1490881317793
TsAfNT = 1490881317792
ReqAction = ["seq=1, name=OAuth2Validator, status=Success", "seq=2,
name=SLAEnforcement, status=Reject, err='No Service Contract found for type: %s'"]
Position = after
ServiceProviderId = partner
DenyCode = 34
TransactionId = 04a6dce8-5f86-4092-99d0-5bd0af36881a_IDX_4
ServiceName = /ECHOServer/1
State = ENTER_AT
Class = oracle.ocsg.daf.trafficlogger.SingleTierTrafficLogger
ApplicationId = exposed-calendar
TsBeAT = 1490881317725
HttpMethod = POST
status = Reject
TsBeNT = 1490881317734
Timestamp = 1490881317793
Direction = south
Source = exception
URL = /ECHOServer/1/echo
ServiceProviderGroup = gold
AppInstanceId = appkey_Password2
RspMsgSize = 1468
ErrCat = ActionErr
ServerName = Server1
ApiId = ECHOServer
ReqMsgSize = 16
```

Tranformation of Error Content

HTTP 4xx responses that originate from Actions or security providers go through content transformation before being sent to the caller. Table 3–16 describes the supported conversions:

Table 3–16 Supported Conversions for HTTP 4xx Responses

From	To
JSON	XML
XML	JSON
Plain	JSON
Plain	XML

If no transformation is needed or one cannot be performed, the original response is sent.

4

Managing External SLAs

This chapter describes how external Service Level Agreements (SLAs) are managed and provisioned in Oracle Communications Services Gatekeeper (OCSG).

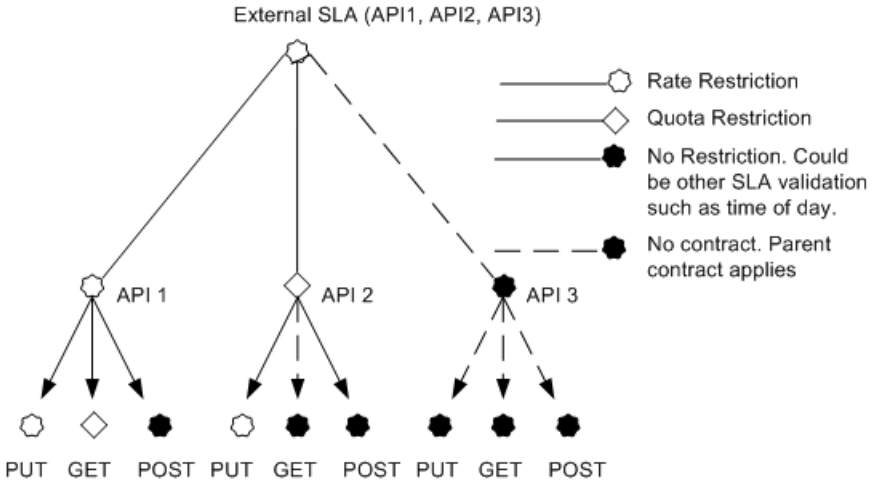
Understanding External SLAs

An *external service level agreement* (SLA) is equivalent to a plan in API Cloud Service. In Oracle Communications Services Gatekeeper, it is a new data entity that is referred to from group SLAs. It is stored in a database and managed from REST APIs or MBean interfaces. An application subscribes to an external SLA through `createApplication` and `updateApplication` operations.

The validity period for an external SLA, which consists of start and end dates, is attached to the SLA itself and also to its relationship to the group SLA. For example, in the first case an investor might want to offer a free service for any application but make the plan valid only through the month of February. For a scenario in which the validity period is attached to the external SLA's relationship to the group SLA, you could have the application subscribe to a plan on an application level. In this case, application A and application B might both subscribe to a plan, but application A pays quarterly (three-month period) while application B pays annually (twelve-month period).

An external SLA is constructed in three levels from root to leaves. A *composed service contract* can be composed of all APIs in the plan, but it can have another composition as well. Figure 4–1 illustrates the three levels of an external SLA.

Figure 4–1 The Three Levels of an External SLA



You can set an exemption or constraint on the leaf level in the tree, which consists of the PUT, GET, and POST verbs. An exemption cannot be set on the composed service contract level. Note that some verbs lack contract on the service contract level. For example, SLA -> API2 -> GET does not have a contract, so the leaf for this verb is actually External SLA -> API2. At API2, you can set the service type contract to exemption, but then you also must set exemption on the service contract level.

Quota usage occurs as follows: 1) for each successful transaction, the quota is consumed in all levels that are used; 2) for each transaction that has an exception in OCSG before a send request to the back-end server, no quota is consumed at any level.

Exemptions and constraints are enforced as follows. When validating the current API action, OCSG first looks at the service contract. If it's not found, OCSG goes to the parent level, which is service type contract. If that is also not found, it looks at the composed service contract to see if the API is part of the composed contract. If not, then there is no contract for this action and the request is denied.

If OCSG finds a service contract, it checks whether it's an exemption. If so, it validates the service contract but not the service type or global level.

If OCSG finds a service contract and it is a constraint, all parent contracts will also be constraints and each will be evaluated in order: 1) service contract, 2) service type contract, and 3) composed service contract.

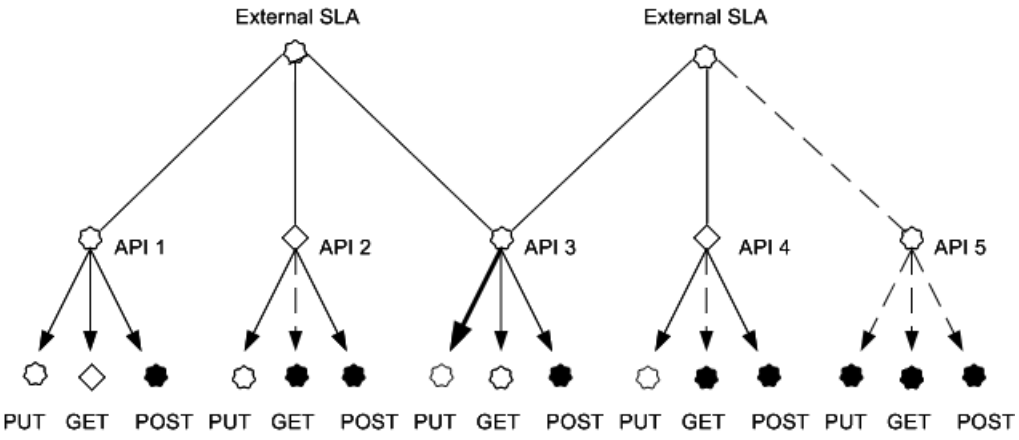
If the request fails on any level, all budgets that have been used are returned.

The exemption is valid only with one external SLA. If there is overlap with other external SLAs or provider-group SLAs, they are treated separately. The return of used budgets applies across all levels.

Understanding SLA Overlaps

OCSG allows overlaps between external SLAs for counters but you should use caution because overlap can quickly become quite complex. Figure 4–2 illustrates one type of overlap.

Figure 4–2 Overlap Between External SLAs



Here two external SLAs overlap in API3 PUT. OCSG evaluates each external SLA sequentially, so restrictions from both plans are enforced. In this example, an application can subscribe to only one external SLA. Of all the external SLAs an application subscribes to, none can contain the same API, including the method.

In addition, when updating an existing external SLA, OCSG checks for API subscription overlap. If an application subscribes to the same API through different external SLAs, the update will fail.

Managing with MBean

Two new SLA types have been created to support re-use of SLAs by several applications: `external` and `sytem:geo_external`, which are managed from MBean `com.bea.wlcp.wing.account.management.ServiceLevelAgreementMbean`.

```
/**
 * Lists the external IDs for the specified SLA type.
 *
 * The following system level SLA types are valid for externals:
 * external - External SLA stored and enforced only on the local site.
 * system:geo_external - External SLA replicated and enforced on all
 * domains in a geo-redundant configuration.
 * <p><strong>Scope</strong>: Domain</p>
 *
 * @param slaType a system level application group SLA type ("external" or
 * "system:geo_external") or an SLA type defined by setupCustomSlaXSDDefinition
 * @param offset Offset within the complete resultset. Must be >= 0.
 * @param size Number of entries to return. 0 means no limit.
 * @throws InputManagementException if slaType is null or not applicable.
 * @return The application groups matching the criteria.
 */
public String[] listExternalIdsByType(String slaType, int offset, int size)
throws InputManagementException;

/**
 * Loads a system level or custom SLA for the specified external id.
 *
 * The following system level SLA types are valid for the external:
 * external - External SLA stored and enforced only on the local site.
 * system:geo_external - External SLA replicated and enforced on all
 * domains in a geo-redundant configuration.
 * <p><strong>Scope</strong>: Domain</p>
 *
 * @param slaType A system level external SLA type ("external" or
 * "system:geo_external") or an SLA type defined by setupCustomSlaXSDDefinition
 * @param id The external SLA identifier.
 * @param fileContent The new service level agreement to use.
 * A Null or empty string removes the SLA.
 * @throws ManagementException Validation of service
 * level agreement failed.
 * @throws KeyNotFoundException If the group does not exist.
 * @throws InputManagementException if the slaType is null or not applicable.
 */

public void loadExternalSlaByType(String slaType, String id,
     String fileContent)
throws ManagementException, KeyNotFoundException, InputManagementException;

/**
 * Retrieves a system or custom Service Level Agreement for the specified external
 * id.
 *
 * The following system level SLA types are valid for external:
 * external - External SLA stored and enforced only on the local site.
 * system:geo_external - External SLA replicated and enforced on all
```

```
 * domains in a geo-redundant configuration.
 * <p><strong>Scope</strong>: Domain</p>
 *
 * @param slaType A system level external SLA type ("external" or
 * "system:geo_external") or an SLA type defined by setupCustomSlaXSDDefinition
 * @param id The external SLA identifier.
 * @throws InputManagementException if slaType is null or not applicable.
 * @return The custom Service Level Agreement.
 */
public String retrieveExternalSlaByType(String slaType, String id)
throws InputManagementException;
```

Listing from the Database

The **wing_external_slas** database table stores external SLAs. Table 4–1 describes **wing_external_slas**:

Table 4–1 The wing_external_slas Database Table

Field	Type	Null	Key	Default
id	varchar (255)	No	Primary	Null
sla_content	blob	Yes		Null
description	varchar (255)	Yes		Null
state	integer (10)	Yes		Null
stored_ts	bigint(20)	No		Null

The table **global_wing_external_slas** has the same definition.

The following example shows how to list SLAs from the MySQL console:

```
select sla_content from wlng_external_slas;
```

Understanding the XSD

There is a new XML system descriptor (XSD) for external SLAs, with three parts for the different levels:

- service contract

  Defines the constraints and exemptions on the API resource table row level

- service type contract

  Defines the constraints and exemptions on the API level

- composed service contract

  Defines the constraints on the PLAN level (*no exemptions* on the PLAN level)

> **Note:** No rate or quota element means that the rate or quota is unlimited on that level (or method).

The XSD pathname is:

```
{SRC}\bea\modules\account\src\main\resources\sla_schema\external_sla_file.xsd
```

The XSD is based on the original application group SLA, which is used to support external SLA subscription in the application, with these three differences: no group ID attribute, `startDate` and `endDate` are optional, and the `isExemption` attribute is added.

Figure 4–3 illustrates these differences:

Figure 4–3 XSD Differences for External SLA

```
··<!---========================================================================
····A·service·contract:·defines·the·constraints/exemptions·on·the·API·reso
··==============================================================================
··<xs:complexType·name="serviceContract">
····<xs:sequence>
·····<xs:element·name="startDate"·minOccurs="0"·maxOccurs="1"·type="xs:dat
·····<xs:element·name="endDate"·minOccurs="0"·maxOccurs="1"·type="xs:date"
·····<xs:element·name="scs"·type="xs:string"/>
·····<xs:element·name="method"·type="xs:string"·minOccurs="0"·maxOccurs="u
·····<xs:element·name="contract"·type="contract"·minOccurs="0"·maxOccurs="
·····<xs:element·name="overrides"·minOccurs="0"·maxOccurs="1">
·····<xs:complexType>
·····<xs:sequence>
·····<xs:element·name="override"·type="override"·minOccurs="1"·maxOc
·····</xs:sequence>
·····</xs:complexType>
·····</xs:element>
·····<xs:element·name="enforceAcrossGeoSites"·type="xs:boolean"·minOccurs=
····</xs:sequence>
····<!---when·isExemption·is·true,·regards·it·as·exemption·--->
····<xs:attribute·name="isExemption"·type="xs:boolean"·default="false"/>
··</xs:complexType>
```

The XSDs for application and service-provider SLAs have been extended with the `externalSla` element so that you can embed external SLAs in application or service-provider group SLAs.

Table 4–2 describes support for external SLA elements:

Table 4–2 Support for External SLA Elements

Element	Description
override and overrides	In a serviceContract, can be used to set an alternative contract during a specific time. For example, day of week, time of day, and so on.
limitExceeded (quota)	If true, traffic can continue after the quota constraint reaches its limit. The counter always calculates. Default: false.
methodAccess and blacklistedMethod (contract)	A general purpose tunneling feature. You can pass something to an action that isn't part of the interface of the API but is instead attached to the SLA. Could for instance be a price plan identifier or similar.
params and MethodParameters (contract)	Describes parameters and method parameters.
requestContext (contract)	A general purpose tunneling feature. You can pass something to an action that isn't part of the API interface but instead attached to the SLA, for instance a price plan identifier or similar.
resultRestrictions (contract)	Filter responses before returned to caller.

Subscribing to External SLAs

Currently an application subscribes to APIs by invoking the `CREATE` or `UPDATE` application REST calls. An application also has the option to subscribe to external

SLAs. It can subscribe to both APIs and external SLAs or it can subscribe to only one of them. The following examples illustrate each of these cases.

External SLA only

The following example illustrates a REST `createApplication` statement for an external SLA only:

```
{
  "createApplication":{
    "application":{
      "applicationID":"leavePlanApp",
      "applicationName":"leavePlanApp",
      "description":"Plan a RESTful vacation or leave",
      "partnerName":"partner",
      "applicationAPIs":[

      ],
      "externalSlas":[
        {
          "id":"myCalendar"
        },
        {
          "id":"myParental"
        },
        {
          "id":"myVacation"
        }
      ],
      "trafficPassword":"d2VibG9naWMx",
      "icon":"expressive/sunny.png"
    }
  }
}
```

The `updateApplication` statement would look like this:

```
{
  "updateApplication":{
    "application":{
      "applicationID":"leavePlanApp",
      "applicationName":"leavePlanApp",
      "description":"Plan a RESTful vacation or leave",
      "partnerName":"partner",
      "applicationAPIs":[

      ],
      "externalSlas":[
        {
          "id":"myCalendar"
        },
        {
          "id":"myParental"
        },
        {
          "id":"myVacation"
        }
      ],
      "trafficPassword":"d2VibG9naWMx",
      "icon":"expressive/sunny.png"
    }
```

```
  }
}
```

The XML for the application group SLA looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Sla xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="app_sla_file.xsd"
  applicationGroupID="partner-leavePlanApp">
  <externalSla>
    <id>myCalendar</id>
    <id>myParental</id>
    <id>myVacation</id>
  </externalSla>
</Sla>
```

API Only

The following example illustrates a REST `updateApplication` statement for an API only:

```
{
  "updateApplication":{
    "application":{
      "applicationID":"leavePlanApp",
      "applicationName":"leavePlanApp",
      "description":"Plan a RESTful vacation or leave",
      "effectiveFrom":"2016-04-15",
      "effectiveTo":"3017-01-12",
      "trafficUser":"partner_leavePlanApp",
      "appKey":"Password1",
      "partnerName":"partner",
      "quota":{
        "days":"1",
        "qtaLimit":"10000000"
      },
      "rate":{
        "reqLimit":"1500",
        "timePeriod":"1"
      },
      "applicationAPIs":[
        {
          "apiName":"ECHOServer"
        }
      ],
      "externalSlas":[
      ],
      "trafficPassword":"d2VibG9naWMx",
      "icon":"expressive/leavePlanApp.png"
    }
  }
}
```

The XML for the application group SLA looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Sla xsi:noNamespaceSchemaLocation="app_sla_file.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  applicationGroupID="partner-leavePlanApp">
  <serviceContract>
    <startDate>2016-04-15+02:00</startDate>
```

```
    <endDate>3017-01-12+01:00</endDate>
    <scs>ECHOServer</scs>
    <method>POST_/echo</method>
  </serviceContract>
  <composedServiceContract>
    <composedServiceName>leavePlanApp</composedServiceName>
    <service>
      <serviceTypeName>ECHOServer</serviceTypeName>
    </service>
    <startDate>2016-04-15+02:00</startDate>
    <endDate>3017-01-12+01:00</endDate>
    <rate>
      <reqLimit>1500</reqLimit>
      <timePeriod>1000</timePeriod>
    </rate>
    <quota>
      <qtaLimit>10000000</qtaLimit>
      <days>1</days>
      <limitExceedOK>false</limitExceedOK>
    </quota>
  </composedServiceContract>
</Sla>
```

External SLA and API

The following example illustrates a REST `updateApplication` statement for an external SLA *and* an API:

```
{
"updateApplication":{
  "application":{
    "applicationID":"leavePlanApp",
    "applicationName":"leavePlanApp",
    "description":"Plan a RESTful vacation or leave",
    "effectiveFrom":"2016-04-15",
    "effectiveTo":"3017-01-12",
    "trafficUser":"partner_leavePlanApp",
    "appKey":"Password1",
    "partnerName":"partner",
    "quota":{
      "days":"1",
      "qtaLimit":"10000000"
    },
    "rate":{
      "reqLimit":"1500",
      "timePeriod":"1"
    },
    "applicationAPIs":[
      {
        "apiName":"ECHOServer"
      }
    ],
    "externalSlas":[
    ],
    "trafficPassword":"d2VibG9naWMx",
    "icon":"expressive/leavePlanApp.png"
    }
  }
}
```

The XML for the application group SLA looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Sla xsi:noNamespaceSchemaLocation="app_sla_file.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  applicationGroupID="partner-leavePlanApp">
  <serviceContract>
    <startDate>2016-04-15+02:00</startDate>
    <endDate>3017-01-12+01:00</endDate>
    <scs>ECHOServer</scs>
    <method>POST_/echo</method>
  </serviceContract>
  <composedServiceContract>
    <composedServiceName>leavePlanApp</composedServiceName>
    <service>
      <serviceTypeName>ECHOServer</serviceTypeName>
    </service>
    <startDate>2016-04-15+02:00</startDate>
    <endDate>3017-01-12+01:00</endDate>
    <rate>
      <reqLimit>1500</reqLimit>
      <timePeriod>1000</timePeriod>
    </rate>
    <quota>
      <qtaLimit>10000000</qtaLimit>
      <days>1</days>
      <limitExceedOK>false</limitExceedOK>
    </quota>
  </composedServiceContract>
  <externalSla>
    <id>myCalendar</id>
    <id>myParental</id>
    <id>myVacation</id>
  </externalSla>
</Sla>
```

Life Cycle

An external SLA can be active or inactive and it can be in use or not in use. You can create an external SLA with the status set to active or inactive and you can change the status at any time. An application can subscribe to, and be subscribed to, an external SLA in either of these states. Table 4–3 describes the life cycle rules for an external SLA.

Table 4–3 Life Cycle Rules

	Update	Delete	Subscription	Enforce in Traffic
Active	Y	Y	Y	Y
Inactive	Y	Y	Y	-

The following rules apply:

- Applications can subscribe to an inactive external SLA during traffic processing run time but it cannot be enforced.
- An application is in use when one or more applications have subscribed to it.
- An external SLA that is in use can be updated but cannot be deleted.
- An external SLA that is *not* in use can be deleted whether its status is active or inactive.

Identifying the Plan

The plan ID in the EDR identifies which plan was selected to enforce for a given runtime API request.

When the PLAN is active and takes effect on the traffic per the current request, the plan ID and status occur in the EDR message as follows:

```
ExtSlaIds = id = plan_id0, status=ACTIVE
```

Using External SLA Management REST Interfaces

The existing API for creating and updating applications has been extended with the `externalSLAs` data element as described in chapter Subscribing/using external SLAs.

The base URL is:

```
pm_rest/services/prm_pm/services
```

Table 4–4 provides additional information regarding payload attributes:

Table 4–4 Notes on Plan Attributes

Attribute	Comments
quotas	Optional. No `quotas` means unlimited and each unit is optional. SECONDS, MINUTES and HOURS are supported. All units (SECONDS, MINUTES, HOURS, DAYS, WEEKS, MONTHS, YEARS) can be applied in the API method and each is optional. SECONDS, MINUTES, and HOURS are released by duration. DAYS, WEEKS, MONTHS, YEARS are released by calendar at midnight for each day or Saturday (for week) or first day of month or year.
rate	Optional. The unit for timePeriod is milliseconds. If rate occurs, reqLimit and timePeriod are mandatory.
slaItem/id	Optonal. For creation, if `id` is not provided, one is generated in UUID format. For updating, use the value in the path.
slaItem/state	Optional. For creation, default is `inactive` and value is not changed for updating.
slaItem/description	Optional. Default is null.
slaItem/apis	Mandatory.
slaItem/apis/exemption	Optional. Default is false.
slaItem/apis/rate	Optional. If rate occurs, however, reqLimit and timePeriod are mandatory.
slaItem/apis/quotas	Optional. No quotas means unlimited, and each unit is optional
slaItem/apis/methods	Optional. No methods means all API methods are allowed.
slaItem/apis/methods/exemptions	Optional. Default is false.
slaItem/apis/methods/startDate	Optional. If no startDate, API's startDate is used.
slaItem/apis/methods/endDate	Optional. If no endDate, API's endDate is used.

Table 4–4 (Cont.) Notes on Plan Attributes

Attribute	Comments
slaItem/apis/methods/rate	Optional. If rate occurs, however, reqLimit and timePeriod are mandatory.
slaItem/apis/methods/quotas	Optional. No means unlimited and each unit is optional.
slaItem/apis/methods/scopes	Optional.

Creating a Plan

Create a plan by issuing a POST command to the URL:

```
partner_manager/sla/external
```

The following example illustrates:

```
POST /partner_manager/sla/external
{
  "slaItem": {
    "id": "plan_id0",
    "name": "plan_name0",
    "state": "inactive",
    "description": "desc1",
    "startDate": "2007-01-25",
    "endDate": "2097-09-25",
    "rate": {
      "reqLimit": 5000000,
      "timePeriod": 2
    },
    "quotas": [{
        "unit": "SECONDS",
        "qtaLimit": 2,
        "limitExceedOK": false
      },
      {
        "unit": "MINUTES",
        "qtaLimit": 10,
        "limitExceedOK": false
      },
      {
        "unit": "HOURS",
        "qtaLimit": 20,
        "limitExceedOK": false
      },
      {
        "qtaLimit": 5,
        "unit": DAYS,
        "limitExceedOK": false
      },
      {
        "qtaLimit": 50,
        "unit": WEEKS,
        "limitExceedOK": false
      },
      {
        "qtaLimit": 500,
        "unit": MONTHS,
        "limitExceedOK": false
      },
```

```
{
  "qtaLimit": 5000,
  "unit": YEARS,
  "limitExceedOK": false
}],
"apis": [{
  "apiId": "id-test-api0",
  "exemption": false,
  "startDate": "2007-02-26",
  "endDate": "2097-08-25",
  "rate": {
    "reqLimit": 5000000,
    "timePeriod": 2
  },
  "quotas":[{
    "days": 1,
    "qtaLimit": 4,
    "limitExceedOK": false
  },
  {
    "qtaLimit": 5,
    "unit": DAYS,
    "limitExceedOK": false
  },
  {
    "qtaLimit": 50,
    "unit": WEEKS,
    "limitExceedOK": false
  },
  {
    "qtaLimit": 500,
    "unit": MONTHS,
    "limitExceedOK": false
  },
  {
    "qtaLimit": 5000,
    "unit": YEARS,
    "limitExceedOK": false
  }],
  "methods": [{
    "path": "GET_/getPath/*",
    "exemption": true,
    "startDate": "2007-03-25",
    "endDate": "2097-07-25",
    "rate": {
      "reqLimit": 5000000,
      "timePeriod": 2
    },
    "quotas": [{
      "days": 1,
      "taLimit": 10,
      "limitExceedOK": true
    },
    {
      "qtaLimit": 5,
      "unit": DAYS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 50,
```

```
            "unit": WEEKS,
            "limitExceedOK": false
          },
          {
            "qtaLimit": 500,
            "unit": MONTHS,
            "limitExceedOK": false
          },
          {
            "qtaLimit": 5000,
            "unit": YEARS,
            "limitExceedOK": false
          }]
        },
        {
          "path": "PATCH_/patchPath/*",
          "exemption": false,
          "startDate": "2007-04-25",
          "endDate": "2097-06-25",
          "rate": {
             "reqLimit": 5000000,
             "timePeriod": 2
          },
          "quotas":[{
            "days": 1,
            "qtaLimit": 12,
            "limitExceedOK": false
          }],
          "scopes": [
            {"name":"name1", "value": "value1"},
            {"name":"name2", "value": "value2"}
          ]
        }]
    }]
  }
}
```

A successful response returns status code 201 Created to the location:

```
/partner_manager/sla/external/plan_id0
```

Getting a Plan

To get a plan, issue a GET command to the URL `/partner_manager/sla/external/`*`id`* where *`id`* is the plan ID.

```
GET partner_manager/sla/external/plan_id0
```

A successful response returns a status code 200 OK with the requested plan:

```
{
  "slaItem": {
    "id": "plan_id0",
    "name": "plan_name0",
    "state": "inactive",
    "description": "desc1",
    "startDate": "2007-01-25",
    "endDate": "2097-09-25",
    "rate": {
      "reqLimit": 5000000,
      "timePeriod": 2
```

```
},
"quotas": [{
    "days": 1,
    "qtaLimit": 5,
    "limitExceedOK": false
  },
  {
    "qtaLimit": 5,
    "unit": DAYS,
    "limitExceedOK": false
  },
  {
    "qtaLimit": 50,
    "unit": WEEKS,
    "limitExceedOK": false
  },
  {
    "qtaLimit": 500,
    "unit": MONTHS,
    "limitExceedOK": false
  },

  {
    "qtaLimit": 5000,
    "unit": YEARS,
    "limitExceedOK": false
  }],
"apis": [
  {
    "exemption": false,
    "apiId": "id-test-api0",
    "startDate": "2007-02-26",
    "endDate": "2097-08-25",
    "rate": {
      "reqLimit": 5000000,
      "timePeriod": 2
    },
    "quotas": [{
       "days": 1,
       "qtaLimit": 4,
       "limitExceedOK": false
    },
    {
      "qtaLimit": 5,
      "unit": DAYS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 50,
      "unit": WEEKS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 500,
      "unit": MONTHS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 5000,
      "unit": YEARS,
```

```
          "limitExceedOK": false
        }],
        "methods": [
        {
          "exemption": true,
          "path": "GET_/getPath/*",
          "startDate": "2007-03-25",
          "endDate": "2097-07-25",
          "rate": {
            "reqLimit": 5000000,
            "timePeriod": 2
          },
          "quotas": [{
            "days": 1,
            "qtaLimit": 10,
            "limitExceedOK": true
          },
          {
            "qtaLimit": 5,
            "unit": DAYS,
            "limitExceedOK": false
  },
          {
            "qtaLimit": 50,
            "unit": WEEKS,
            "limitExceedOK": false
          },
          {
            "qtaLimit": 500,
            "unit": MONTHS,
            "limitExceedOK": false
          },
          {
            "qtaLimit": 5000,
            "unit": YEARS,
            "limitExceedOK": false
          }],
        },
        {
          "exemption": false,
          "path": "PATCH_/patchPath/*",
          "startDate": "2007-04-25",
          "endDate": "2097-06-25",
          "rate": {
            "reqLimit": 5000000,
            "timePeriod": 2
          },
          "quotas": [{
            "days": 1,
            "qtaLimit": 12,
            "limitExceedOK": false
          }],
          "scopes": [
            {"name":"name1", "value": "value1"},
            {"name":"name2", "value": "value2"}
          ]
        }]
      }]
    }
}
```

Getting All Plans

To get all plans, issue a GET command to the URL `/partner_manager/sla/external`, as the following example shows:

```
GET /partner_manager/sla/external
```

A successful response returns a status code 200 OK with all plans:

```
{
  "slaitems": {
    "items": [
      {
        "id": "plan_id0",
        "description": "desc1"
      }
    ],
    "hasMore": false
  }
}
```

Updating a Plan

To update a plan, issue a PUT command to the following URL, where *id* is the plan ID:

`/partner_manager/sla/external/id`

The following example illustrates. A successful response returns the status code 204 No Content.

```
PUT /partner_manager/sla/external/plan_id0
{
  "slaItem": {
    "id": "plan_id0",
    "name": "plan_name0",
    "state": "inactive",
    "description": "desc1",
    "startDate": "2007-01-25",
    "endDate": "2097-09-25",
    "rate": {
      "reqLimit": 5000000,
      "timePeriod": 2
    },
    "quotas": [{
      "days": 1,
      "qtaLimit": 5,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 5,
      "unit": DAYS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 50,
      "unit": WEEKS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 500,
      "unit": MONTHS,
      "limitExceedOK": false
    },
```

```
  {
    "qtaLimit": 5000,
    "unit": YEARS,
    "limitExceedOK": false
  }],
  "apis": [{
    "apiId": "id-test-api0",
    "exemption": false,
    "startDate": "2007-02-26",
    "endDate": "2097-08-25",
    "rate": {
      "reqLimit": 5000000,
      "timePeriod": 2
    },
    "quotas":[{
      "days": 1,
      "qtaLimit": 4,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 5,
      "unit": DAYS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 50,
      "unit": WEEKS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 500,
      "unit": MONTHS,
      "limitExceedOK": false
    },
    {
      "qtaLimit": 5000,
      "unit": YEARS,
      "limitExceedOK": false
}],
    "methods": [{
      "path": "GET_/getPath/*",
      "exemption": true,
      "startDate": "2007-03-25",
      "endDate": "2097-07-25",
      "rate": {
        "reqLimit": 5000000,
        "timePeriod": 2
      },
      "quotas":[ {
        "days": 1,
        "qtaLimit": 10,
        "limitExceedOK": true
      },
      {
        "qtaLimit": 5,
        "unit": DAYS,
        "limitExceedOK": false
      },
      {
        "qtaLimit": 50,
```

```
          "unit": WEEKS,
          "limitExceedOK": false
        },
        {
          "qtaLimit": 500,
          "unit": MONTHS,
          "limitExceedOK": false
        },
        {
          "qtaLimit": 5000,
          "unit": YEARS,
          "limitExceedOK": false
        }]
      },
  {
        "path": "PATCH_/patchPath/*",
        "exemption": false,
        "startDate": "2007-04-25",
        "endDate": "2097-06-25",
        "rate": {
          "reqLimit": 5000000,
          "timePeriod": 2
        },
        "quotas":[ {
          "days": 1,
          "qtaLimit": 12,
          "limitExceedOK": false
        }]
      }]
    }]
  }
}
```

Deleting a Plan

To delete a plan, issue a DELETE command to the URL `/partner_manager/sla/external/`*`id`* where *`id`* is the plan ID. The following example illustrates how to delete a plan:

```
DELETE /partner_manager/sla/external/plan_id0
```

A successful outcome returns a status 200 OK along with the deleted plan. See "Getting a Plan" for an example of the returned plan.

Updating a Plan State

Update a plan state by issuing a `PUT` statement to the URL `/partner_manager/sla/external/`*`id`*`/state`, where *`id`* is the plan ID. The following example illustrates how to update a plan state:

```
PUT /partner_manager/sla/external/plan_id0/state
{
  "slaItem": {
    "state": "active"
  }
}
```

5

Managing Groups

This chapter describes how service provider and application groups are managed and provisioned in Oracle Communications Services Gatekeeper.

The management of groups comprises the following:

- Managing Service Provider Groups
- Managing Application Groups

Managing Service Provider Groups

Table 5–1 describes the tasks related to service provider groups and the methods you use to perform those tasks. The methods belong to **ApplicationGroupMBean** MBean. For information on the MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Table 5–1 Tasks Related to Service Provider Groups

Task	ApplicationGroupMBean Operation to Use
Get information about the number of service provider groups	**countServiceProviderGroups**
Add, remove and get information about a service provider group	**addServiceProviderGroup** **removeServiceProviderGroup** **getServiceProviderGroup**
List registered service provider groups	**listServiceProviderGroups**
Define additional properties for an service provider group	**setServiceProviderGroupProperties**

Managing Application Groups

Table 5–2 describes the tasks related to application groups and the operations you use to perform those tasks. The methods belong to **ApplicationGroupMBean** MBean. For information on the MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Table 5–2 Tasks Related to Application Groups

Task	ApplicationGroupMBean Operation to Use
Get information about the number of application groups	countApplicationGroups
Add, remove and get information about an application group	addApplicationGroup removeApplicationGroup getApplicationGroup
List registered application groups	listApplicationGroups
Define additional properties for an application group	setApplicationGroupProperties

6

Managing Service Provider and Application Accounts

This chapter describes how service provider and application accounts are managed and provisioned in Oracle Communication Services Gatekeeper.

The management of accounts comprises the following:

- Managing Application Accounts
- Managing Service Provider Accounts

Account information is described in the following sections:

- About Account States
- Account Properties
- Account Reference

Managing Application Accounts

Table 6–1 lists the tasks related to application accounts and the methods you use to perform those tasks. The methods belong to **ApplicationAccountMBean**. For detailed information on this MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Table 6–1 Tasks Related to Application Accounts

Task	ApplicationAccount MBean Methods to Use
Get information about the number of application accounts	**countApplicationAccounts**
Add, remove and get information about an application account	**addApplicationAccount** **removeApplicationAccount** **getApplicationAccount**
List registered application accounts	**listApplicationAccounts**
Define additional properties for an application account	**setApplicationAccountGroup** **setApplicationAccountProperties** **setApplicationAccountReference** **setApplicationAccountState**

Managing Service Provider Accounts

Table 6–2 lists the tasks related to service provider accounts and the methods you use to perform those tasks. These methods belong to **ApplicationAccountMBean**. For detailed information on this MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Table 6–2 Tasks Related to Service Provider Accounts

Task	ApplicationAccountMBean Operation to Use
Get information about the number of service provider accounts	**countServiceProviderAccounts**
Add, remove and get information about a service provider account	**addServiceProviderAccount** **removeServiceProviderAccount** **getServiceProviderAccount**
List registered service provider accounts	**listServiceProviderAccounts**
Define additional properties for a service provider account	**setServiceProviderAccountGroup** **setServiceProviderAccountProperties** **setServiceProviderAccountReference** **setServiceProviderAccountState**

About Account States

To provide an easy way to take an application out-of-service temporarily, service provider and application accounts have associated states. States are also used by the Partner Relationship Management module to enforce workflow management.

Accounts can be in either one of the states described in Table 6–3:

Table 6–3 Account State Indicators

Account State	Restriction on Application Belonging to Account
ACTIVATED	An application belonging to such an account is allowed to send requests to the application-facing interfaces exposed by Services Gatekeeper.
DEACTIVATED	An application belonging to such an account is not allowed to send requests to the application-facing interfaces exposed by Services Gatekeeper.

State transition is provided using these methods belonging to the **ApplicationAccountMBean** MBean:

- **setApplicationAccountState**
- **setServiceProviderAccountState**

You can filter on account state in these methods of the **ApplicationAccountMBean** MBean:

- **countApplicationAccounts**
- **countServiceProviderAccounts**
- **listApplicationAccounts**
- **listServiceProviderAccounts**

When you create an account using **addApplicationAccount** or **addServiceProviderAccount** methods, the state is always ACTIVATED.

Account Properties

An account can have a set of associated properties. These are generic key-value pairs.You cannot set these using the Administration Console, but you can use MBeans or the PRM Web Services.

The properties are displayed in these methods belonging to **ApplicationInstanceMBean**:

- **getApplicationInstance**
- **getApplicationAccount**
- **getApplicationGroup**
- **getServiceProviderAccount**
- **getServiceProviderGroup**

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

Account Reference

An account can have an associated reference. The reference is a form of alias or internal ID for the account. It correlates the account with an operator-internal ID.

The references are defined as parameters in these **ApplicationInstanceMBean** operations:

- **addApplicationInstance**
- **setApplicationInstanceReference**
- **setApplicationInstanceProperties**
- **addApplicationAccount**
- **setApplicationAccountReference**
- **addServiceProviderAccount**
- **setApplicationAccountReference**

These references are retrieved as part of the result of:

- **getApplicationInstance**
- **getApplicationAccount**
- **getServiceProviderAccount**

Set field values and use methods from the Administration Console by selecting **Container** then **Services** followed by **ApplicationInstanceMBean**. Or, use a Java application. For information on the methods and fields, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

7

Managing Sessions

This chapter describes how to configure the behavior of and manage ongoing sessions in Oracle Communications Services Gatekeeper.

About Sessions

You can configure whether to use sessions between applications and Services Gatekeeper. Sessions can be used for establishing site affinity when using geo-redundant sites.

The setting of **sessionRequired** field specifies whether sessions are used. You can make sessions required, optional, or have sessions disabled.

When using sessions, the application must get a session from the Session Web Service before using the other application-facing interfaces. The application must also provide the session ID in all requests to the application-facing interfaces, except for requests to the Session Web Service.

A session has a unique ID. A session, once established, can have two states:

- **ACTIVE**
- **INVALID**

After a session has been established, it is always in the **ACTIVE** state.

Table 7–1 describes the possible states for a session, the state into which it can transition from the current state, and the condition under which such a transition occurs.

Table 7–1 Session States and Conditions for Successful Transition

Current Session State	Valid Transition State	Scenario in Which Transition Occurs
ACTIVE	INVALID	The age of the ACTIVE state of a session is older than a configurable time-interval. See the validityTime field of the **ApplicationSessionMBean** MBean in the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.
ACTIVE	Not applicable This is not a state but rather that the session is not current or does not exist.	A management user performs one of the following **ApplicationSessionMBean** operations: ▪ **destroySession** ▪ **destroyApplicationInstanceSession** ▪ **destroyApplicationSessions** ▪ **destroyServiceProviderSessions**

Table 7–1 (Cont.) Session States and Conditions for Successful Transition

Current Session State	Valid Transition State	Scenario in Which Transition Occurs
INVALID	Not applicable This is not a state but rather that the session is not current or does not exist.	The age of the INVALID state of a session is older than a configurable time-interval. See the expiryTime field in **ApplicationSessionMBean in the** "All Classes" section of *Services Gatekeeper OAM Java API Reference*. An **ApplicationSessionMBean** user performs one of the following management operations: ■ **destroyApplicationInstanceSession** ■ **destroyApplicationSessions** ■ **destroySession** ■ **destroyServiceProviderSessions**

For a description of the attributes and operations of the **ApplicationSessionMBean** MBean, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*

8

Defining Service Provider Group and Application Group SLAs

This chapter describes how you can define service provider group and application group SLAs in Oracle Communications Services Gatekeeper.

An application's usage policies of Services Gatekeeper are specified in Service Level Agreement (SLA) XML files. There is an SLA associated with the service provider group and one associated with the application group. For more information on this administration model, see "About Service Level Agreements and Accounts".

For a sample of a complete SLA, see "Sample SLA XML File".

This chapter describes the following topics:

- Structure of a Service Level Agreement
- Structure of a Contract
- Structure of a Composed Service Contract

Structure of a Service Level Agreement

The xsds for the SLAs are in **/sla_schema/** in *Middleware_Home***/ocsg_6.0/**modules/com.bea.wlcp.wlng.account_*release_level*.**jar**:

- **app_sla_file.xsd** is the application group SLA xsd.
- **sp_sla_file.xsd** is and the service provider group SLA xsd.

An SLA contain can contain three different types of contracts:

- `<serviceContract>`
- `<serviceTypeContract>`
- `<composedServiceContract>`

The SLA must contain at least one `<serviceContract>`. The `<serviceTypeContract>` and `<composedServiceContract>` elements are optional.

The `<serviceTypeContract>` element includes usage restrictions per service type. The `<serviceContract>` element defines restrictions on a more granular level: per interface and method. The `<composedServiceContract>` element defines usage restrictions that apply to multiple communication services.

It is possible to override parts of a `<serviceContract>` based on time-of-day and day-of-week. This is not possible in a `<serviceTypeContract>` or `<composedServiceContract>`.

Example 8–1 shows the service provider level SLA XML file's main structure.

Note: If the SLA XML file has white space before the <?xml . . .> tag, the SLA will not load.

Example 8–1 Service Level Agreement Structure

```
<Sla [serviceProviderGroupID | applicationGroupID] ="<Group>"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="[sp_sla_file.xsd | app_sla_file.xsd]">
  <serviceTypeContract>
    <serviceTypeName></serviceTypeName>
    <startDate></startDate>
    <endDate></endDate>
    <rate></rate>
    <quota></quota>
  </serviceTypeContract>
  <serviceContract>
    <startDate></startDate>
    <endDate></endDate>
    <scs></scs>
    <contract>
    ....
    </contract>
    <overrides>
      <override>
        <startDate></startDate>
        <endDate></endDate>
        <startDow></startDow>
        <endDow></endDow>
        <startTime></startTime>
        <endTime></endTime>
        <contract>
        ...
        </contract>
      </override>
    </overrides>
  </serviceContract>
  <composedServiceContract>
    <composedServiceName></composedServiceName>
      <service>
        <serviceTypeName></serviceTypeName>
        <method>
          <scs></scs>
          <methodName></methodName>
        </method>
      </service>
      <service>
        <serviceTypeName></serviceTypeName>
      </service>
      <startDate></startDate>
      <endDate></endDate>
      <rate>
        <reqLimit></reqLimit>
        <timePeriod></timePeriod>
      </rate>
      <quota>
        <qtaLimit></qtaLimit>
        <days></days>
```

```
        <limitExceedOK></limitExceedOK>
      </quota>
  </composedServiceContract>
</Sla>
```

<Sla>

This element contains service contracts specifying the conditions under which a service provider or an application is allowed to access and use service capabilities.

The **serviceProviderGroupID** attribute specifies the service provider group to which the service provider belongs and for which group the SLA is valid. For SLAs on application level, the **applicationGroupID** attribute is used instead to specify the application group to which the application belongs and for which the SLA is valid.

The **xmlns:xsi** attribute contains processing information and should not be changed.

The **xsi:noNamespaceSchemaLocation** attribute points to the xsd for the service provider group SLA or the application group SLA, depending on which type of SLA it is:

- **app_sla_file.xsd** is the application group SLA xsd
- **sp_sla_file.xsd** is and the service provider group SLA xsd

The SLA for a a service provider group of application group can contain one or more of the following child elements:

- <serviceTypeContract>
- <serviceContract>
- <composedServiceContract>

<serviceTypeContract>

Parent: <Sla>

This element contains contractual data specifying the conditions under which a service provider or an application is allowed to access and use specific service types in Services Gatekeeper. One `<serviceTypeContract>` element is needed for each service type that a service provider or an application can access.

The data to be defined in the `<serviceTypeContract>` element for each interface is described below.

The `<serviceTypeContract>` element contains the following child elements:

- <serviceTypeName>
- <startDate>
- <endDate>
- <rate>
- <quota>

<serviceTypeName>

Parent: <serviceTypeContract>, <service>

This element specifies the service type to which a `<serviceTypeContract>` or `<service>` applies. It specifies an identifier that defines the service type. Use the Plug-in Manager's **listServiceType** operation to get a list of the service type names.

See "Managing and Configuring the Plug-in Manager" in *Services Gatekeeper System Administrator's Guide* for information on how to get a list of service types using the **listServiceType** operation.

For example, the service type for the interfaces for Parlay X 2.1 Short Messaging and RESTful Short Messaging interfaces is **Sms**.

There can be only one `<serviceTypeContract>` for a specified `<serviceTypeName>` element defined in the SLA for a group in the combined set of SLAs (the local SLA and the geo-redundant SLA). For example, if a geo-redundant service provider group SLA defines a `<serviceContract>` for the **CallNotification** service type, this service type cannot be defined in the local service provider group SLA.

Example 8–2 <serviceTypeContract> Element

```
...
<serviceTypeContract>
  <serviceTypeName>CallNotification</serviceTypeName>
  <startDate>2008-11-01</startDate>
  <endDate>2008-11-30</endDate>
  <rate>
    <reqLimit>25</reqLimit>
    <timePeriod>1000</timePeriod>
  </rate>
  <quota>
    <qtaLimit>250</qtaLimit>
    <days>1</days>
    <limitExceedOK>true</limitExceedOK>
  </quota>
</serviceTypeContract>
<serviceTypeContract>
  <serviceTypeName>Sms</serviceTypeName>
  <startDate>2008-11-01</startDate>
  <endDate>2008-11-30</endDate>
  <rate>
    <reqLimit>35</reqLimit>
    <timePeriod>1000</timePeriod>
  </rate>
  <quota>
    <qtaLimit>350</qtaLimit>
    <days>1</days>
    <limitExceedOK>true</limitExceedOK>
  </quota>
</serviceTypeContract>
...
```

<serviceContract>

Parent: <Sla>

This element contains contractual data specifying under which conditions a service provider or an application is allowed to access and use specific interfaces and methods in Services Gatekeeper. One `<serviceContract>` element is needed for each application-facing interface that a service provider or an application can access.

The `<serviceContract>` element contains the following child elements:

- <startDate>
- <endDate>
- <scs>

- <contract>
- <overrides>

The <contract> element directly following the <scs> element contains the default restrictions.

To override these default restrictions, use a <contract> inside an <override> element.

See "Structure of a Contract" for details about the child elements contained within a <contract> element.

<composedServiceContract>

Parent: <Sla>

This element is defined all the services that comprise a composed service. The <composedServiceContract> element must contain at least one <service> element.

The <composedServiceContract> element contains the following child elements:

- <composedServiceName>
- <service>
- <startDate>
- <endDate>
- <rate>
- <quota>

See "Structure of a Composed Service Contract" for information about composed services and the <service> element.

<startDate>

Parent: <serviceTypeContract>, <serviceContract>, <composedServiceContract>, <override>, <nodeContract>, <globalContract>,

For a <serviceTypeContract>, <serviceContract> or <composedServiceContact>, this element specifies the date the application can start using the service capability.

For an <override>, it specifies the first date that the contract data in the <override> element is valid.

For a <nodeContact>, it specifies the date that the service provider can begin to access the network.

For a <globalContact>, it specifies the date that Services Gatekeeper can begin to access the network on behalf of any service provider.

Use format *YYYY-MM-DD*.

A later start date on the service provider level service contract overrides this date.

<endDate>

Parent: <serviceTypeContract>, <serviceContract>, <composedServiceContract>, <override>, <nodeContract>, <globalContract>

For a <serviceTypeContract>, <serviceContract> or <composedServiceContact>, this element specifies the last date the application can use the service capability.

For an <override>, it specifies the last date that the contract data in the <override> element is valid.

For a <nodeContact>, it specifies the last date that the service provider can access the network node.

For a <globalContact>, it specifies the last date that Services Gatekeeper can access the network on behalf of any service provider.

Use format *YYYY-MM-DD*.

An earlier end date on the service provider level service contract overrides this date.

<scs>

Parent: <serviceContract>, <method>

This element specifies the application-facing interface to which the service contract or composed service contract applies. The content is a Java representation of the Web Service Definition Language (WSDL) that defines the interface.

Use the Plug-in Manager's **getServiceInfo** operation to get a list of the application-facing interfaces. You must pass the appropriate service type as a parameter to the **getServiceInfo** operation. You can obtain a list of service types with the Plug-in Manager's **listServiceTypes** operation. See section "Managing and Configuring the Plug-in Manager" in *Services Gatekeeper System Administrator's Guide* for information on using the **getServiceInfo** and **listServiceTypes** operations.

Typically, the Java representation is expressed according to the pattern below:

package_name.*standard indicator*.**plugin.***interface_name_from_WSDL*

For example, the Java representation of the interfaces for Parlay X 2.1 Short Messaging interfaces are:

- com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin
- com.bea.wlcp.wlng.px21.plugin.ReceiveSmsPlugin
- com.bea.wlcp.wlng.px21.plugin.SmsNotificationManagerPlugin

There can be only one <serviceContract> for a specified <scs> defined for a group in its combined set of SLAs (the local SLA and the geo-redundant SLA). For example, if the geo-redundant service provider group SLA defines a <serviceContract> for the **com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin** interface, this interface cannot be defined in the local service provider group SLA.

<overrides>

Parent: <serviceContract>

This element is a container of one or more <override> elements.

SLAs enforced across geo-redundant sites cannot contain an <overrides> element.

<override>

Parent: <overrides>

The contract data specified within this element overrides the default contractual data specified in the <contract> element that directly follows the <scs> element.

When an override occurs, only the restrictions specified in the <override> element are used. All restrictions specified in the default contract are disregarded if not they are not explicitly restated in the <override> element.

Each <override> element can contain the following child elements:

- <startDate>: Specifies the start date that the contract data in the <override> element is valid. If omitted, the start date for the service contract is used.
- <endDate>: Specifies the end date that the contract data in the <override> element is valid. If omitted, the end date for the service contract is used.
- <startDow>: Specifies the starting weekday for which the contract data in the <override> element is valid.
- <endDow>: Specifies the end weekday for which the contract data in the <override> element is valid. Use 1 for Sunday, 2 for Monday and so on. Optional if <startDow> is not specified.
- <startTime>: Specifies the start time for which the contract data in the <override> element is valid.
- <endTime>: Specifies the end time for which the contract data given within the <override> element is valid.
- <contract>: Specifies the contract data that overrides the default contract data specified directly after the <scs> element. See "Structure of a Contract" for details about the elements contained within a <contract> element.

For an override to be active all of the following conditions must be true:

- Today's date must be the same as or later than the start date.
- Today's date must be earlier than end date (must not be the same date).
- Current time must be between the start time and end time. If the end time is earlier than the start time, the time period spans midnight.
- Current day of week must be between start day of week and end day of week or equal to start day of week or day of week. If end day of week is less than start day of week, the time period spans the weekend.

Since several <override> elements can be defined with different settings for the time periods for which the restrictions applies, make sure the time periods do not overlap. if time periods overlap, there is no guarantee which contract applies.

Example 8–3 <overrides> Element with Two <override> Children

```
<overrides>
  <override>
    <startDate>2008-11-01</startDate>
    <endDate>2008-11-30</endDate>
    <startDow>2</startDow>
    <endDow>2</endDow>
    <startTime>09:00:00</startTime>
    <endTime>10:00:00</endTime>
    <contract>
    ...
    </contract>
  </override>
  <override>
    <startDate>2008-12-01</startDate>
    <endDate>2008-12-30</endDate>
    <startDow>2</startDow>
```

```
    <endDow>2</endDow>
    <contract>
    ...
    </contract>
  </override>
</overrides>
```

<proxyhost>

Parent: <contextAttribute>

This attribute specifies the IP address of a proxy host to use for notifications or callbacks to an application. You can use this attribute alone, and the default port 8080 is specified for notifications, or you can specify a different port using the <proxyport> attribute. See "*Adding Proxy Servers for Notifications and Callbacks*" in *Services Gatekeeper Application Developer's Guide* for more information.

This SLA fragment specifies a proxy host with the IP address of 10.161.159.185, using the port number 7999 for the OneAPI SMS plug-in.

```
<scs>oracle.ocsg.parlayrest.callback.ClientSmsNotificationCallback</scs>
         <contract>
             <requestContext>
                 <contextAttribute>
                     <attributeName>proxyhost</attributeName>
                     <attributeValue>10.161.159.185</attributeValue>
                 </contextAttribute>
                 <contextAttribute>
                     <attributeName>proxyport</attributeName>
                     <attributeValue>7999</attributeValue>
                 </contextAttribute>
             </requestContext>
         </contract>
```

<proxyport>

Parent: <contextAttribute>

This attribute specifies the port number of a proxy host to use for notifications or callbacks from an application. Must be paired with a <proxyhost> element.

<rate>

Parent: <serviceTypeContract>, <methodRestriction>, <composedServiceContract>

This element defines the maximum number of requests to be guaranteed over a short time period, measured in milliseconds.

The `<rate>` element contains the following child elements:

- <reqLimit>
- <timePeriod>

Overall performance can be adversely affected if the <reqLimit> is frequently exceeded, because requests rejected for exceeding the limit cannot be processed until the next `<timePeriod>` begins.

To minimize the chances of this type of delay occurring, define a rate with small time frames, in terms of minutes rather than hours. For example, 1000 requests per minute,

as defined in Example 8–4:

Example 8–4 Rate in Small Time Frames: Minute

```
<rate> <reqLimit>1000</reqLimit> <timePeriod>60000</timePeriod> </rate>
```

is similar in terms of throughput to 60,000 requests per hour as defined in Example 8–5.

Example 8–5 Rate in Large Time Frames: Hour

```
<rate> <reqLimit>60000</reqLimit> <timePeriod>3600000</timePeriod> </rate>
```

But for Example 8–4, the maximum number of possible rejected requests is much smaller for the one-minute <timePeriod> than for the one-hour <timePeriod> in Example 8–5. If you define a rate with a one-hour <timePeriod> and the <reqLimit> is reached in the first 10 minutes, all subsequent requests will be rejected for the rest of the hour.

The enforcement of rates set in SLAs is based on budgets that are set in the Budget service. For information about budgets, see "Managing and Configuring Budgets" in *Services Gatekeeper System Administrator's Guide*.

<reqLimit>

Parent: <rate>, <nodeRestriction>, <globalRestrictions>

This element specifies the maximum number of requests over the time period specified in the corresponding <timePeriod> element.

<timePeriod>

Parent: <rate>, <nodeRestrictions>, <globalRestrictions>

This element specifies the time period in which the corresponding <reqLimit> applies, in milliseconds.

<quota>

Parent: <serviceTypeContract>, <methodRestriction>, <composedServiceContract>

This element defines the maximum number of requests over a long period, measured in days.

The counters associated with the quota are reset at the beginning of each time period.

The <quota> element contains the following child elements:

- <qtaLimit>
- <days>
- <limitExceedOK>

<qtaLimit>

Parent: <quota>

This element defines the maximum number of requests over the time period defined in the <days> element. Only one <qtaLimit> element can be specified per <quota> element.

<days>

Parent: <quota>

This element defines the time period to which the quota limit applies, specified in days. Only integers are valid.

The starting day, (day 0), is the same day as the <startdate> element for the service contract. See <startDate>. Only one <days> element can be specified per <quota> element.

<limitExceedOK>

Parent: <quota>

This element specifies the action to take if the quota limit is exceeded. If true, the request will be allowed even if the quota is exceeded. If false, the request will be rejected. An alarm is always emitted if the limit is exceeded

<startDow>

Parent: <override>

Specifies the starting weekday for which the contract data given within the <override> element is valid. Use 1 for Sunday, 2 for Monday and so on. Optional.

<endDow>

Parent: <override>

Specifies the end weekday for which the contract data in the <override> element is valid. Use 1 for Sunday, 2 for Monday and so on. Optional if <startDow> is not specified

<startTime>

Parent: <override>

Specifies the start time for which the contract data given within the <override> element is valid. Use format *hh*:*mm*:*ss*, where *hh* can be 00–24. This entry is optional.

<endTime>

Parent: <override>

Specifies the end weekday for which the contract data in the <override> element is valid. Use 1 for Sunday, 2 for Monday and so on. This entry is optional if <startDow> is not specified.

Structure of a Contract

Example 8–6 illustrates the structure of the <contract> element in an SLA.

Example 8–6 Contract Structure

```
<contract>
  <guarantee>
    <methodGuarantee>
      <methodNameGuarantee></methodNameGuarantee>
      <reqLimitGuarantee></reqLimitGuarantee>
```

```
      <timePeriodGuarantee></timePeriodGuarantee>
    </methodGuarantee>
  </guarantee>
  <methodRestrictions>
    <methodRestriction>
      <methodName></methodName>
      <rate>
        <reqLimit></reqLimit>
        <timePeriod></timePeriod>
      </rate>
      <quota>
        <qtaLimit></qtaLimit>
        <days></days>
        <limitExceedOK></limitExceedOK>
      </quota>
    </methodRestriction>
  </methodRestrictions>
  <methodAccess>
    <blacklistedMethod>
      <methodName></methodName>
    </blacklistedMethod>
  </methodAccess>
  <requestContext>
    <contextAttribute>
      <attributeName></attributeName>
      <attributeValeu></attributeValue>
    </contextAttribute>
  </requestContext>
  <resultRestrictions>
    <resultRestriction>
      <methodName></methodName>
      <parameterRemovalName></parameterRemovalName>
      <parameterMatch>
        <parameterName></parameterName>
        <parameterValues></parameterValues>
      </parameterMatch>
      <filterMethod></filterMethod>
    </resultRestriction>
  </resultRestrictions>
</contract>
```

<contract>

Parent: <serviceContract>, <override>

This element includes all contract-specific data. There is one `<contract>` element for every `<serviceContract>` element and one for every `<override>` element.

The `<contract>` element can contain the following child elements:

- <guarantee>
- <methodRestrictions>
- <methodAccess>
- <params>
- <requestContext>
- <resultRestrictions>

<guarantee>

Parent: <contract>

This optional element specifies the number of method requests that the service provider or application is guaranteed during a specified time period. The time period is expressed in milliseconds.

Method requests from service providers and applications for which the method is tagged as guaranteed have precedence over requests from service providers and applications not having the method tagged as guaranteed.

Requests are given high priority if the request rate is below either the request rate specified in the service-provider-level SLA or the application-level SLA, whichever has the higher request rate. For example, if the service-provider-level SLA guarantees 30 requests per second and the application-level SLA guarantees 40 requests per second, the application level-SLA applies.

The `<guarantee>` element contains one or more <methodGuarantee> elements.

Each method specified by a `<methodGuarantee>` must have a corresponding `<methodRestriction>` element. The time period defined for a method must be identical in both the `<guarantee>` and `<methodRestriction>` elements.

<methodGuarantee>

Parent: <guarantee>

This element specifies a method for which high priority service is guaranteed. See <guarantee> for details.

The `<methodGuarantee>` element contains the following child elements:

- <methodNameGuarantee>
- <timePeriodGuarantee>
- <reqLimitGuarantee>

The `<methodGuarantee>` element is ignored for mobile-originated traffic.

<methodNameGuarantee>

Parent: <methodGuarantee>

This element specifies the name of the method to have guaranteed precedence.

Use the Plug-in Manager's **getServiceInfo** operation to get a list of valid method names for a service type. You must pass the appropriate service type as a parameter to the **getServiceInfo** operation.You can obtain a list of service types with the Plug-in Manager's **listServiceTypes** operation. See section "Managing and Configuring the Plug-in Manager" in *Services Gatekeeper System Administrator's Guide* for information on using the **getServiceInfo** and **listServiceTypes** operations.

<timePeriodGuarantee>

Parent: <methodGuarantee>

This element specifies the length of time for which the guarantee is applied, in milliseconds.

<reqLimitGuarantee>

Parent: <methodGuarantee>

This element specifies the number of requests to be guaranteed over the time period specified in the corresponding <timeperiodGuarantee>.

<methodRestrictions>

Parent: <guarantee>

This optional element restricts the number of method requests that an application is allowed over a specified time period. A restriction over a short time period is called a rate. A rate typically spans a few seconds. A restriction over a longer time period is called a quota. A quota typically spans several days.

The <methodRestrictions> element contains one or more <methodRestriction> elements. One <methodRestriction> element is needed for each method for which usage should be restricted. Only one <methodRestrictions> element is allowed per contract.

The <methodRestrictions> element is ignored for mobile-originated traffic.

Example 8–7 shows a sample <methodRestrictions> element with restrictions for quotas and rates. It specifies the usage restrictions for the **SendSms** and **SendSmsLogo** methods. The usage restriction for the rate is 5 requests per second (5 requests divided by 1000 milliseconds). The usage restriction for the quota is 600 requests over a 3 day period.

Example 8–7 Method Restrictions Element

```
<methodRestrictions>
  <methodRestriction>
    <methodName>sendSms</methodName>
    <rate>
      <reqLimit>5</reqLimit>
      <timePeriod>1000</timePeriod>
    </rate>
    <quota>
      <qtaLimit>600</qtaLimit>
      <days>3</days>
      <limitExceedOK>true</limitExceedOK>
    </quota>
  </methodRestriction>
  <methodRestriction>
    <methodName>sendSmsLogo</methodName>
    <rate>
      <reqLimit>5</reqLimit>
      <timePeriod>1000</timePeriod>
    </rate>
    <quota>
      <qtaLimit>600</qtaLimit>
      <days>3</days>
      <limitExceedOK>true</limitExceedOK>
    </quota>
  </methodRestriction>
</methodRestrictions>
```

The most restrictive limit is always enforced, so if a restriction in a service-provider-level SLA is more restrictive than a restriction defined in an application-level SLA, the service-provider-level SLA is enforced.

If the application does not have any usage restrictions within the allowed methods, omit the <methodRestrictions> element.

<methodRestriction>

Parent: <methodRestrictions>

This element specifies restrictions on the usage of a method in the application-facing interface.

The <methodRestriction> element contains the following child elements:

- <methodName>
- <rate>
- <quota>

<methodName>

Parent: <methodRestriction>, <blacklistedMethod>, <methodParameters>, <resultRestriction>, <method>

This element specifies the name of the method to restrict.

Use the Plug-in Manager's **getServiceInfo** operation to get a list of valid method names for a service type. You must pass the appropriate service type as a parameter to the **getServiceInfo** operation.You can obtain a list of service types with the Plug-in Manager's **listServiceTypes** operation. See section "Managing and Configuring the Plug-in Manager" in *Services Gatekeeper System Administrator's Guide* for information on using the **getServiceInfo** and **listServiceTypes** operations.

<methodAccess>

Parent: <contract>

This optional element blocks the application from accessing one or more methods in the service capability.

If the application is allowed to access all methods, omit the <methodAccess> element.

The <methodAccess> element contains the <blacklistedMethod> child element.

<blacklistedMethod>

Parent: <methodAccess>

This element contains one <methodName> child element.

It prohibits use of the method specified in its <methodName> child element.

Use a separate <blacklistedMethod> element for each method to block. Example 8–8 blocks Native SMPP applications from sending **SubmitMulti** messages:

Example 8–8 A blacklistedMethod Element

```
<serviceContract>
    <startDate>2015-07-22</startDate>
    <endDate>9999-12-31</endDate>
    <scs>oracle.ocsg.protocol.smpp.plugin.SMPPPluginNorth</scs>
    <contract>
       <methodAccess>
           <blacklistedMethod>
```

```
                <methodName>submitMulti</methodName>
              </blacklistedMethod>
          </methodAccess>
     </contract>
</serviceContract>
```

<params>

Parent: <contract>

This optional element is used both to allow and to prohibit certain parameters values provided by applications.

The <params> element contains zero (0) or more <methodParameters> elements.

Example 8–9 shows a <params> element that allows allowed parameter values A, B, and C for the **sendMessage** method. No other values for this parameter are allowed.

Example 8–9 <params> Element

```
<params>
  <methodParameters>
    <methodName>sendMessage</methodName>
    <parameterName>arg0.subject</parameterName>
    <parameterValues>A B C</parameterValues>
    <acceptValues>true</acceptValues>
  </methodParameters>
</params>
```

<methodParameters>

Parent: <params>

This optional element specifies method parameter values to allow or prohibit.

It contains the following child elements:

- <methodName>
- <parameterName>
- <parameterValues> simple
- <acceptValues>

<parameterName>

Parent: <methodParameters>, <parameterMatch>

This element specifies the name of the parameter whose values are to be allowed or prohibited. The content of this element is the Java representation of the parameter defined in the WSDL. The parameter representation is **arg0.***name_of_parameter_as_defined_in_WSDL*.

You can use the Plug-in Manager's **getServiceInfo** operation to get a list of valid parameter names for the method. You must pass the appropriate service type as a parameter to the **getServiceInfo** operation.You can obtain a list of service types with the Plug-in Manager's **listServiceTypes** operation. See section "Managing and Configuring the Plug-in Manager" in *Services Gatekeeper System Administrator's Guide* for information on using the **getServiceInfo** and **listServiceTypes** operations.

Example 8–10 shows an example of the parameters that **getServiceInfo** returns for the **sendMessage** method for the MultimediaMessaging service type.

Example 8–10 Sample Output from getServiceInfo (type: MultimediaMessaging) for SendMessage

```
Interface: com.bea.wlcp.wlng.px21.plugin.SendMessagePlugin
  Method: sendMessage
    Arguments:
     java.net.URI arg0.addresses[]
     java.math.BigDecimal arg0.charging.amount
     java.lang.String arg0.charging.code
     java.lang.String arg0.charging.currency
     java.lang.String arg0.charging.description
     java.lang.String arg0.priority.value
     java.lang.String arg0.receiptRequest.correlator
     java.net.URI arg0.receiptRequest.endpoint
     java.lang.String arg0.receiptRequest.interfaceName
     java.lang.String arg0.senderAddress
     java.lang.String arg0.subject
    Return arguments:
     java.lang.String arg0.result
```

If the parameter is an array (for example, arg0.addresses[]), do not use the brackets in the SLA. For the example, the parameter name in the SLA should be:

```
<parameterName>arg0.addresses</parameterName>.
```

<parameterValues> simple

Parent: <methodParameters>

This element contains a list of parameter values to allow or prohibit, when contained in a `<methodParameters>` element. Multiple values can be defined, separated by white space.

If the SLA is regulating the Binary SMS Communication Service, UDH values must be specified in decimal, not hexadecimal, format. In the following example, any UDH with the values of 04, 05, 44, or 7F will be blocked:

```
<methodParameters>
        <methodName>sendBinarySms</methodName>
        <parameterName>arg0.binaryMessag[0].udh</parameterName>
        <parameterValues>4 5 68 127</parameterValues>
        <acceptValues>false</acceptValues>
</methodParameters>
```

<parameterValues> complex

Parent: <parameterMatch>

This element contains one or more <parameterValue> elements.

<parameterValue>

Parent:<parameterValues> complex

Each `<parameterValue>` element contains the value of the `<parameterName>` to match if this element is the grandchild of a <parameterMatch> element. The value can be expressed as a regular expression.

<acceptValues>

Parent: <methodParameters>

This element specifies whether the parameter values controlled by the containing <params> element are allowed or prohibited.

If <acceptValues> is true, the parameter values specified in the <parameterValues> element are allowed and all other values are prohibited. If false, the specified parameter values are prohibited and all other values are allowed

<requestContext>

Parent: <contract>

This optional element sends additional parameters, which may not be defined in the interface, to a plug-in. The concept is similar to parameter tunneling, except that the attribute-value pairs are defined in the SLA instead of being passed by the application. For more information see "Parameter Tunneling" in *Services Gatekeeper Extension Developer's Guide*.

The <requestContext> element contains one or more <contextAttribute> elements.

Example 8–11 shows a <requestContext> element that assigns values to two context attributes.

Example 8–11 <requestContext> Example

```
<requestContext>
  <contextAttribute>
    <attributeName>com.bea.wlcp.wlng.plugin.sms.testName1</attributeName>
    <attributeValue>testValue1</attributeValue>
  </contextAttribute>
  <contextAttribute>
    <attributeName>com.bea.wlcp.wlng.plugin.sms.testName2</attributeName>
    <attributeValue>testValue2</attributeValue>
  </contextAttribute>
</requestContext>
```

<contextAttribute>

Parent: <requestContext>

A <contextAttribute> element defines the attribute-value pairs to be sent to the plug-in.

This element contains the following one of each of the following child elements:

- <attributeName>
- <attributeValue>

A plug-in can retrieve the value specified in <attributeValue> using the name specified in <attributeName>. The plug-in must implement the functionality for fetching the value by name. For more information, see "Using Request Context Parameters in SLAs" in *Services Gatekeeper Extension Developer's Guide*.

See the "Tunneled Parameters" sections in *Services Gatekeeper Communication Service Reference Guide* for descriptions of the context attributes that are applicable to your communication service. Note that not all communication services support tunneled parameters and context attributes.

<attributeName>

Parent: <contextAttribute>

This element contains the name that the plug-in uses to fetch the attribute.

<attributeValue>

Parent: <contextAttribute>

This element contains the value associated with the corresponding `<attributeName>` in the `<contextAttribute>` element.

<resultRestrictions>

Parent: <contract>

This optional element filters results returned from application-initiated requests.

This element contains one or more `<resultRestriction>` elements.

Result Restrictions Example

Example 8–12 illustrates a results filter for the getData() method.

The getData() method returns an array of data, where data is a complex type consisting of the name-value pair:

- dataName
- dataValue

Example 8–12 <resultRestrictions> Element for Blacklisting Array Parameter Values

```
<resultRestrictions>
  <resultRestriction>
    <methodName>getData</methodName>
    <parameterRemovalName>result.data[].dataName</parameterRemovalName>
    <parameterMatch>
      <parameterName>result.data[].dataName</parameterName>
      <parameterValues>
        <parameterValue>ssn<parameterValue>
        <parameterValue>homephone<parameterValue>
      <parameterValues>
    </parameterMatch>
    <filterMethod>BLACK_LIST</filterMethod>
  </resultRestriction>
<resultRestrictions>
```

Assuming the result before the filter is applied is:

- result.data[0].dataName = “cellphone”
- result.data[0].dataValue =”415-555-1234”
- result.data[1].dataName = “ssn”
- result.data[1].dataValue =” 123 45 6789”
- result.data[2].dataName = “homephone”
- result.data[2].dataValue =”415-333-4444”

After applying the result filter, the result is:

- result.data[0].dataName = “cellphone”
- result.data[0].dataValue =”415-555-1234”

because <filterMethod> is **BLACK_LIST** and <parameterValues> filters out **ssn** and **homephone**. Those data items are removed from the result returned to the application.

If <filterMethod> had been **WHITE_LIST**, the result after applying the filter would be:

- result.data[0].dataName = “ssn”
- result.data[0].dataValue =”123 45 6789”
- result.data[0].dataName = “homephone”
- result.data[0].dataValue =”415-333-4444”

Only these parameters would match the filter that filters in the values in <parameterValues>.

If <parameterRemovalName> had been **result.data** instead of **result.data[]** the filtered result would be null in both cases, since both filters match. The absence of the **[]** means that all results starting from the **result.data** hierarchy should be removed, given that the filter matches the result.

<resultRestriction>

Parent: <resultRestrictions>

This element defines a result restriction.

The result restriction makes it possible to remove parameters returned from application-initiated requests.

The return parameter to which the restriction applies is expressed as a String representation of a Java class. Arrays are expressed using the [] notation directly following the parameter name. The notation format is:

- **arg0.**result.*name_of_parameter_as_defined_in_WSDL* when the parameter is a single entity.
- **arg0.**result[].*name_of_parameter_as_defined_in_WSDL* when the parameter is an array.

If the return parameter is a complex type, the hierarchy is expressed using dot notation. For example, if an array for the **dateTime** parameter is returned, the return value expressed in XML is:

```
<xsd:complexType name="SmsMessage">
  <xsd:sequence>
   <xsd:element name="message" type="xsd:string"/>
   <xsd:element name="senderAddress" type="xsd:anyURI"/>
   <xsd:element name="smsServiceActivationNumber" type="xsd:anyURI"/>
   <xsd:element name="dateTime" type="xsd:dateTime" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

The String representation is:

```
arg0.result[].dateTime.firstDayOfWeek
arg0.result[].dateTime.lenient
arg0.result[].dateTime.minimalDaysInFirstWeek
arg0.result[].dateTime.time.date
arg0.result[].dateTime.time.hours
```

```
arg0.result[].dateTime.time.minutes
arg0.result[].dateTime.time.month
arg0.result[].dateTime.time.seconds
arg0.result[].dateTime.time.time
arg0.result[].dateTime.time.year
arg0.result[].dateTime.timeInMillis
arg0.result[].dateTime.timeZone.ID
arg0.result[].dateTime.timeZone.rawOffset
arg0.result[].message
arg0.result[].senderAddress
arg0.result[].smsServiceActivationNumber
```

The `<resultRestriction>` element contains the following child elements:

- <methodName>: method for which to restrict the return parameter.
- <parameterRemovalName>
- <parameterMatch>: optional filter value to match
- <filterMethod>: defines how the filter is applied

<parameterRemovalName>

Parent: <resultRestriction>

The `<parameterRemovalName>` element defines which part of a complex return parameter is affected by the restriction.

It is possible to specify a leaf or a node in the hierarchy of the parameter. When a node is specified, the node and all its siblings are removed.

If the <parameterRemovalName> element is:

```
<parameterRemovalName>arg0.result[].dateTime.timeZone.ID</parameterRemovalName>
```

only the **timeZone.ID** part of the result parameter is removed, because the element specifies a leaf.

If the <parameterRemovalName> element is:

```
<parameterRemovalName>arg0.result[].dateTime.time</parameterRemovalName>
```

all of the **dateTime** siblings are removed from the result parameters, because the element specifies a node. In this case, the following values would be removed:

- arg0.result[].dateTime.time.date
- arg0.result[].dateTime.time.hours
- arg0.result[].dateTime.time.minutes
- arg0.result[].dateTime.time.month
- arg0.result[].dateTime.time.seconds
- arg0.result[].dateTime.time.time
- arg0.result[].dateTime.time.year

If the specified parameter is a Boolean, the parameter is not removed but set to `false`.

For information on how to get a list of valid parameter names for a method with a specific service type, follow the instructions for <methodName>.

<parameterMatch>

Parent:<resultRestriction>

This element is optional.

If <parameterMatch> is used, result filtering occurs only if the parameter defined in the <parameterName> child element has a value defined in one of its <parameterValue> child elements.

The <parameterMatch> element contains the following child elements:

- <parameterName>: exactly one
- <parameterValues> complex: exactly one

<filterMethod>

Parent: <resultRestriction>

This element defines how the filter defined in <parameterMatch> is applied.

The value is an enumeration. Valid values are:

- BLACK_LIST
- WHITE_LIST

If <filterMethod> is defined as **BLACK_LIST**, all parameters matching <parameterRemovalName> are removed if the request matches the filter.

If <filterMethod> is defined as **WHITE_LIST**, only the parameters matching <parameterRemovalName> are kept if the request matches the filter.

Structure of a Composed Service Contract

A composed service contract is enforced for a composed service.

A composed service is created by combining multiple communication services, all of which must be available and registered in Services Gatekeeper. You can then define and apply enforcements on the composed service, instead of defining identical enforcements separately for each service. A request of any one of the communication services that participate in the composed service is treated as a request of the composed service for the purposes of SLA enforcement.

A single SLA can contain composed service contracts as well as simple service contracts. A composed service contract is located at the same level in the SLA hierarchy as a service contract.

You can configure and test a composed service contract using the Platform Test Environment. See the discussion of composed service-level agreements in the *Services Gatekeeper Platform Test Environment User's Guide* for more information.

Composed Service Contracts

You can define a composed service contract in an application group SLA or in a service provider group SLA.

The composed service contract is created when an SLA containing a <composedServiceContract> element is loaded using the loadApplicationGroupSlaByType or loadServiceProviderGroupSlaByType operations to **ServiceLevelAgreementMBean**. See the "All Classes" section of *Services Gatekeeper OAM Java API Reference* for details.

Scope

A composed service contract includes all the service types of its constituent communication services and all the request rate and quota restrictions that may be applied to those services under an individual service type contract.

A composed service contract can optionally be defined at the method level of granularity so that enforcement is implemented on only specified methods. If individual methods are not specified for a specific service in the composed service contract, enforcement is applied for all methods of the specified service.

Multiple Composed Services

A single SLA can contain multiple composed service contracts.

A single communication service can participate as a component in multiple composed service contracts. For example, one composed service named "Messaging" could be composed of the Short Messaging and Multimedia Messaging communication services, while another composed service named "LocationNotification", defined in the same SLA, could be composed of the Short Messaging and Terminal Location communication services. In this case, the shared Short Messaging service would be subject to the enforcements of both composed service contracts.

Conflicting Enforcements

It is possible to specify different enforcements that affect a single service within a single SLA. If the enforcements in the multiple contracts are different, the more restrictive enforcement is applied to requests of the shared service. But because separate counters are maintained for each service within a composed service, requests of the non-shared service are not counted towards the limit of the shared service.

For example, imagine an SLA that contains the following contracts, all of which include the Sms service:

1. a service contract for the Sms service that allows 40 requests per second
2. a Messaging composed service contract, composed of the Sms service and the Multimedia Messaging service, that allows 50 requests per second
3. a LocationNotification composed service contract, composed of the Sms service and the Terminal Location service, that allows 60 requests per second

The maximum number of requests to the Sms service will be 40 requests per second. But a request of one of the non-shared services, for example of the Multimedia messaging or Terminal Location service, does not count against this 40-request limit on the Sms service.

Budget Implications

All communication services in a composed service share the same budget restrictions in the composed service contract. Any request of the composed service triggers a decrease in the budget. If a request exceeds the budget restriction, subsequent requests of the composed service are denied. The result is that if one of the services in the composed SLA causes the budget of the composed SLA to be exceeded, requests from the other constituent services are denied.

Example Composed Service SLA

Example 8–13 shows an SLA that contains a simple service contract for the Short Messaging service and a composed service contract for a composed service named "Messaging."

The Messaging composed service includes the **sendSMS** and **sendSmsLogo** operations in the ParlayX2.1 Short Messaging communication service and all of the operations that Services Gatekeeper supports in the ParlayX2.1 Multimedia Messaging communication service.

Example 8–13 Service Level Agreement with Composed Service Contract

```
<Sla xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
serviceProviderGroupID="default_sp_group" >
    <serviceContract>
        <startDate>2010-04-17</startDate>
        <endDate>2011-04-17</endDate>
        <scs>com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin</scs>
    </serviceContract>
    <composedServiceContract>
        <composedServiceName>Messaging</composedServiceName>
        <service>
            <serviceTypeName>Sms</serviceTypeName>
            <method>
                 <scs>com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin</scs>
                 <methodName>sendSMS</methodName>
            </method>
            <method>
                 <scs>com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin</scs>
                 <methodName>sendSmsLogo</methodName>
             </method>
        </service>
        <service>
            <serviceTypeName>MultiMediaMessage</serviceTypeName>
        </service>
        <startDate>2010-04-17</startDate>
        <endDate>2011-04-17</endDate>
        <rate>
            <reqLimit>50</reqLimit>
            <timePeriod>50</timePeriod>
        </rate>
        <quota>
            <qtaLimit>100</qtaLimit>
            <days>1</days>
            <limitExceedOK>false</limitExceedOK>
        </quota>
     </composedServiceContract>
</Sla>
```

<composedServiceName>

Parent: <composedServiceContract>

This element describes the unique, user-defined name for the composed service.

<service>

Parent: <composedServiceContract>

This element defines a constituent communication service in the composed service. The service must be available and registered in Services Gatekeeper.

There must be at least one `<service>` element in a `<composedServiceContract>` element. Normally there are multiple `<service>` elements.

The `<service>` element contains the following child elements:

- <serviceTypeName>
- <method>

The service is identified by the <serviceTypeName> child element of the <service> element. The <serviceTypeName> element is required.

The <method> child element of the <service> element is optional.

<method>

Parent: <service>

This optional element, if used, specifies a method of the service to which the composed service contract's enforcements are applied. There can be multiple <method> child elements of a single <service> element.

If at least one <method> element is specified, only the specified method or methods participate in the composed service.

If there are no <method> elements contained in a <service> element, enforcement is applied to all of the service's methods.

The <method> element contains the following child elements:

- <scs>
- <methodName>

9

Defining Global Node and Service Provider Group Node SLAs

This chapter describes node SLAs, which define access to underlying network elements in Oracle Communications Services Gatekeeper.

Node SLAs are different from the SLAs that define applications' use of Oracle Communications Services Gatekeeper described in "Defining Service Provider Group and Application Group SLAs".

There are two kinds of node SLAs:

- Service provider group node SLAs define a specific service provider's access to the network.
- Global node SLAs define Services Gatekeepers's access to the network regardless of service provider.

Structure of a Node Service Level Agreement

The node SLAs are written in XML. Global node SLAs and service provider group node SLAs are slightly different.

The schema for the global node SLA is in *Middleware_Home*/**ocsg_6.0/modules/com.bea.wlcp.wlng.account_6.0.0.0.jar: sla_schema/global_node_sla_file.xsd**.

The schema for the service provider group node SLA is in *Middleware_Home*/**ocsg_6.0/modules/com.bea.wlcp.wlng.account_6.0.0.0.jar: sla_schema/sp_node_sla_file.xsd**.

<Sla>

When used to define a node SLA, the `<Sla>` element contains one of the following:

- zero or more <nodeContract> elements for a service provider group node SLA
- zero or more <globalContract> elements for a global node SLA

The `<nodeContract>` and `<globalContract>` elements are mutually exclusive in one SLA instance. You cannot combine `<nodeContract>` and `<globalContract>` elements in the same `<Sla>`.

If the SLA is a service provider group node SLA, the **serviceProviderGroupID** attribute specifies the service provider group for which the SLA is valid.

If the SLA is a global node SLA, omit the **serviceProviderGroupID** attribute.

Service Provider Group Node SLA

The service provider group node SLA consists of an <Sla> element containing zero (0) or more <nodeContract> elements.

The **serviceProviderGroupID** attribute in the `<Sla>` element specifies the service provider group for which the SLA file is valid.

Example 9–1 shows the structure of a service provider group node SLA.

Example 9–1 Service Provider Group Node SLA Structure

```
<Sla serviceProviderGroupID="spGroup1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="sp_node_sla_file.xsd">
<nodeContract>
    <!--Contract data for network node 1-->
  </nodeContract>
<nodeContract>
    <!--Contract data for network node 2-->
  </nodeContract>
<nodeContract>
    <!--Contract data for network node n-->
  </nodeContract>
</Sla>
```

<nodeContract>

Parent: <Sla>

The `<nodeContract>` element defines under which conditions Services Gatekeeper can access one or more network nodes on behalf of the service provider group specified in the `<Sla>` element.

The `<nodeContract>` element contains the following child elements:

- <startDate>: one (1)

  Specifies the date the service provider can start accessing the network node.

- <endDate> one (1)

  Specifies the last date that the service provider can accessing the network node.

- <nodeID>: one (1)
- <nodeRestrictions>: zero (0) or one (1)

Example 9–2 shows a `<nodeContract>` that limits access to node A to 10 requests per second (10 requests divided by 1000 milliseconds).

Example 9–2 <nodeContract> Element

```
<nodeContract>
  <startDate>2005-01-01</startDate>
  <endDate>2010-06-01</endDate>
  <nodeID>A</nodeID>
  <nodeRestrictions>
    <nodeRestriction>
      <reqLimit>10</reqLimit>
      <timePeriod>1000</timePeriod>
    </nodeRestriction>
  </nodeRestrictions>
```

```
</nodeContract>
```

<nodeID>

Parent: <nodeContract>

The <nodeID> element specifies the network node ID of the node for which a <nodeContract> controls access. The node ID is registered in the Plug-in Manager. A node ID can be assigned to one or more network nodes and the contract can therefore be valid for one or more nodes.

Use the Plug-in Manager's **getPluginNodeId** operation to get the node ID. See "Managing and Configuring the Plug-in Manager" in *Services Gatekeeper System Administrator's Guide* for information the **getPluginNodeId** operation.

<nodeRestrictions>

Parent: <nodeContract>

The <nodeRestrictions> element contains zero or one <nodeRestriction> elements.

<nodeRestriction>

Parent: <nodeRestrictions>

The <nodeRestriction> element defines the restrictions imposed by the <nodeContract> in a service provider group node SLA.

This element contains the following child elements:

- <reqLimit>: zero (0) or one (1)
- <timePeriod>: zero (0) or one (1)

Global Node SLA

The global node SLA consists of an <Sla> element containing zero (0) or more <globalContract> elements.

For a global node SLA, there should be no **serviceProviderGroupID** attribute in the <Sla> element.

Example 9–3 shows the structure of a global node SLA.

Example 9–3 Global Node SLA Structure

```
<Sla xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="global_node_sla_file.xsd">
<globalContract>
    <!--Contract data for network node 1-->
  </globalContract>
<globalContract>
    <!--Contract data for network node 2-->
  </globalContract>
<globalContract>
    <!--Contract data for network node n-->
  </globalContract>
</Sla>
```

<globalContract>

Parent: <Sla>

The <globalContract> element defines the conditions under which Services Gatekeeper can access one or more network nodes, regardless of which service provider sent the request.

The <globalContract> element contains the following child elements

- <startDate>: one (1)

  Specifies the date that Services Gatekeeper can start accessing the network node.

- <endDate>: one (1)

  Specifies the last date that Services Gatekeeper can access the network node.

- <nodeID>: one (1)

- <<globalRestrictions>: zero (0) or one(1)

Example 9–4 shows a <globalContract> that limits Services Gatekeeper's access to node A to 10 requests per second (10 requests divided by 1000 milliseconds). Normal priority requests are passed to the node until the maximum request rate is reached.

Example 9–4 <globalContract> Element

```
<globalContract>
  <startDate>2005-01-01</startDate>
  <endDate>2010-06-01</endDate>
  <nodeID>A</nodeID>
  <globalRestrictions>
    <globalRestriction>
      <reqLimit>1000</reqLimit>
      <timePeriod>10000</timePeriod>
      <guaranteePercentage>50</guaranteePercentage>
    </globalRestriction>
  </globalRestrictions>
</globalContract>
```

<globalRestrictions>

Parent: <globalContract>

The <globalRestrictions> element contains zero or one <globalRestriction> elements.

<globalRestriction>

Parent: <globalRestrictions>

The <globalRestriction> element defines the restrictions imposed by the <globalContract> in a global node SLA.

This element contains the following child elements:

- <reqLimit>: zero (0) or one (1)
- <timePeriod>: zero (0) or one (1)
- <guaranteePercentage>: zero (0) or one (1)

<guaranteePercentage>

Parent: <globalRestriction>

The <guaranteePercentage> element is used in a <globalRestriction> element in a global node contract.

It specifies the relative priority between requests marked as guaranteed (high priority) and normal priority requests. The integer value represents a percentage.

If set to 0, guaranteed requests and normal priority requests are treated equally, no matter whether they are guaranteed or normal priority. The value of 0 makes the priority of the requests irrelevant.

If set to 100, only guaranteed requests are passed on to the network node. No normal priority requests are allowed.

If set to 50, normal priority requests are passed on to the network node up to the point where the maximum request rate is reached. After that point, requests with normal priority are rejected.

Other values can be used to fine-tune how likely it is that requests of different priority are allowed. A higher value makes is more likely that normal priority requests will be rejected as traffic rate increases. A lower value makes it less likely that normal priority requests will be rejected.

The default value is 50.

A

Sample SLA XML File

The following is a complete sample Service Level Agreement (SLA) XML file for the Parlay X 2.1 Short Messaging Communication service. It contains a <serviceContract>, a <serviceTypeContract>, and a <composedServiceContract>.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Sla applicationGroupID="default_app_group"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="app_sla_file.xsd">
    <serviceContract>
        <startDate>2005-07-22</startDate>
        <endDate>9999-12-31</endDate>
        <scs>com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin</scs>
        <contract>
            <guarantee>
                <methodGuarantee>
                    <methodNameGuarantee>sendSmsLogo</methodNameGuarantee>
                    <reqLimitGuarantee>10000</reqLimitGuarantee>
                    <timePeriodGuarantee>80000</timePeriodGuarantee>
                </methodGuarantee>
            </guarantee>
            <methodRestrictions>
                <methodRestriction>
                    <methodName>sendSms</methodName>
                    <rate>
                        <reqLimit>9000</reqLimit>
                        <timePeriod>1000</timePeriod>
                    </rate>
                    <quota>
                        <qtaLimit>900000</qtaLimit>
                        <days>1</days>
                        <limitExceedOK>false</limitExceedOK>
                    </quota>
                </methodRestriction>
            </methodRestrictions>
            <params>
                <methodParameters>
                    <methodName>sendSms</methodName>
                    <parameterName>arg0.message</parameterName>
                    <parameterValues>foo</parameterValues>
                    <acceptValues>false</acceptValues>
                </methodParameters>
            </params>
            <methodAccess>
                <blacklistedMethod>
                    <methodName>sendSmsLogo</methodName>
                </blacklistedMethod>
```

```
    </methodAccess>
    <requestContext>
        <contextAttribute>
            <attributeName>key1</attributeName>
            <attributeValue>value1</attributeValue>
        </contextAttribute>
    </requestContext>
    <resultRestrictions>
        <resultRestriction>
            <methodName>getSmsDeliveryStatus</methodName>
           <parameterRemovalName>arg0.requestIdentifier</parameterRemovalName>
            <parameterMatch>
                <parameterName>arg0.requestIdentifier</parameterName>
                <parameterValues>
                    <parameterValue>demo</parameterValue>
                </parameterValues>
            </parameterMatch>
            <filterMethod>BLACK_LIST</filterMethod>
        </resultRestriction>
    </resultRestrictions>
</contract>
<overrides>
    <override>
        <startDate>2010-11-30</startDate>
        <endDate>2012-11-30</endDate>
        <startDow>2</startDow>
        <endDow>6</endDow>
        <contract>
            <guarantee>
                <methodGuarantee>
                    <methodNameGuarantee>sendSms</methodNameGuarantee>
                    <reqLimitGuarantee>1000</reqLimitGuarantee>
                    <timePeriodGuarantee>40000</timePeriodGuarantee>
                </methodGuarantee>
            </guarantee>
            <methodRestrictions>
                <methodRestriction>
                    <methodName>sendSms</methodName>
                    <rate>
                        <reqLimit>500</reqLimit>
                        <timePeriod>1000</timePeriod>
                    </rate>
                    <quota>
                        <qtaLimit>10000</qtaLimit>
                        <days>1</days>
                        <limitExceedOK>false</limitExceedOK>
                    </quota>
                </methodRestriction>
            </methodRestrictions>
            <params>
                <methodParameters>
                    <methodName>sendSms</methodName>
                    <parameterName>arg0.message</parameterName>
                    <parameterValues>foo2</parameterValues>
                    <acceptValues>false</acceptValues>
                </methodParameters>
            </params>
            <methodAccess>
                <blacklistedMethod>
                    <methodName>sendSmsLogo</methodName>
```

```
                            </blacklistedMethod>
                            <blacklistedMethod>
                                <methodName>sendSmsRingtone</methodName>
                            </blacklistedMethod>
                        </methodAccess>
                        <requestContext>
                            <contextAttribute>
                                <attributeName>key2</attributeName>
                                <attributeValue>value2</attributeValue>
                            </contextAttribute>
                        </requestContext>
                        <resultRestrictions>
                            <resultRestriction>
                                <methodName>getSmsDeliveryStatus</methodName>

<parameterRemovalName>arg0.requestIdentifier</parameterRemovalName>
                                <parameterMatch>
                                    <parameterName/>
                                    <parameterValues>
                                        <parameterValue>22</parameterValue>
                                        <parameterValue>33</parameterValue>
                                    </parameterValues>
                                </parameterMatch>
                                <filterMethod>WHITE_LIST</filterMethod>
                            </resultRestriction>
                        </resultRestrictions>
                    </contract>
                </override>
            </overrides>
        </serviceContract>
        <serviceTypeContract>
            <serviceTypeName>Sms</serviceTypeName>
            <startDate>2010-11-30</startDate>
            <endDate>2010-11-30</endDate>
            <rate>
                <reqLimit>1000</reqLimit>
                <timePeriod>1000</timePeriod>
            </rate>
            <quota>
                <qtaLimit>90000</qtaLimit>
                <days>1</days>
                <limitExceedOK>false</limitExceedOK>
            </quota>
        </serviceTypeContract>
        <composedServiceContract>
            <composedServiceName>Messaging</composedServiceName>
            <service>
                <serviceTypeName>Sms</serviceTypeName>
                <method>
                    <scs>com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin</scs>
                    <methodName>sendSMS</methodName>
                </method>
             </service>
            <service>
                <serviceTypeName>MultiMediaMessage</serviceTypeName>
            </service>
            <startDate>2010-04-17</startDate>
            <endDate>2011-04-17</endDate>
            <rate>
                <reqLimit>50</reqLimit>
```

```
            <timePeriod>50</timePeriod>
        </rate>
        <quota>
            <qtaLimit>100</qtaLimit>
            <days>1</days>
            <limitExceedOK>false</limitExceedOK>
        </quota>
    </composedServiceContract>
</Sla>
```