FUJITSU

FUJITSU Software Enterprise Service Catalog Manager V17.5.0

Developer's Guide

December 2017 - Initial draft

Contents

About this Manual

This manual describes the public Web services and application programming interfaces (APIs) of) FUJITSU Software Enterprise Service Catalog Manager, hereafter referred to as ESCM, and how to integrate applications and external systems with ESCM.

The manual is structured as follows:

Chapter	Description
Introduction on page 8	Provides an overview of a developer's tasks, and describes the basic concepts of Web services.
Platform Services and Web Service APIs on page 11	Describes the ESCM platform services and public APIs, as well as their purpose and usage.
Integrating Applications with ESCM on page 18	Describes how to implement the Web service interfaces between an application and ESCM.
Integrating External Process Control on page 32	Describes how to integrate an external process control system with ESCM.
Integrating an External Parameter Configuration Tool on page 34	Describes how to integrate an external parameter configuration tool with ESCM.
Integrating Certificates for Trusted Communication on page 44	Provides an introduction to the usage of certificates for securing communication between ESCM applications or external systems.
Customer Billing Data on page 48	Describes the elements of an XML file created by exporting customer billing data.
Revenue Share Data on page 65	Describes the elements of XML files created by exporting revenue share data.

Readers of this Manual

This manual is directed to developers who carry out tasks for different organizations involved in the usage of ESCM, for example, integrate applications or external process control systems with ESCM.

This manual assumes that you are familiar with the following:

- ESCM concepts as explained in the *Overview* manual
- Basic Web service concepts
- XML and the XSD language
- Web services standards and concepts, SOAP, WSDL, REST, and JSON
- A programming language that can be used to create and invoke Web services, for example, Java
- Java, Java servlets, and Java server pages
- Installation and basic administration of Web servers

Notational Conventions

This manual uses the following notational conventions:

Add	Names of graphical user interface elements.
`init`	System names, for example command names and text that is entered from the keyboard.
`<variable>`	Variables for which values must be entered.
`[option]`	Optional items, for example optional command parameters.
`one | two`	Alternative entries.
`{one | two}`	Mandatory entries with alternatives.

Abbreviations

This manual uses the following abbreviations:

API	Application programming interface
APP	Asynchronous Provisioning Platform
ESCM	Enterprise Service Catalog Manager
CA	Certification authority
IaaS	Infrastructure as a Service
JAX-WS	Java API for XML Web services
JSP	Java Server Pages
PaaS	Platform as a Service
PSP	Payment service provider
REST	Representational State Transfer
RPC	Remote Procedure Calls
SaaS	Software as a Service
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
XSD	XML Schema Definition

Available Documentation

The following documentation on ESCM is available:

- *Overview:* A PDF manual introducing ESCM. It is written for everybody interested in ESCM and does not require any special knowledge.
- *Online Help:* Online help pages describing how to work with the administration portal of ESCM. The online help is intended for and available to everybody working with the administration portal.

- *Operator's Guide:* A PDF manual for operators describing how to administrate and maintain ESCM.
- *Technology Provider's Guide:* A PDF manual for technology providers describing how to prepare applications for usage in a SaaS model and how to integrate them with ESCM.
- *Supplier's Guide:* A PDF manual for suppliers describing how to define and manage service offerings for applications that have been integrated with ESCM.
- *Reseller's Guide:* A PDF manual for resellers describing how to prepare, offer, and sell services defined by suppliers.
- *Broker's Guide:* A PDF manual for brokers describing how to support suppliers in establishing relationships to customers by offering their services on a marketplace.
- *Marketplace Owner's Guide:* A PDF manual for marketplace owners describing how to administrate and customize marketplaces in ESCM.
- *Developer's Guide:* A PDF manual for application developers describing the public Web services and application programming interfaces of ESCM and how to integrate applications and external systems with ESCM.
- *Amazon Web Services Integration:* A PDF manual for operators describing how to offer and use virtual servers controlled by the Amazon Elastic Compute Cloud Web service through services in ESCM.
- *OpenStack Integration:* A PDF manual for operators describing how to offer and use virtual systems controlled by OpenStack through services in ESCM.
- Javadoc and YAML documentation for the public Web services and application programming interfaces of ESCM and additional resources and utilities for application developers.

1 Introduction

Enterprise Service Catalog Manager (ESCM) is a set of services which provide all business-related functions and features required for turning on-premise applications and tools into "as a Service" (aaS) offerings and using them in the Cloud. This includes ready-to-use account and subscription management, online service provisioning, billing and payment services, and reporting facilities.

With its components, ESCM supports software vendors as well as their customers in leveraging the advantages of Cloud Computing.

The basic scenario of deploying and using applications as services in the ESCM framework involves the following users and organizations:

- **Technology providers** (e.g. independent software vendors) technically prepare their applications for usage in the Cloud and integrate them with ESCM. They register the applications as technical services in ESCM.
- **Suppliers** (e.g. independent software vendors or sales organizations) define service offerings, so-called marketable services, for the technical services in ESCM. They publish the services to a marketplace.
- **Customers** register themselves or are registered by an authorized organization in ESCM and subscribe to one or more services. Users appointed by the customers work with the underlying applications under the conditions of the corresponding subscriptions.
- **Marketplace owners** are responsible for administrating and customizing the marketplaces to which services are published.
- **Operators** are responsible for deploying and maintaining ESCM.

In extended scenarios, the suppliers who define marketable services may involve additional users and organizations in offering and selling these services:

- **Brokers** support suppliers in establishing relationships to customers by offering the suppliers' services on a marketplace. A service subscription is a contract between the customer and the supplier.
- **Resellers** offer services defined by suppliers to customers applying their own terms and conditions. A service subscription establishes a contract between the customer and the reseller.

Developers in different organizations can use the public Web services and application programming interfaces (APIs) of ESCM for implementing applications that make use of ESCM features or for integrating external systems with ESCM.

1.1 The Developer's Tasks in ESCM

As a developer, you integrate external applications and systems with ESCM. You typically do this for the following organizations and purposes:

- Technology provider: You technically prepare applications for usage in a SaaS model and implement the services and interfaces required to integrate the applications with ESCM. Depending on the application, this may involve: Implementing a provisioning service, adapting the application's login and logout behavior, providing for the generation and sending of events, implementing operations that can be carried out from ESCM.
- Supplier or customer: You can integrate ESCM with an external process control system in order to control the execution of specific ESCM actions. This involves implementing a notification service which is invoked through appropriate triggers configured in ESCM. The

triggers are defined and managed by the administrator of the affected organization. For details, refer to the ESCM online help.

- Supplier: You can integrate an external tool for configuring service parameters with ESCM. The tool can be used by the supplier's customers instead of the default user interface for configuring service parameters on a marketplace.

For performing these tasks, you use the public Web services and APIs of ESCM. These interfaces, their documentation and additional resources, templates, samples, and utilities are provided in the ESCM integration package (*oscm-integration-pack.zip* file). A detailed documentation for the APIs is provided as Javadoc and in YAML files. By opening the `readme.htm` file of the integration package, you can access the available documentation as well as the resources themselves.

In order to provide for secure communication between ESCM and the applications or external systems integrated with it, you can make use of certificates.

An additional task of a developer in the ESCM context is the processing of billing data, for example, in an external accounting system. ESCM allows suppliers, brokers, resellers, marketplace owners, and operators to export customer billing data and revenue share data to XML files. From these files, the required data can be extracted for further processing.

1.2 SOAP-based Web Services

ESCM exposes its basic functionality as public Web services which are based on SOAP (Simple Object Access Protocol) and WSDL (Web Services Description Language). The Web services, their public APIs, and additional resources, templates, samples, and utilities can be used to integrate applications with ESCM in order to make them available as services to customers as well as to connect ESCM with external systems.

A Web service is a software module performing a discrete task or a set of tasks that can be accessed and invoked over a network, especially the World Wide Web. A provider makes Web services available to client applications that want to use them. A client application can invoke Web services through remote procedure calls (RPC). A published Web service is described in a WSDL file that allows you to locate it and evaluate its suitability for your needs. As an example, a company could provide a Web service to its customers to check an inventory on products before they order them.

Web services as well as the client applications can be written in different languages and run on different platforms. The ESCM Web services are written in Java using JAX-WS.

Web Service Standards

The development of Web services is based on the following standards:

- SOAP (Simple Object Access Protocol)

  SOAP is a transport-independent messaging protocol. SOAP messages are XML documents that are sent back and forth between a Web service and the calling application. SOAP uses one-way messages, although it is possible to combine messages into request-response sequences. The SOAP specification defines the format of the XML message but not its content and how it is actually sent. However, the SOAP specification defines how SOAP messages are routed over HTTP.

  The ESCM Web services use SOAP for the XML payload (XML data part) and HTTP as the transport protocol for the SOAP messages. The ESCM Web services support the SOAP 1.1 protocol.

- WSDL (Web Service Description Language), version 1.1

WSDL is an XML-based language used to define Web services and describe how to access them. Specifically, it describes the data and message contracts a Web service offers. By examining a Web service's WSDL document, developers know what methods are available and how to call them using the proper parameters.

For more information about SOAP and WSDL, refer to the SOAP and WSDL documents on the website of the World Wide Consortium website (`www.w3c.org`).

Web Service Versions

The current version of an ESCM Web service is provided in a `<documentation>` tag in the service's WSDL file.

A compatibility layer ensures that SOAP messages received from a Web service client are analyzed, and, if required, converted to the newest format. In addition, outgoing SOAP messages from ESCM are adapted to the version the client can consume.

Web service clients must provide the version of the Web services they want to consume in the header of their outgoing SOAP messages.

The version value can be specified in a configuration property of the Web service client. In case of compatible changes to the Web services in a new release of ESCM, it is sufficient to change the version value in the configuration of the client. The client itself only needs to be adapted if there are incompatible changes.

1.3 RESTful Web Services

Some ESCM components comply with the constraints of REST (Representational State Transfer) and offer a public REST API in addition to the SOAP-based API. For example, a REST API is available for integrating external process management systems with ESCM.

REST is an architectural style that specifies certain constraints with the purpose to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability. If applied to Web services, these properties are intended to enable the services to work best on the Web. In REST, data and functionality are considered resources. The resources are accessed using Uniform Resource Identifiers (URIs)and a stateless communication protocol. The resources are retrieved and manipulated with a set of simple, well-defined operations: `GET`, `PUT`, `POST`, and `DELETE`.

The ESCM REST APIs use HTTP as the protocol and JSON for data representation and transfer. The API version to be addressed is specified as part of the URI.

2 Platform Services and Web Service APIs

An application that integrates with ESCM invokes the functionality of the ESCM platform services.

In addition, the ESCM integration package contains APIs for implementing a provisioning service, an operation service, or a notification service.

The subsequent sections describe the platform services and APIs in detail.

ESCM ships the Java interfaces for its Web services as `.jar` files. You can directly write your Java code and include these `.jar` files for calls from ESCM to the application (outbound calls), or implement the Java interfaces for calls from the application to ESCM (inbound calls). No code generation is required. Both call directions can be deployed in Java EE environments.

The available REST APIs are described in YAML files which are shipped with ESCM. With appropriate tools such as Swagger Editor, you can view the descriptions and collect experience with the APIs.

2.1 Platform Services

The following platform services are available:

- Account management service
- Billing service
- Categorization service
- Discount service
- Event management service
- Identification service
- Marketplace management service
- Organizational unit service
- Reporting service
- Review service
- Search service
- Service provisioning service
- Session service
- Subscription management service
- Tag service
- Trigger service
- Trigger definition service
- VAT service

The APIs of the platform services are available in the ESCM integration package.

2.1.1 Purpose and Usage

This section contains a description of the purpose and usage of each platform service. For details on the interfaces and resources, refer to the Javadoc and YAML documentation.

Account Management Service

Java interface: `org.oscm.intf.AccountService`

This service is used for managing the account data of organizations, including billing and payment information and custom attributes.

Billing Service

Java interface: `org.oscm.intf.BillingService`

This service is used for exporting billing data and revenue share data.

Operators, suppliers, resellers, brokers, and marketplace owners can export customer billing data and revenue share data from ESCM and process it using accounting, billing, and payment facilities that have already been established in their organization. This is useful, for example, to create invoices for customers who decide to pay on receipt of invoice, or for managing the revenue shares of the different organizations involved in offering and selling services.

The results of export operations are stored in XML files. For detailed information on the elements of the XML files, refer to *Customer Billing Data* on page 48 and *Revenue Share Data* on page 65.

Categorization Service

Java interface: `org.oscm.intf.CategorizationService`

This service is used for defining and managing service categories.

A marketplace owner organization can create any number of categories for its marketplace. Suppliers, brokers, and resellers can assign these categories to the services they publish on the marketplace. Customers can use the categories for browsing the service catalog and searching for services on the marketplace.

Discount Service

Java interface: `org.oscm.intf.DiscountService`

This service is used for retrieving discount values.

Event Management Service

Java interface: `org.oscm.intf.EventService`

This service is used for recording events which are generated during the operation of applications that are integrated with ESCM. Events can be used for billing and reporting. Examples of events are the completion of a specific transaction, or the creation or deletion of specific data.

For details on how to integrate an application with the ESCM event management, refer to *Integrating with ESCM Event Management* on page 29.

Identification Service

Java interface: `org.oscm.intf.IdentityService`

This service is used for managing user accounts, user roles, and logins.

Marketplace Management Service

Java interface name: `org.oscm.intf.MarketplaceService`

This service is used for managing marketplaces and the marketable services published on them.

Organizational Unit Service

Java interface: `org.oscm.intf.OrganizationalUnitService`

This service is used for creating and managing organizational units, their members, and the services they can access.

Reporting Service

Java interface: `org.oscm.intf.ReportingService`

This service is used for retrieving a list of available reports.

ESCM offers comprehensive reports for different purposes and at different levels of detail. Different types of report satisfy the needs of all participating parties. Reports can be displayed at the ESCM user interface and exported to different file formats.

Review Service

Java interface: `org.oscm.intf.ReviewService`

This service is used for creating and retrieving service reviews and ratings on a marketplace.

Search Service

Java interface: `org.oscm.intf.SearchService`

This service is used for searching for services published on a marketplace.

Service Provisioning Service

Java interface: `org.oscm.intf.ServiceProvisioningService`

This service is used for managing technical and marketable services in ESCM in order to make applications available in service offerings.

Session Service

Java interface: `org.oscm.intf.SessionService`

This service is used for storing, retrieving, and deleting ESCM session data. If you need to implement your own token handler and logout listener for an application integrated with ESCM, you have to implement calls to methods of this service. For details, refer to *Adapting the Login/ Logout Implementation* on page 23.

Subscription Management Service

Java interface: `org.oscm.intf.SubscriptionService`

This service is used for managing subscriptions. Depending on the instance provisioning mechanism of your applications integrated with ESCM, you may have to implement calls to methods of this service in the provisioning service. For details, refer to *Implementation Details* on page 20.

Tag Service

Java interface: `org.oscm.intf.TagService`

This service is used for retrieving the tags (search terms) of the tag cloud of a marketplace. The tag cloud is the area of a marketplace containing defined search terms (tags).

The operator can set a value for the maximum number of tags composing the tag cloud, and the minimum number of times a tag must be used in services to be shown in the tag cloud. The more often a specific tag is used for services, the bigger the characters of the tag are displayed at the user interface.

Trigger Service

Java interface: `org.oscm.intf.TriggerService`

REST API: `trigger-service-rest-api.yaml`

This service is used for retrieving and manipulating trigger process data. These data are used in a notification service to approve or reject ESCM actions in a process control system. For details on how to integrate external process control systems, refer to *Integrating External Process Control* on page 32 and the online help.

Trigger Definition Service

Java interface: `org.oscm.intf.TriggerDefinitionService`

REST API: `trigger-service-rest-api.yaml`

This service is used for managing the definitions of triggers which are used to interact with external process control systems. The triggers are defined and managed by the administrator of the affected organization. For details, refer to the ESCM online help.

VAT Service

Java interface: `org.oscm.intf.VatService`

This service is used for handling VAT-related tasks.

Suppliers can enable VAT rate support for their services and customers in order to invoice usage charges for subscriptions as gross prices. The suppliers are responsible for setting the correct VAT rates. For details on billing and payment, refer to the *Supplier's Guide*.

2.1.2 Addressing the Platform Services - SOAP

Once deployed, the ESCM public Web services can be addressed via WSDL URLs.

In the application server administration console, the `oscm-webservices.jar` subcomponent of the `oscm` application includes a descriptor file with all the information needed for finding out the WSDL URL of a platform service:

1. In the GlassFish administration console, go to **Common Tasks** -> **Applications** -> **oscm**.
2. On the **Descriptor** tab, open the `META_INF/sun-ejb-jar.xml` descriptor file of the `oscm-webservices.jar` subcomponent.

   For every platform service, the endpoint address URI shows the Web service name and whether it is to be addressed through basic authentication (`BASIC`), certificate-based authentication (`CLIENTCERT`), or a security token service (`STS`). Refer to *Authentication* on page 17 for details on the authentication types.

   The URL pointing to the WSDL definition of a platform service is constructed as follows:

   `<BASE_URL_HTTPS>/<endpoint-address-uri>?wsdl`

   where

   `<BASE_URL_HTTPS>` points to the local server and port where the platform services have been deployed.

   `<endpoint-address-uri>` is the address as defined in the `sun-ejb-jar.xml` descriptor file.

   `?wsdl` is the suffix to be used for identifying a WSDL file.

   Example: `https://myserver:8081/AccountService/BASIC?wsdl`

The following code creates a client for the Web service with a given WSDL.

```
QName qName = new QName("http://oscm.org/xsd",
   AccountService.class.getSimpleName());
//local wsdl file
URL wsdlURL = this.getClass().getResource("/AccountService.wsdl");
Service service = Service.create(wsdlURL, qName);
AccountService port = service.getPort(AccountService.class);
BindingProvider bindingProvider = (BindingProvider) port;
Map<String, Object> clientRequestContext =
 bindingProvider.getRequestContext();

clientRequestContext.put(BindingProvider.USERNAME_PROPERTY, "1234");
clientRequestContext.put(BindingProvider.PASSWORD_PROPERTY, "password");
clientRequestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
   "https://localhost:8081/AccountService/BASIC");
```

In addition, the client application must integrate the current version number in the SOAP message header:

```
<S:Header> ...
   <auth xmlns="http://com/auth/"
     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
     SOAP-ENV:actor="ctmg.service.version">v1.9</auth> </S:Header>
```

The ESCM integration package contains the following library: `oscm-api-client-handler.jar`. This library provides a SOAP handler to modify the outgoing SOAP messages, i.e. it can be used to add the version number to the SOAP message header.

The source files of this library are part of the `IntegrationhelperSRC.zip` archive.

2.1.3 Addressing the Platform Services - REST

In order to address resources of ESCM by means of the REST API, you use URLs built according to the following format:

`https://<host>:<port>/<basePath>/<resource>`

`<host>` and `<port>` point to the server where ESCM has been deployed. `<basePath>` is the `basePath` stated in the YAML file; it consists of the relevant ESCM component and the API version. `<resource>` is the resource path.

Example:

`https://myserver:8081/oscm-trigger/v1/triggers`

2.2 Web Service APIs

ESCM offers APIs for implementing the following Web services:

- Provisioning service
- Notification service
- Operation service

The APIs are available in the ESCM integration package.

2.2.1 Purpose and Usage

This section contains a description of the purpose and usage of each API. For details on the interfaces and resources, refer to the Javadoc and YAML documentation.

Provisioning Service API

WSDL file: `ProvisioningService.wsdl`

This interface must be implemented for providing a provisioning service to integrate an application with ESCM. For details, refer to *Implementing a Provisioning Service* on page 18.

Notification Service API

WSDL file: `NotificationService.wsdl`

REST API: `notification-rest-api.yaml`

This interface must be implemented if you want to integrate an external process control system. For details, refer to *Integrating External Process Control* on page 32.

Operation Service API

WSDL file: `OperationService.wsdl`

This interface must be implemented for providing additional functions or operations for a technical service that are to be accessible via the ESCM user interface. For details, refer to *Implementing Technical Service Operations* on page 30.

2.2.2 Addressing the APIs - SOAP

ESCM ships its APIs as WSDL files and as `.jar` files. You can directly implement the interfaces for calls from your application to ESCM. No code generation is required.

For all Web service interfaces, ESCM acts as a Web service client. ESCM tries to detect the version of the implemented Web service and adapts its client version to the Web service version. ESCM expects the version in the `<documentation>` tag of the respective WSDL file. If there is no version information, ESCM assumes the latest service version.

It is recommended to use the WSDL files delivered with the ESCM integration package for the deployment of the Web service you implement.

Below is an example for implementing a provisioning service as an EJB (annotated bean) for an application that is to be deployed in a Java EE-compliant application server (GlassFish):

1. Copy the `ProvisioningService.wsdl` file delivered with the ESCM integration package as well as its schema to the `META-INF/wsdl` directory of your application.
2. In the `webservices.xml` descriptor file, add the WSDL file as follows:

```
<webservices xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.2"
   xsi:schemaLocation=
      "http://java.sun.com/xml/ns/javaee
       http://www.ibm.com/webservices/xsd/
            javaee_web_services_1_2.xsd">
      <webservice-description>
         <display-name>ProvisioningService</display-name>
         <webservice-description-name>ProvisioningService
            </webservice-description-name>
         <wsdl-file>ProvisioningService.wsdl</wsdl-file>
         <port-component> ... </port-component>
```

```
      ...
    </webservice-description>
```

Below is an example for implementing a provisioning service for an application that is to be deployed as a standard `.war` archive (non-EJB implementation):

1. Copy the `ProvisioningService.wsdl` file delivered with the ESCM integration package as well as its schema to the `META-INF/wsdl` directory of your application.
2. In the `sun-jaxws.xml` descriptor file, add the WSDL file as follows:

```
<endpoints version="2.0"
       xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime">
       <endpoint name="ProvisioningService"
 implementation="org.oscm.jaxws.ProvisioningServicImpl"
             url-pattern="/ProvisioningService"
             wsdl="ProvisioningService.wsdl" />
</endpoints>
```

Refer to *Implementing a Provisioning Service* on page 18 for details.

2.2.3 Addressing the APIs - REST

The resources of Web services you develop using the REST APIs of ESCM are addressed by their URLs in the following format:

`https://<host>:<port>/<resource>`

`<host>` and `<port>` point to the server where the Web service has been deployed. `<resource>` is the path and name of the resource.

Example:

`https://myServer:8081/mynotification/v1/processes`

2.3 Authentication

ESCM is installed as a platform for public access from anywhere in the Internet. Users are authenticated with ESCM and can be managed in ESCM or an existing LDAP system of an organization. Web service calls are authenticated in ESCM either by providing a user key or ID and a password in their header, or by certificates.

Client Authentication

Every application integrated with ESCM or accessing the platform services acts as a Web service client. ESCM acts as the server, and the client must provide its authenticating data to the server:

The caller sends the key or ID and password of an ESCM user. SOAP-based calls address the ESCM Web services with the `BASIC` suffix.

3 Integrating Applications with ESCM

Integrating an application with ESCM involves the following implementation tasks:

- Implement a provisioning service using the provisioning API of ESCM.
- Adapt the login/logout implementation using the ESCM integration helpers.
- Integrate the application with the ESCM event management using the API of the event management service.
- Implement service operations using the operation API of ESCM.

Note:	For the integration, you use the SOAP-based interfaces of ESCM. REST APIs are not available for this purpose.

The sections of this chapter describe the required implementation steps in detail.

Prerequisites

Before you can start integrating an application with ESCM, you have to set up your development environment. The following prerequisites must be fulfilled:

- You have installed a servlet container or application server, where you can deploy your application and the provisioning service, if any.
- You have installed a Web service framework, for example JAX-WS (Java API for XML Web Services), to deal with the Web services.

  JAX-WS is a Java API for creating and using Web services. It is part of the Java EE platform from Oracle.

3.1 Implementing a Provisioning Service

As a first integration step, you implement a so-called provisioning service that exposes its operations as a Web service. A provisioning service is required for integrating an application with the subscription management of ESCM. The provisioning service is called by ESCM when customers subscribe to a service and manage their subscriptions. Additionally, the provisioning service may be called for creating and managing users.

You do not need to implement a provisioning service if you have chosen to use the external access type. With this access type, users access an application directly after subscribing to a corresponding service. They are redirected immediately to the underlying application. Any further interaction takes place directly between the user and the application without involving ESCM in any way. For details on access types, refer to the *Technology Provider's Guide*.

For the implementation of a provisioning service, you need to consider the following:

- **Instance provisioning**

  When a customer subscribes to a service, the underlying application is supposed to perform specific steps required for the subscription and return an identifier to ESCM for future reference. The term 'instance' denotes all the items that the application has provisioned for a subscription.

  The actions to be performed and the items to be created, if any, depend entirely on the concepts and functionality of your application. For example, if a customer creates and stores data when using your application, your application may create a separate workspace in a data container or a separate database instance.

- **Provisioning mode**

Instance provisioning can be performed in synchronous or asynchronous mode.

Synchronous mode is used if provisioning can be completed right away. The provisioning service triggers the application to perform all the required actions and confirms the operation as complete. ESCM then sets the subscription to active, which means that the service is ready to be used by the customer.

Asynchronous mode is used if provisioning operations take a long time because long-running processes or manual steps are involved, or when huge amounts of data or virtual machines need to be set up. In this case, the provisioning service notifies ESCM that the provisioning is pending. Required actions may have started on the application side, but have not been completed yet. The provisioning service needs to notify ESCM using the subscription management service when provisioning is either complete or cannot be completed.

ESCM supports the development of asynchronous provisioning services with the asynchronous provisioning platform (APP). This is a framework which provides a provisioning service as well as functions, data persistence, and notification features which are always required for integrating applications in asynchronous mode. The framework, samples, and documentation are provided in the integration package for asynchronous provisioning (`oscm-integration-app-pack.zip` file).

- **Application parameters**

  An application may have parameters that are of relevance for the service provisioning in ESCM. Parameters can be used to define different feature configurations or service restrictions, for example, the maximum number of folders, files, or objects that can be created. Application parameters are specified in the technical service definition.

  ESCM can pass parameters to your application through the instance provisioning call. Any further processing must be carried out by your application. Especially if parameters are used to impose restrictions on service usage, the application needs to ensure that the restrictions are met. For example, if there is a parameter to restrict the maximum number of files created for a subscription, the application needs to track the actual number and ensure that the maximum number is not exceeded.

  For details on how to define parameters in the technical service definition, refer to the *Technology Provider's Guide*.

- **User management**

  If users access your application through ESCM, you need to implement user management operations. These operations are called when users are assigned to or deassigned from a subscription in ESCM, or when user profiles are updated. Your application may take corresponding actions, for example, create corresponding user accounts in its own user management system.

To implement a provisioning service, you have two options:

- Implementing the service as a **Java service** (see *Implementing the Provisioninig Service as a Java Service* on page 19)
- Implementing the service as a **non-Java service** (see *Implementing the Provisioning Service as a Non-Java Service* on page 20)

3.1.1 Implementing the Provisioninig Service as a Java Service

It is recommended to use the JAX-WS framework for implementing a provisioning service. JAX-WS offers APIs to directly address a Web service specified by a WSDL file and to obtain a proxy implementation of the interface.

A provisioning service implementation is used for inbound calls. This means that you implement the Web service and annotate the implementation to get published.

If the application is deployed in a Java EE-compliant application server, the provisioning service can be deployed as an annotated bean. The service implementation can be either a stateless session bean when deployed as J2EE application, or a plain Java object when deployed as part of a Web application. In both cases, the annotation is `javax.jws.WebService` with a reference to the service definition.

Your provisioning service must implement the provisioning API of ESCM, for example, as shown here:

```
@WebService(serviceName = "ProvisioningService",
          targetNamespace = "http://oscm.org/xsd",
          endpointInterface =
             "org.oscm.provisioning.intf.ProvisioningService")
public class MyProvisioningServiceImpl implements ProvisioningService
{
   //...
}
```

Note that the target namespace must be exactly the same as defined in the Web service interface.

The `endpointInterface` attribute refers to the Java interface defining the Web service. This allows the implementation class to provide the respective Java method implementations. All Web service-related annotations are taken from the interface.

3.1.2 Implementing the Provisioning Service as a Non-Java Service

Non-Java clients need to conform to the WSDL and XSD files shipped with ESCM: You create a Web service based on the `ProvisioningService.wsdl` document.

Depending on the platform and the Web services framework you are using, code generation will be required or dynamic usage will be possible. For example, dynamic languages like PHP or Python allow generating proxy object instances directly from a WSDL at runtime.

3.1.3 Implementation Details

The following steps provide an overview of the methods you need to implement for a provisioning service.

Note: If you are using the asynchronous provisioning platform for developing a provisioning service, you do not need to implement these methods. Instead, you develop a service controller.

For details, refer to the documentation provided with the integration package for asynchronous provisioning (`oscm-integration-app-pack.zip` file).

1. Implement at least the following methods:
   - `createInstance` for the synchronous provisioning of an instance or `asyncCreateInstance` for the asynchronous provisioning of an instance.
   - `deleteInstance`
2. If your application has parameters that are set during instance provisioning, implement the `modifySubscription` or `asyncModifySubscription` method.
3. If your application allows for upgrading or downgrading to another application instance, implement the `upgradeSubscription` or `asyncUpgradeSubscription` method.

4. If users access your application through ESCM, implement the following methods:
   - `createUsers`
   - `deleteUsers`
   - `updateUsers`
5. If you are using asynchronous instance provisioning (`asyncCreateInstance`), you need to implement calls to the following methods of the subscription management service (see *Platform Services* on page 11):
   - `abortAsyncSubscription`
   - `completeAsyncSubscription`
6. If you implement the `asyncModifySubscription` method, you need to implement calls to the following methods of the subscription management service (see *Platform Services* on page 11):
   - `abortAsyncModifySubscription`
   - `completeAsyncModifySubscription`
7. If you implement the `asyncUpgradeSubscription` method, you need to implement calls to the following methods of the subscription management service (see *Platform Services* on page 11):
   - `abortAsyncUpgradeSubscription`
   - `completeAsyncUpgradeSubscription`
8. Deploy your provisioning service and make sure that it is operational.

For each organization role, the following tables list the methods of the provisioning service which are called when specific user actions are executed in ESCM. The dependencies on the chosen access type and other parameters are also described, if applicable.

For details on access types, refer to the *Technology Provider's Guide*.

Technology Provider

User Action	Method	Behavior
Import service definition Click a service entry to view the details in the ESCM administration portal.	`sendPing()`	`sendPing()` is called for all access types except external.

Supplier/Reseller/Broker

User Action	Method	Behavior
Terminate a subscription	`deleteInstance()`	`deleteInstance()` is called for each deleted subscription and for all access types except external.
Activate or deactivate services Activate a service.	`sendPing()`	`sendPing()` is called for all access types except external.

User Action	Method	Behavior
Manage payment types Deactivate/reactivate a payment type for a customer.	`deactivateInstance()` `activateInstance()`	`deactivateInstance()` and `activateInstance()` are called for all access types except external. `deactivateInstance()` is called if the customer uses the deactivated payment type to pay for his subscriptions. `activateInstance()` is called if a deactivated payment type used by the customer is made available again. The methods are called for each affected subscription.

Customer

User Action	Method	Behavior
Subscribe to a service	`createInstance()` `asyncreateInstance()`	`createInstance()` is called for login, direct, and user access in synchronous mode. `asyncreateInstance()` is called for login, direct, and user access in asynchronous mode. A list of all parameters and values of a service can be retrieved with the `getParameterValue()` method of the `InstanceRequest` class.
Manage Users Modify user profiles, no changes in assignments to subscriptions	`updateUsers()`	`updateUsers()` is called for login and user access. The method is called independent of any changes in the user profile.
Assign users to a subscription Assign one or more users to the selected subscription	`createUsers()`	`createUsers()` is called for login and user access. The method is called for each assigned user. The `users` parameter contains the list of users who are to be assigned to the selected subscription.
Deassign users from a subscription Deassign one or more users from the selected subscription	`deleteUsers()`	`deleteUsers()` is called for login and user access. The method is called for each deassigned user. The `users` parameter contains the list of users who are to be deassigned from the selected subscription.

User Action	Method	Behavior
Modify a subscription	`modifySubscription()` `asyncModifySubscription()`	`modifySubscription()` is called in synchronous mode for all access types except external. `asyncModifySubscription()` is called in asynchronous mode for all access types except external. The method is called when customizable parameters or the subscription ID are changed by the customer.
Up/Downgrade a subscription	`upgradeSubscription()` `asyncUpgradeSubscription()`	`upgradeSubscription()` is called in synchronous mode for all access types except external. `asyncUpgradeSubscription()` is called in asynchronous mode for all access types except external. The method is called when a customer upgrades or downgrade a subscription.
Terminate a subscription	`deleteInstance()`	`deleteInstance()` is called for all access types except external.

If a subscription expires, the `deactivateInstance()` method is called and the instances related to the subscription are deactivated. If the subscription is updated and activated again, the `activateInstance()` method is called.

For details on the methods and data objects, refer to the Javadoc for the ESCM provisioning API shipped in the ESCM integration package.

3.2 Adapting the Login/Logout Implementation

If you opt for access through ESCM (login access), you need to adapt the login/logout implementation of your application to pass the control and authentication of users from the application to ESCM, and implement the relevant methods defined by the provisioning service.

The required functionality for login and logout is distributed between a token handler, a custom login module, a custom logout module, and a logout listener:

- The **token handler** is responsible for requesting ESCM to resolve a user token into a user ID. It takes up the task of creating a session object and storing the user ID in that object. Additionally, it forwards requests containing a resolved user token to the custom login module.
- The **custom login module** passes the user ID to the application. It enables users to log in to the application without requesting any further credentials. Users are trusted because they have already been authenticated by ESCM. For example, a custom login module might pass the user ID and a default password to the application.

  To ensure that any login takes place through ESCM, the direct login to your application must be bypassed. For this purpose, you have to implement the required interface methods between the application and ESCM.

- The **custom logout module** closes user sessions on the application side and redirects users to the logout page of ESCM. The URL of the logout page is returned to the application by the `deleteServiceSession` method of the ESCM session service.
- The **logout listener** notifies ESCM when a user logs out or a session timeout occurs.

As for the custom login and logout modules, you need to analyze the existing functionality of your application and adapt it to match the required behavior.

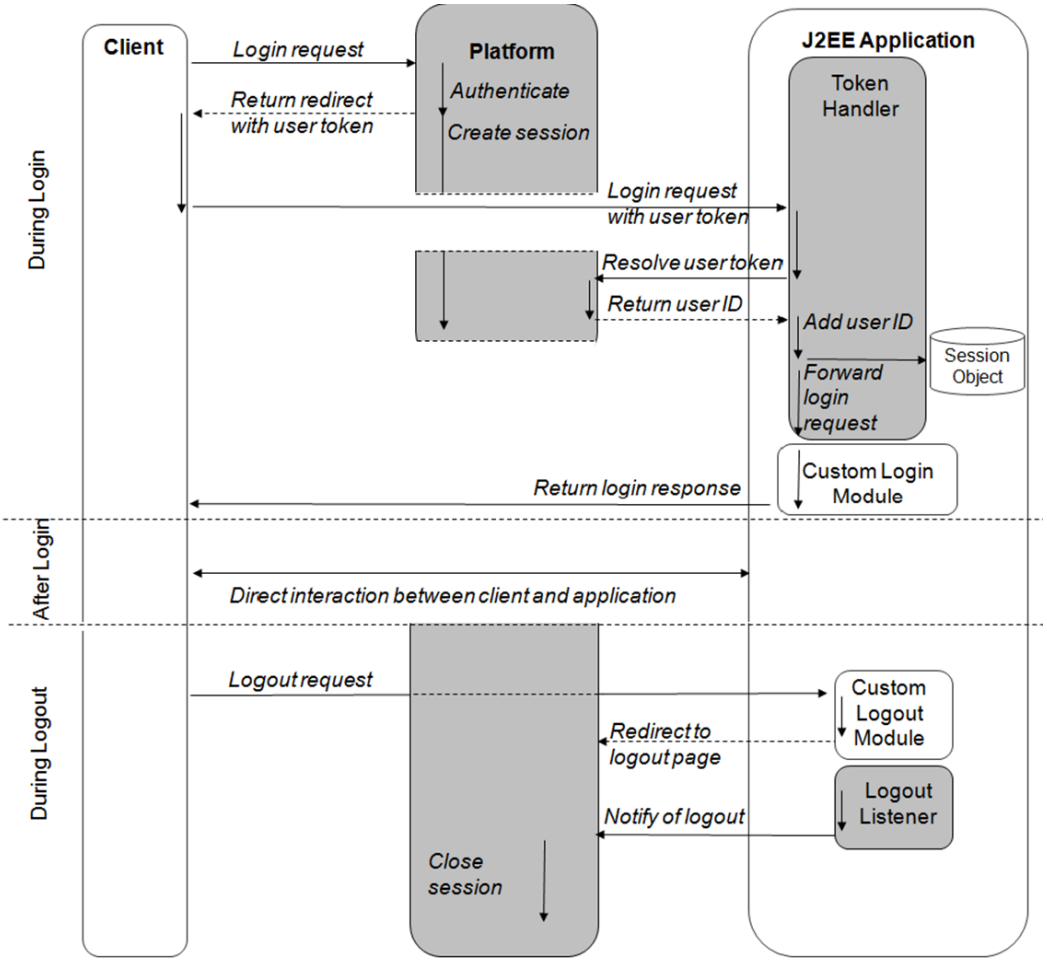
ESCM provides resources for J2EE environments that support you in adapting the login/logout implementation of a Java-based Web application. The resources are called **integration helpers** and contain a ready-to-use token handler and logout listener.

The integration helpers can be used with any Java-based Web application. The token handler is implemented as a Java servlet and as a JSP; the logout listener is implemented as a session listener.

If your application is based on a different technology, you need to implement your own token handler and logout listener by calling the following methods of the platform Session Service (see *Platform Services* on page 11):

- `resolveUserToken` for requesting ESCM to resolve a user token
- `deleteServiceSession` for notifying ESCM of a logout or session timeout

The following figure illustrates how login access works and which parts have to be added or adapted in the login/logout implementation of your application (components provided by ESCM including the integration helpers are shown in gray):



The integration helpers are contained in the `integrationhelper` subfolder of the ESCM integration package (*`oscm-integration-pack.zip`*):

- `oscm-webservices-proxy.jar`: A library containing a Web service proxy for `BASIC` Web service ports of the ESCM platform services. This proxy is implemented in Java and ready to be used. It creates a Web service proxy containing all information required to establish a connection with an ESCM Web service.
- `Integrationhelper.jar`: A library containing a token handler and logout listener. They are also implemented in Java and ready to be used.
- `oscm-api-client-handler.jar`: A library containing a SOAP handler for adding a service version number to the outbound SOAP messages of an ESCM Web service client. The service version can be found in the `<documentation>` tag included in the WSDL files.
- `Integrationhelper.war`: A Web archive including the above integration helpers and additional resources required by them.

- `IntegrationhelperSRC.zip`: An archive file containing the source code of the integration helpers.

The `Integrationhelper.war` package has the following contents:

Package Contents	Description

`Tokenhandler.jsp`
`Logout.jsp`
`Welcome.jsp`
`TestWebService.jsp`
	Token handler, welcome, and logout page implemented as Java server pages (JSP), as well as a page for testing Web service calls.

`WEB-INF/web.xml`	A sample servlet configuration for the integration of a token handler and a logout listener in a custom Web application.
`WEB-INF/lib`	A folder containing all JAR files required by the ESCM integration helpers.
`WEB-INF/classes/com`	A folder containing the ready-to-use logout listener class:`LogoutListener.class`, as well as the ready-to-use token handler class: `TokenHandler.class`.
`WEB-INF/classes/` `tokenhandler.properties`	A property file specifying the relative URL to the custom login module of your application. The custom login module must be on the same machine and in the same context as the token handler.
`WEB-INF/classes/` `webserviceclient.properties`	A property file specifying the information needed to access the ESCM platform services.

To integrate the ESCM integration helpers into your Web application:

1. Extract the `Integrationhelper.war` package from the ESCM integration package to a location of your choice, or add the `Integrationhelper.jar` and `oscm-webservices-proxy.jar` packages to the class path of your application.
2. Configure the `webserviceclient.properties` file and enter the following information:
   - Server and port where the ESCM platform services have been deployed.
   - The URL of the WSDL file of the platform's Session Service.

     The Web service port of an ESCM platform service is `BASIC`.
   - ID or key as well as password of the calling user. The user must have the role of a technology manager in a technology provider organization.
   - The ESCM platform service version.
3. Configure the `tokenhandler.properties` file and specify the relative URL to the custom login module of your application. The custom login module must be on the same machine and in the same context as the token handler.
4. Copy the contents of the `WEB-INF/lib` and `WEB-INF/classes` folders to the corresponding folders of your Web application.

5. Do one of the following:
   - If you want to use the JSP-based token handler, copy the `Tokenhandler.jsp` file to your Web application.
   - If you want to use the servlet-based token handler, register it in the `web.xml` file of your Web application.

     To do this, you can copy the `servlet` and `servlet-mapping` sections from the sample `web.xml` file, which is part of the `Integrationhelper.war` package, to the `web.xml` file of your Web application.
6. Register the logout listener in the `web.xml` file of your Web application.

   To do this, you can copy the `listener` section from the sample `web.xml` file, which is part of the `Integrationhelper.war` package, to the `web.xml` file of your Web application.
7. In the technical service definition of the application, the path to the token handler must be specified in the `loginPath` attribute for login access.

   For details on the technical service definition, refer to the *Technology Provider's Guide*.

Example

Suppose you implemented a servlet-based custom login module for your application, which is to be called after the user token has been resolved. The custom login module is available as a Web resource as a `DoAutoLoginServlet` servlet at `/DoAutoLogin`. Your Web application is running on `myserver` at port `7777` using the context `myapplication`. The ESCM platform services are listening at port `8081`. ESCM is installed in SAML_SP mode.

Depending on whether you use the servlet-based or the JSP-based token handler, the configuration files look as shown below:

Servlet-Based Token Handler

- `webserviceclient.properties`

```
# Machine where OSCM / ESCM is hosted and port to access the web
# service

cm.host=localhost
cm.port=8081

# ESCM web services are available on different web
# service ports (endpoints),
# depending on the authentication mode.
#
# Example - Web service port URLs for the OSCM session
# service:
# https://localhost:8081/SessionService/BASIC?wsdl
# https://localhost:8081/SessionService/CLIENTCERT?wsdl
#
# Set the web service port suffix, depending on the
# authentication mode:
# - authentication mode INTERNAL: use BASIC or CLIENTCERT

cm.service.port=BASIC

# OSCM web service version

cm.service.version=v1.9

# User credentials to access the OSCM web service.
```

```
# User key is used for BASIC and CLIENTCERT ports.

cm.service.user.key=1000
cm.service.user.password=<user_password>
```

- tokenhandler.properties

```
FORWARD=/Welcome.jsp
```

- web.xml

```
...
<servlet>
    <display-name>TokenhandlerServlet</display-name>
    <servlet-name>TokenhandlerServlet</servlet-name>
    <servlet-class>org.oscm.integrationhelper.TokenhandlerServlet
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>TokenhandlerServlet</servlet-name>
    <url-pattern>/resolveToken</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>DoAutoLoginServlet</servlet-name>
    <servlet-class>servlets.DoAutoLoginServlet
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DoAutoLoginServlet</servlet-name>
    <url-pattern>/doAutoLogin</url-pattern>
</servlet-mapping>

<listener>
 <listener-class>org.oscm.integrationhelper.LogoutListener
    </listener-class>
</listener>
...
```

- Technical service definition

```
...
<TechnicalService
   id="SampleService"
   baseUrl="https://myserver:7777/myservice"
   loginPath="/resolveToken"
   ...
>
...
```

JSP-Based Token Handler

- webserviceclient.properties

```
# Machine where ESCM is hosted and port to access the
# web service
cm.host=localhost
cm.port=8081

# ESCM web services are available on 3 different web
# service ports (endpoints),
# depending on the ESCM authentication mode.
```

```
#
# Example - Web service port URLs for the ESCM session
# service:
# https://localhost:8081/SessionService/BASIC?wsdl
# https://localhost:8081/SessionService/CLIENTCERT?wsdl
# https://localhost:8081/SessionService/STS?wsdl&tenantID=xxxxxxxx
#
# Set the web service port suffix, depending on the ESCM
# authentication mode:
# - authentication mode INTERNAL: use BASIC or CLIENTCERT
# - authentication mode SAML-SP : use STS

cm.service.port=BASIC

# ESCM web service version

cm.service.version=v1.9

# User credentials to access ESCM web service.
# User key is used for BASIC and CLIENTCERT ports, and user ID
# and tenant ID for STS port.

cm.service.user.key=1000
cm.service.user.id=administrator
cm.service.user.password=<user_password>
cm.service.user.tenantID=<tenant_ID>
```

- `tokenhandler.properties`

```
FORWARD=/Welcome.jsp
```

- `web.xml`

```
...
<listener>
   <listener-class>org.oscm.integrationhelper.LogoutListener
      </listener-class>
</listener>
...
```

- Technical service definition

```
...
<TechnicalService
   id="SampleService"
   baseUrl="https://myserver:7777/myservice"
   loginPath="/Tokenhandler.jsp"
   ...
>
...
```

3.3 Integrating with ESCM Event Management

The event management service in ESCM collects specific events generated during application operation. These events can be used for price models, billing, and reporting. Examples of events are the completion of a specific transaction, or the creation or deletion of specific data.

Your application can send events to ESCM at runtime through the event management service, which is one of the ESCM platform services.

To integrate an application with ESCM event management:

1. If your application does not generate the required events yet, implement the generation of events.
2. Depending on the information available for a subscription, implement the sending of events to ESCM by calling one of the following methods of the event management service (see *Platform Services* on page 11):
   - `recordEventForSubscription` (subscription key must be specified)
   - `recordEventForInstance` (ID of the technical service and instance ID must be specified)

   For details on the methods, refer to the Javadoc for the ESCM platform services.
3. When preparing the technical service definition, declare the events that your application will send.

Note: ESCM comes with two predefined events, which can be used with login access: login to a service and logout from a service. These events are generated automatically and need not to be implemented in the application or defined in the technical service definition. To use the logout event, you must implement a logout listener. For details, refer to *Adapting the Login/Logout Implementation* on page 23.

3.4 Implementing Technical Service Operations

You may wish that your technical service offers additional operations or functions that are to be accessible via the ESCM user interface. In a SaaS environment, applications are not installed locally but provisioned as services. Therefore, users cannot access the system resources the applications are using, for example, to perform administrative tasks such as system backup or shutdown. Technical service operations can be used to access the resources of an application and perform administrative tasks without actually opening the application.

To provide technical service operations, a Web service based on the operation service API must be implemented. The operations, their parameters, and the access information of the Web service must be specified in the technical service definition for the application. For details, refer to the *Technology Provider's Guide*.

To integrate a technical service operation:

1. Implement the `executeServiceOperation()` and the `getParameterValues()` methods in a Web service definition using the `OperationService` interface of the operation API, which is part of the ESCM integration package.

   The following information is provided as parameter settings for the `executeServiceOperation()` method:
   - The user identifier. This ID is unique in the instance context and allows the technical service to perform built-in security checks.
   - The instance identifier. This ID uniquely identifies the application instance that is to be affected by the operation.
   - For asynchronous operations: The transaction identifier. This ID uniquely identifies the transaction to be executed.
   - The operation identifier. This ID uniquely identifies the operation to be executed as defined in the technical service definition. For details, refer to the *Technology Provider's Guide*.

- Parameters. This is the list of parameters and their values as retrieved with the `getParameterValues()` method or - depending on the type of parameter - their specified values. The parameters must also be defined in the technical service definition. For details, refer to the *Technology Provider's Guide*.

You need to provide the operation in a WSDL file that is accessible by a URL.

2. Edit the technical service definition XML file and include the operation and its parameters.

   For details, refer to the *Technology Provider's Guide*.

When implementing service operations, be aware of the following:

- The return type is not configurable; currently it can only be of type `OperationResult`.
- ESCM does not provide access control. Every user who can use the subscription can also execute the operations.
- The operations can be executed in synchronous or asynchronous mode. The transaction mode is defined in the `OperationResult` class of the `OperationService` interface.
- For allowing users to view the status of asynchronous service operations, implement the `updateAsyncOperationProgress()` method using the `SubscriptionService` interface of the ESCM public Web service API.

4 Integrating External Process Control

Organizations often have specific processes for registering users, subscribing to services, or defining prices. Usually, such processes include approval processes and are modeled and automated with a process control system.

Certain actions of customers and suppliers can be carried out under the control of an external process control system. You can configure so-called triggers which are invoked when these actions are carried out. The triggers start the corresponding process in the process control system. If approval for the action is required, it is suspended until it is approved or rejected in the process control system. If no approval is required, the action is instantly executed.

As a prerequisite for controlling actions by processes, a notification service must exist and be deployed. This service forms the interface between the platform and the process control system.

Users can see all pending actions at the ESCM user interface, cancel them, or delete aborted ones.

Process-Controlled Actions

The following actions may be subject to process control and thus to approval in an external process control system before they are executed in ESCM:

- For any type of organization, the following triggers can be configured:
  - A subscription is to be changed, for example, renamed (`Modify subscription`).
  - A billing run is completed, and the billing data for a billing period is calculated (`Billing run finished`).
  - A subscription is to be added (`Subscribe to service`).
  - A user is to be assigned to or removed from a subscription (`Assign users to subscription`).

    For the case that a user is automatically assigned to a new subscription when subscribing to a service, no trigger can be configured. The `Assign users to subscription` trigger only applies to users that are manually assigned to a subscription.
  - A subscription is to be upgraded or downgraded (`Up/Downgrade subscription`).
  - A subscription is to be terminated (`Terminate subscription`).
  - A user is to be registered (`Register user`).

    Triggers can only be configured for registering single users. The import of multiple users in one operation from an LDAP system or a user data file cannot be executed under process control.
- For suppliers, the following additional triggers can be configured:
  - A customer is to be registered (`Register customer`).
  - The payment types for a customer are to be changed (`Manage payment types for customer`).
  - A marketable service is to be activated (`Activate service`).
  - A marketable service is to be deactivated (`Deactivate service`).
  - A user of an organization subscribes to a service offered by the supplier (`Subscription created (any user)`).
  - A user of a customer modifies a subscription to a service of the supplier (`Subscription modified (any user)`).

- A user of a customer terminates a subscription to a service of the supplier (`Subscription terminated (any user)`).

Connecting to an External Process Control System

If you want ESCM to connect to an external system to handle notifications and thus make use of the trigger processing feature of ESCM, the following steps are required:

1. Implement a notification service:

   An external system that is to be notified about certain actions must implement a Web service that connects ESCM with it. The Web service must implement the notification API, which is available in the following variants:

   - SOAP-based API, `org.oscm.notification.intf.NotificationService`.
   - REST API, described in `notification-rest-api.yaml`. Be aware that this API only supports `Subscribe to service`, `Modify subscription`, and `Terminate subscription` triggers.

   Both APIs and their documentation are provided with the ESCM integration package.

2. Deploy the notification service in your environment. You can do this on the system hosting the process control system you want to use, or on any other system.
3. Configure the relevant triggers in ESCM. To do this, you must be an administrator of your organization in ESCM.

   When defining a trigger, you need to provide the URL of the notification service:

   - When using the SOAP-based API, this is the URL of the service's WSDL file, for example: `https://myServer:8280/NotificationService?wsdl`.
   - When using the REST API, this is the URL to which ESCM can send a `POST` request for notification purposes.

   For details on how to define and manage triggers in ESCM, refer to the ESCM online help.

   As soon as the notification service is deployed and the process triggers have been configured in ESCM, every action for which a process trigger has been configured results in a notification to your notification service. A trigger process key is sent for all actions that require a callback from the external system. This key is necessary to approve or reject an action.

   The process triggers are queued and executed asynchronously. They are not called inside the transaction initiated by the user.

4. For approval or rejection, the process control system needs to send its response to the trigger service of ESCM (see *Platform Services* on page 11). The API of the trigger service is available in the following variants:

   - SOAP-based API, `org.oscm.intf.TriggerService`.
   - REST API, described in `trigger-service-rest-api.yaml`.

   As a result of the response, the interrupted action in ESCM is continued or canceled. When rejecting an action, the reason for the rejection can be passed.

5 Integrating an External Parameter Configuration Tool

A supplier organization can integrate a third-party or proprietary tool for configuring service parameters with ESCM and for viewing the prices of parameters per subscription and/or per user. The tool can be used by the supplier's customers for defining and updating the parameters of a service. It replaces the default user interface for configuring service parameters on a marketplace.

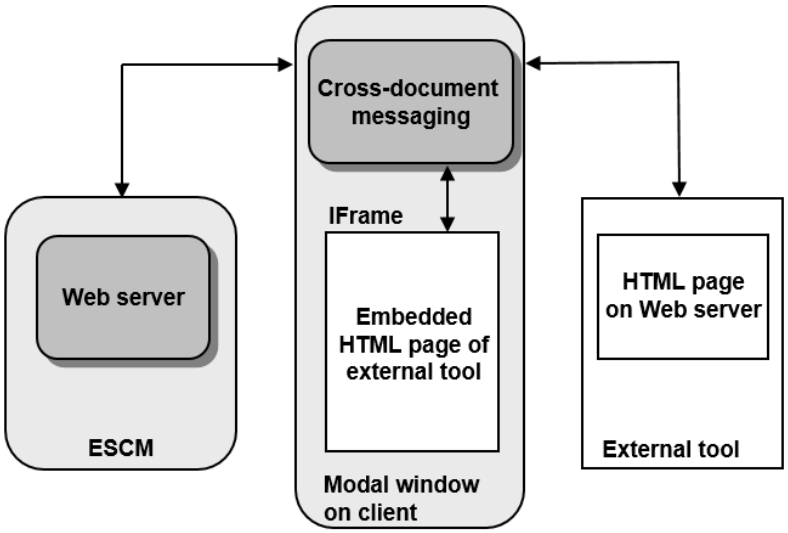
ESCM allows for a seamless integration of the tool. A generic JavaScript interface is provided by which supplier organizations can integrate a tool of their choice. The following basic concepts are relevant:

- Cross-document messaging is used for the communication between ESCM and an external parameter configuration tool.
- The exchange of data is based on JavaScript Object Notation (JSON).

The sections of this chapter describe the basic concepts and give an overview of the required implementation tasks of a developer.

5.1 Cross-Document Messaging

The following illustration shows how ESCM uses cross-document messaging for the communication with a parameter configuration tool:



Cross-document messaging is a JavaScript API which enables communication between documents from different origins or source domains. With the methods of this API, scripts can interact across domain boundaries and send plain text messages from one domain to another. Cross-document messaging can be used in all Web browsers supported by the ESCM user interface.

User action in a Web browser (client) initiates cross-document messaging:

1. When subscribing to a service or updating a service on a marketplace, a customer clicks **Configure** to specify the service parameters. This is the trigger for loading a modal window in the Web browser:
   - The lower part of the modal window is an inline frame element (`<iframe>`) which links the external parameter configuration tool with ESCM. The HTML page of the tool is the source of the frame.
   - The upper part of the window outputs error messages in case the validation of an HTTP response from the tool fails.
2. The Web browser sends an HTTP request to the Web server where the external parameter configuration tool is deployed to load the HTML page of the tool into the inline frame element.
3. As soon as the HTML page of the external parameter configuration tool is loaded, the Web browser receives the first message from the tool, the initialization message. This confirms that the communication is enabled. The HTML page of the tool is displayed in the Web browser.

ESCM delivers a sample JavaScript tool that you can integrate with ESCM to see how cross-document messaging can be implemented. For information on the resources that support you in integrating a parameter configuration tool, refer to *Overview of Tasks* on page 39.

5.2 Data Exchange

ESCM specifies the data structures that can be exchanged during the parameter configuration process with an external tool. Based on the service parameters that are defined for the underlying technical services, ESCM specifies:

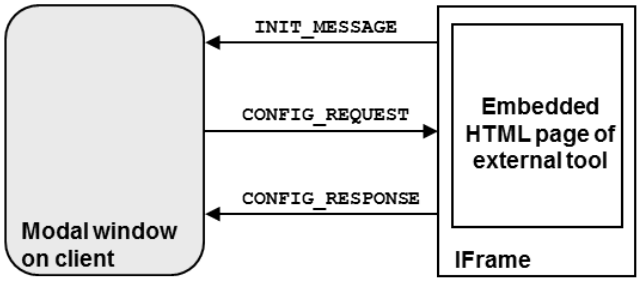
- The input data structures that are sent from ESCM to the parameter configuration tool.
- The output data structures that must be returned to ESCM by the parameter configuration tool.

ESCM uses JavaScript Object Notation (JSON) as data-interchange format. JSON offers an open standard data format that is language-independent and primarily used for transmitting text messages between Web applications.

Note: The parameter configuration tool must support JSON to exchange data with ESCM. It is up to the tool how the text messages are encoded and decoded. The tool must not necessarily be a JavaScript application.

Message Types

The configuration process with an external tool consists of three types of messages: `INIT_MESSAGE`, `CONFIG_REQUEST`, and `CONFIG_RESPONSE`.



An **initialization message** is sent from the external tool to ESCM as soon as the configuration process is triggered by a user action in the Web browser (client). The JSON message defines the height and width of the inline frame element that is used for loading the external HTML page in ESCM. The message is the trigger for displaying the modal window with the external HTML page. It instantiates the external parameter configuration tool.

When the initialization message is received, a **configuration request** is sent from ESCM to the external tool. This JSON message contains the definition of the service parameters that are provided by the underlying technical service with their default values, if available. The user can then specify values for these parameters in the modal window.

A **configuration response** is sent from the external parameter configuration tool to ESCM as soon as the configuration is successfully finished or canceled:

- If the configuration is successfully finished, the JSON message contains the specified parameter values which are instantly validated by ESCM.

  If the validation is successful, the parameter data is stored in ESCM and the modal window is closed. If the validation fails, a corresponding validation error is displayed in the inline frame element and a new configuration request is sent.
- If the configuration is canceled, the modal window is closed.

Message Syntax

The syntax of the messages must be as shown in the following examples.

Initialization message:

```
{"messageType":"INIT_MESSAGE",
    "screenWidth":600, "screenHeight":400
}
```

Configuration request:

```
{"messageType":"CONFIG_REQUEST",
    "locale":en",
    "parameters":[
        {"id":"par1", "valueType":"INTEGER", "value":"", ...},
        {"id":"par2", "valueType":"BOOLEAN", "value":"true", ...},
        {...}
    ]
}
```

Configuration response - configuration process finished:

```
{"messageType":"CONFIG_RESPONSE",
    "responseCode":"CONFIGURATION_FINISHED",
    "parameters":[
        {"id":"par1","valueType":"INTEGER", "value":"100", ...},
        {"id":"par2","valueType":"BOOLEAN", "value":"true", ...},
        {...}
    ]
}
```

Configuration response - configuration process canceled:

```
{"messageType":"CONFIG_RESPONSE",
    "responseCode":"CONFIGURATION_CANCELED"
}
```

Parameter Definitions

The parameters that are to be configured with an external parameter configuration tool are defined in the technical services in ESCM. For details on technical services, refer to the *Technology Provider's Guide*.

The parameter definitions must be passed from ESCM to the external tool and vice versa. Refer to the next section for the complete set of parameter definitions.

Parameter Syntax

In configuration requests and responses, the syntax for the parameters with the different data types must be as shown in the following examples.

Parameters of type `BOOLEAN`:

```
{"messageType":"CONFIG_REQUEST",
    "locale":en",
    "parameters":[
        {"id":"RENAME_FOLDER",
         "valueType":"BOOLEAN",
         "mandatory":false,
         "description":"Rename a user folder",
         "value":"true",
         "readonly":false,
         "modificationType":"STANDARD",
         "valueError":false,
        },
        {...}]
}
```

Parameters of type `INTEGER`:

```
{"messageType":"CONFIG_REQUEST",
    "locale":en",
    "parameters":[
        {"id":"MAX_FILE_NUMBER",
         "valueType":"INTEGER",
         "minValue":"12",
         "maxValue":"500",
         "mandatory":false,
         "description":"Number of files that can be uploaded",
         "value":"200",
         "readonly":false,
         "modificationType":"STANDARD",
         "valueError":false,
        },
        {...}]
}
```

Parameters of type `LONG`:

```
{"messageType":"CONFIG_REQUEST",
    "locale":en",
    "parameters":[
        {"id":"WS_SLEEP",
         "valueType":"LONG",
         "minValue":"0",
         "maxValue":"1200000",
         "mandatory":false,
         "description":"Defines the time interval to wait before
```

```
          an outgoing Web service of the createUsers call will
          be executed",
         "value":"0",
         "readonly":false,
         "modificationType":"STANDARD",
         "valueError":false,
        },
        {...}]
}
```

Parameters of type `STRING`:

```
{"messageType":"CONFIG_REQUEST",
    "locale":en",
    "parameters":[
        {"id":"STRING_DATA_TYPE",
         "valueType":"STRING",
         "mandatory":false,
         "description":"Number of folders that can be created",
         "value":"",
         "readonly":false,
         "modificationType":"STANDARD",
         "valueError":false,
        },
        {...}]
}
```

Parameters of type `ENUMERATION`:

```
{"messageType":"CONFIG_REQUEST",
    "locale":en",
    "parameters":[
        {"id":"DISK_SPACE",
         "valueType":"ENUMERATION",
         "mandatory":false,
         "description":"Incremental disk storage",
         "value":"1",
         "readonly":false,
         "modificationType":"STANDARD",
         "valueError":false,
         "options":[
            {"id":"1",
             "description":"Minimum Storage (100 GB)"
            },
            {"id":"2",
             "description":"Optimum Storage (200 GB)"
            },
            {"id":"3",
             "description":"Maximum Storage (300 GB)"
            }]
        },
        {...}]
}
```

Parameters of type `DURATION`:

```
{"messageType":"CONFIG_REQUEST",
    "locale":en",
    "parameters":[
        {"id":"PERIOD",
         "valueType":"DURATION",
```

```
          "minValue":"0",
          "maxValue":"106751991167",
          "mandatory":false,
          "description":"Number of days after which the
           subscription is deactivated",
          "value":"",
          "readonly":false,
          "modificationType":"STANDARD",
          "valueError":false,
        },
        {...}]
}
```

5.3 Overview of Tasks

To enable the integration with ESCM, the following development tasks have to be implemented on the side of the external parameter configuration tool. ESCM provides resources that support you in the implementation.

Implementing Cross-Document Messaging

For sending text messages from the domain of the external parameter configuration tool to the domain of ESCM, you can use the public JavaScript API for cross-document messaging.

ESCM provides a sample tool implemented in JavaScript. You can integrate it with ESCM to see how cross-document messaging works. The sample demonstrates the key aspects involved in the integration.

The sample tool is provided in the ESCM integration package (*oscm-integration-pack.zip*). From the `readme.htm` file of the integration package, you can download the sample and directly access the description on how to integrate it.

Implementing JSON Encoding and Decoding

For sending data to ESCM, the parameter configuration tool must encode its text messages to JSON. JSON messages received from ESCM must be decoded.

If you want to encode and decode the text messages with JavaScript, you can use the public domain JavaScript library `json2.js`. It contains the methods required for implementing JSON encoders and decoders.

Implementing the Data Structures

For the exchange of data, the message and parameter definitions must be implemented in accordance with the predefined syntax. For details, refer to *Data Exchange* on page 35.

A JavaScript library, `configurationAPI.js`, is provided that you can use for implementing the message and parameter definitions. The library is included in the ESCM integration package (*oscm-integration-pack.zip*). From the `readme.htm` file of the integration package, you can directly access the library.

Next Steps

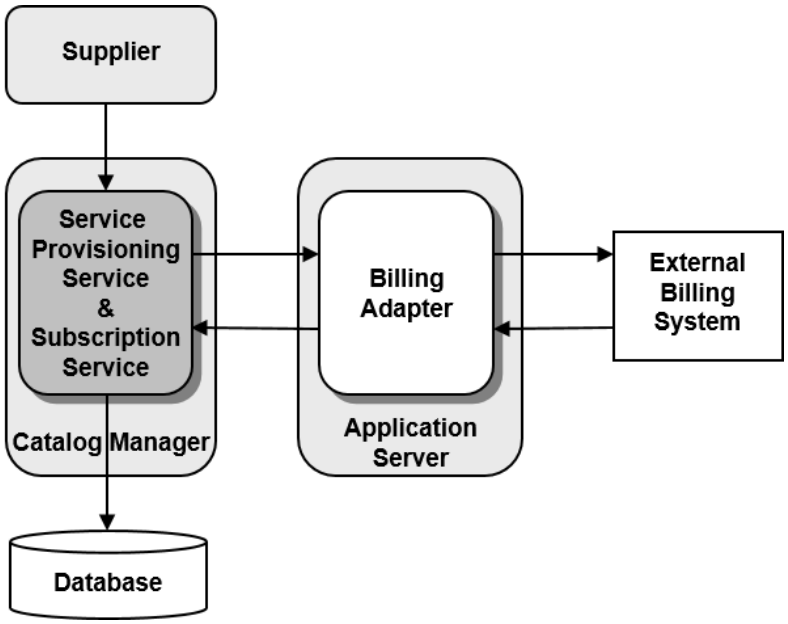
To link the external parameter configuration tool with ESCM, the supplier has to specify the URL of the tool in the service definition. The URL is not validated by ESCM, neither for syntax nor for semantic. The supplier has to ensure that the script does not introduce any security leaks. For details, refer to the *Supplier's Guide*.

6 Integrating an External Billing System

ESCM comes with its own, native billing system for managing price models and calculating costs for services. However, it can also be integrated with one or more external billing systems. This is suitable, for example, when offering services from external providers which have their own pricing and billing facilities, such as Amazon Web Services (AWS). It is also useful when prices and costs for services need to be based on metering information that is not available in ESCM.

By integrating an external billing system, its price models can be imported and displayed for marketable services and subscriptions in ESCM. The calculation of the costs for using these services and subscriptions as well as the billing and payment processing takes place in the external billing system. ESCM does not consider these costs in any calculations, discounts, revenue shares, or reports.

The connection between ESCM and an external billing system is established by means of a **billing adapter**. Upon request of ESCM, the billing adapter gathers and returns the appropriate information from the external billing system, usually a price model for a service or subscription.



The integration and usage of an external billing system involves the following basic tasks:

- **Development and deployment of a billing adapter** that connects ESCM with the external billing system. This task is carried out by a developer in tight cooperation with the supplier of the services that are to use the external billing system. The details of this task are described in the sections below.
- **Registration of the billing adapter** in ESCM. This task is carried out by the operator of ESCM. For details, refer to the *Operator's Guide*.
- **Specification of the external billing system** at one or more technical services on which the marketable services using the billing system are to be based. This task is carried out by the responsible technology provider. For details, refer to the *Technology Provider's Guide*.

- **Import of price models** from the external billing system for usage with marketable services, specific customers, or subscriptions. This task is carried out by the responsible supplier. For details, refer to the *Supplier's Guide*.
- **Display of the imported price models** at the corresponding services and subscriptions on the marketplaces and in the administration portal. This can be done by users with the appropriate authorities.

6.1 Billing Adapters

A billing adapter is needed to:

- Establish a connection between ESCM and an external billing system. Each billing system requires a separate adapter.
- Retrieve price models from the external billing system and return them to ESCM when ESCM sends a corresponding import request triggered by a supplier. In addition, a billing adapter can be extended to upload external price models for subscriptions to ESCM without a previous request.

A billing adapter must be implemented as an EJB (Enterprise Java Bean) with a remote interface. ESCM usually acts as a remote client. It obtains a reference to the remote interface by means of JNDI (Java Naming and Directory Interface) lookup. For this to be possible, the billing adapter needs to be registered and configured in ESCM by the operator.

The providers of the billing adapter are responsible for its deployment on an application server and for ensuring the communication with ESCM on the one side and the billing system on the other. This may, for example, involve the appropriate configuration of firewalls and access paths.

A billing adapter must implement the interfaces and methods of the billing plug-in API (`org.oscm.billing.external` package). This API and its documentation are included in the ESCM integration package, `oscm-integration-pack.zip`.

The ESCM integration package also contains a sample billing adapter. This adapter works with a simple file-based billing system which is included in the sample. For details of how to deploy and use the sample billing system and adapter, refer to the documentation that comes with them.

6.2 Implementing a Billing Adapter

The following sections provide details and hints for developing a billing adapter by implementing the interfaces of the billing plug-in API:

- `BillingPluginService`
- `PriceModelPluginService`

BillingPluginService

The `BillingPluginService` must be implemented for establishing and testing the connection with the external billing system. The `testConnection` method is invoked by ESCM when an operator configures the external billing system and clicks the corresponding button in the administration portal. The billing adapter must take the necessary actions and return the corresponding response to ESCM.

PriceModelPluginService

The `PriceModelPluginService` must be implemented for retrieving price models from the external billing system and returning them to ESCM. The `getPriceModel` method is invoked

by ESCM when a supplier imports a price model for a service, customer, or subscription in the administration portal.

In the method parameters, ESCM passes the context and one or more locales for which the price model is to be returned. The context provides details of the ESCM element for which the billing adapter needs to return the appropriate price model.

For a **service price model**, the context consists of:

- `SERVICE_ID`
- `SERVICE_NAME`
- `SERVICE_PARAMETERS`

For a **customer price model**, the context consists of:

- `SERVICE_ID`
- `SERVICE_NAME`
- `SERVICE_PARAMETERS`
- `CUSTOMER_ID`
- `CUSTOMER_NAME`

For a **subscription price model**, the context consists of:

- `SUBSCRIPTION_ID`

Based on the information in the context and the desired locales, the billing adapter must retrieve the required data from the external billing system, build a price model, and return it to ESCM in a `PriceModel` object. A price model is a file in PDF format. For a service, the billing adapter can return a tag in addition to this file. A tag is a string with short information, such as a price range, which is to be shown directly at the service on the marketplace.

In ESCM, the price model is stored with the element and for the locales for which it was requested. The stored price model is displayed to users upon request.

6.3 Uploading Subscription Price Models

A billing adapter can be extended to upload external price models for subscriptions to ESCM without a previous request from ESCM. This is accomplished by making the billing adapter work as a JMS client and pushing JMS messages with subscription price models into the remote queue of ESCM.

This upload mechanism may be useful, for example, if the price model in the external billing system changes during the life time of a subscription. The billing adapter could check for such changes at regular intervals or the billing system could notify the adapter accordingly. The environment involving ESCM, the billing adapter, and the external billing system must be set up and configured in a way to allow for the passing of the required notifications and JMS messages.

To make a billing adapter work as a JMS client and push JMS messages with price models to the remote queue of ESCM, perform the following steps:

1. Create the initial context for the remote JMS server.

```
Properties jndiProperties = new Properties();
InitialContext context = new InitialContext(jndiProperties);
```

The `jndiProperties` must contain the provider URL of ESCM:

`mq://<host>:8176`

`mq` is the Message Queue protocol of the application server Message Queue software (JMS provider for the application server).

`<host>` is the host where ESCM is deployed.

2. Retrieve the queue connection factory through JNDI lookup and create a connection.

```
Object lookup = context.lookup("jms/bss/taskQueueFactory");
ConnectionFactory conFactory =  ConnectionFactory.class.cast(lookup);
Connection connection = conFactory.createConnection();
```

3. Retrieve the queue in the remote JMS provider through JNDI lookup.

```
Queue queue = (Queue)cntxt.lookup("jms/bss/taskQueue");
```

4. Initialize the communication session.

```
Session session = connection.createSession(false,
 Session.AUTO_ACKNOWLEDGE);
```

5. Create and send the message with the subscription price model.

```
MessageProducer producer = session.createProducer(queue);
            ObjectMessage msg = session.createObjectMessage();
            msg.setObject(priceModel);
            producer.send(msg);
```

The price model is a `PriceModel` object of the billing plug-in API. The context sent in the price model must contain values for:

- `SUBSCRIPTION_ID`
- `TENANT_ID`

7 Integrating Certificates for Trusted Communication

Certificates are required for ESCM to allow for trusted communication between ESCM and an application underlying a technical service, a payment service provider (PSP), an Identity Provider (IdP) or a Security Token Service (STS).

The following organizations are involved when using certificates:

- Platform operator
- Technology providers integrating their applications with ESCM.
- PSPs whose services are to be integrated with ESCM for invoicing and payment collection.
- Organizations integrating an external process control system.
- Any other organization using Web service calls to or from ESCM

7.1 Introduction

Web service calls coming from ESCM (e.g. for provisioning application instances for services, or for integrating a process control system) or sent to it (e.g. by an application such as a PSP system) can be secured with SSL. SSL is used for authentication and for encryption at the transport level.

Every HTTPS connection involves a client and a server. Depending on the calling direction, ESCM can act as a server (Web service calls to ESCM) or as a client (Web service calls from ESCM).

Every application integrated with ESCM or accessing the platform services acts as a Web service client. ESCM acts as the server, and the client must provide its authenticating data to the server:

The caller sends the key or ID and password of an ESCM user. SOAP-based calls address the ESCM Web services with the `BASIC` suffix. This mechanism does not apply certificates for user authentication. However, certificates are involved because the communication between the client and ESCM must be secured using TLS/SSL and HTTPS. You should also configure your network's firewall to block JNDI lookups from the outside when using basic authentication.

Note: Both, the server and the client certificates must be created and signed using the same JRE/JDK. Otherwise, the communication may fail.

Terminology

ESCM uses an **X.509** certificate to prove the identity of an entity. This certificate is always used to prove the server's identity and optionally to prove the client's identity.

A certificate has a **subject** which usually identifies the owner of the certificate, and an **issuer** who signed the certificate. A certificate also includes a validity period. Cryptographic algorithms ensure that the information contained in the certificate cannot be changed without breaking the signature of the certificate.

The subject as well as the issuer is given as a **distinguished name (DN)** consisting of a list of key-value pairs. One of the standardized keys is called **common name (CN)**. The CN is of particular importance to HTTPS servers: The CN must contain the server's domain; otherwise the client will refuse the connection.

The process of issuing a certificate for another entity is called **signing**. Certificates always form a chain up to a certain **root certificate**. In a root certificate, the subject and the issuer are one and the same entity. Such certificates are called "**self-signed**".

Signing certificates or proofing that someone is the owner of a certificate requires the possession of the corresponding **private key**. While certificates can be distributed to other parties, special care must be taken to keep the private key secret.

Each client and server may have a keystore and a truststore. A **keystore** is used to keep certificates along with the corresponding private key. This means that a keystore is used to prove your own identity or to sign certificates. A **truststore** contains public certificates of other entities.

7.2 Requirements for Web Service Calls from ESCM

For the provisioning of application instances and for integrating external process control or PSP systems, ESCM calls other Web services which can be addressed by HTTPS. In this scenario, ESCM is the Web service client while the other entity is the HTTPS server. The following requirements must be fulfilled to establish a connection to the server:

- The server must present a valid certificate: The CN (common name) must correspond to the server's domain name and it must be valid at the time of calling.
- The client (ESCM) must trust the server's certificate. To this end, the server's certificate must have been imported into the client's truststore, or the client's truststore must contain a root certificate with a valid signing chain to the certificate presented by the server.

7.3 Requirements for Web Service Calls to ESCM

ESCM provides Web services (platform services) that can be called by other systems, such as an external process control or a PSP system. In this scenario, ESCM is the HTTPS server while the other system is a Web service client. The following requirements must be met to establish a connection to ESCM:

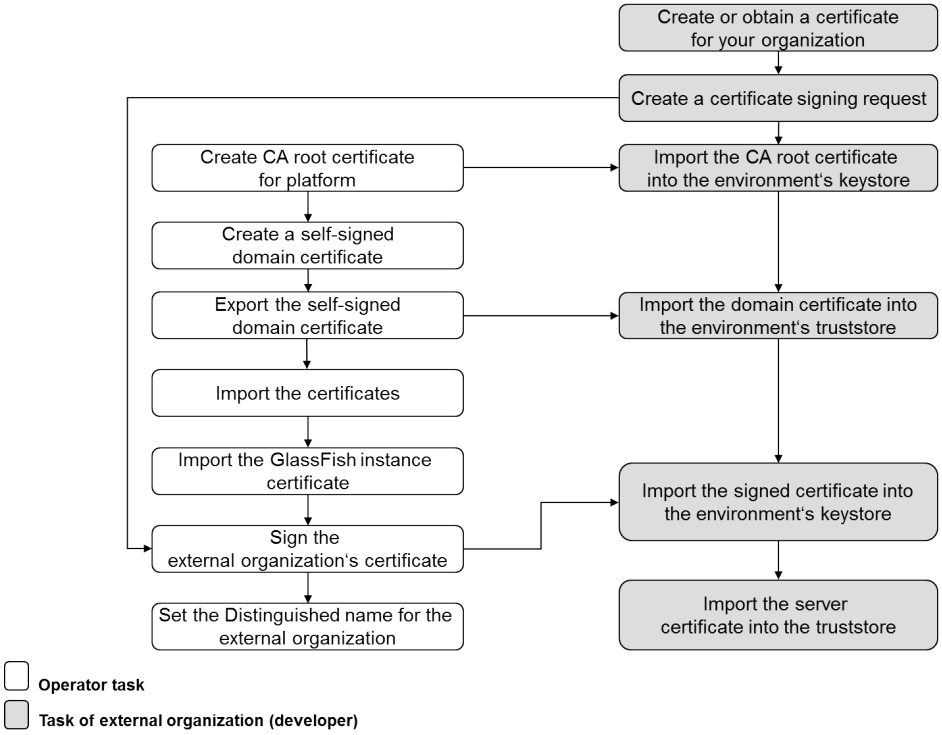
- The ESCM server must present a valid certificate: The CN (common name) must correspond to the server's domain name and it must be valid at the time of calling.
- The clients must trust the ESCM server certificate for SSL connections: The clients' truststore must contain the ESCM server certificate.
- In case of certificate-based authentication, ESCM must trust the client's certificate: The ESCM truststore must contain a certificate with a valid signing chain to the certificate presented by the client.

7.4 Certificate Integration Procedures

For implementing a secure ESCM environment where the nodes trust each other, the usage of custom root certificates is recommended. The basic procedure is as follows:

The ESCM operator creates the root certificate. This certificate is imported to the ESCM truststore as well as handed out to all technology providers or other organizations calling ESCM Web services.

The following figure illustrates the process and tasks involved in using custom root certificates for secure communication:



As a developer who integrates an application or external system with ESCM and wants to make use of certificates, you need to perform the tasks depicted in gray. These tasks are described in detail in the subsequent sections. The operator tasks are described in the *Operator's Guide*.

7.4.1 Creating a Certificate

The following command creates a certificate:

```
%JAVA_HOME%\bin\keytool -genkey -alias tpcert -keysize 1024
-keystore keystore.jks
```

You will be prompted for a password for the `keystore.jks` keystore that will be created containing the certificate identified by the alias `tpcert`. In addition, you need to specify detailed information on your organization and its location.

7.4.2 Importing the Signed Certificates

When the signing process is completed, you will receive the following from the ESCM operator:

- CA public (root) certificate for ESCM
- Self-signed domain certificate
- Your own, signed certificate

You have to import the certificates into your application server's keystore and truststore as follows:

1. Import the CA root certificate into the client's keystore by executing the following command:

```
%JAVA_HOME%\bin\keytool -import -alias cacert -file ca.crt
-keystore keystore.jks
```

2. Update your certificate (signed by the ESCM operator) in the keystore so that the certificate chain will be stored in the keystore:

```
%JAVA_HOME%\bin\keytool –import –file <bssDomain>
-keystore cacerts.jks –alias bssDomain
```

   For checking which entries are contained in your keystore, you can execute the following command:

```
%JAVA_HOME%\bin\keytool -list -keystore keystore.jks
```

   The result should look like

```
Keystore type: jks
Keystore provider: SUN

Your keystore contains two entries.

tpcert, 2010-05-21, keyEntry,
Certificate finger print (MD5):
 FA:E7:AF:99:33:A5:C9:8E:4F:28:50:04:08:0F:3F:4A
mykey, 2010-05-21, trustedCertEntry,
Certificate finger print (MD5):
 33:E5:B9:77:C3:34:8R:9F:34:99:1G:78:D5:3H:4B:22
```

3. Import the domain certificate into the client's truststore by executing the following command:

```
%JAVA_HOME%\bin\keytool –import –file <bssDomain>
-keystore cacerts.jks –alias bssDomain
```

7.4.3 Importing the ESCM Server Certificate

To secure the calls from ESCM to the provisioning service of an application, the ESCM server certificate has to be imported into the truststore valid for the provisioning service so that the service can identify ESCM as a client.

Execute the following command:

```
%JAVA_HOME%\bin\keytool -import -file bsssrv.crt -keystore
 keystore.jks -alias bsssrv
```

Appendix A: Customer Billing Data

The charges for the usage of a service in ESCM are calculated based on the price model defined for the service, customer, or subscription.

A supplier or reseller can export the billing data for one or several of his customers for a specific time. Suppliers also have access to the billing data of customers of broker organizations that sell their services. The exported data can be forwarded, for example, to an accounting system for further processing.

The result of the export is stored in an XML file, the customer billing data file. The billing data file conforms to the XML schema `BillingResult.xsd`, which can be found in the ESCM integration package.

The billing data file is named `<date>BillingData.xml`, where `<date>` represents the creation date.

This appendix describes the meaning of the elements and attributes that may occur in a billing data file.

BillingDetails

Top-level container element of a billing data file. For each subscription, a `BillingDetails` element is added to the billing data file.

A `BillingDetails` element contains the following subelements:

- `Period` (see *Period* on page 48)
- `OrganizationDetails` (see *OrganizationDetails* on page 49)
- `Subscriptions` (see *Subscriptions* on page 49)
- `OverallCosts` (see *OverallCosts* on page 62)

A `BillingDetails` element has the following attributes:

`key` - (optional, data type `long`) Unique identifier allowing, for example, accounting systems to relate billing data to an invoice. The billing data key is printed on the invoice. Suppliers and customers may use the billing data key to create a detailed billing report for an existing invoice or subscription. A supplier can retrieve the key from a billing report, a customer gets the billing data key from the corresponding invoice.

`timezone` - (required, data type `string`) Time zone based on the UTC time standard. It reflects the standard server time without daylight saving time. For example, 18:00:00 o'clock on June 1st in Berlin (UTC+1) will be output as follows in the XML file:

```
<BillingDetails timezone="UTC+01:00" key="31122">
  <Period startDate="1370106000000"
  startDateIsoFormat="2013-06-01T16:00:00.000Z" ..."/>
```

The time zone is relevant for price models with costs (see *PriceModel* on page 51).

Period

Specifies the billing period for which the data is exported. The start and end time of the billing period are output according to the start day of the billing period which was defined by the supplier or reseller.

A `Period` element has the following attributes:

- **`startDate`** - (data type `long`) Start time of the period. The start time is specified in milliseconds, the starting point for the calculation is `1970-01-01, 00:00`.

- **startDateIsoFormat** - (optional, data type `dateTime`) Same as `startDate`, but specified in ISO 8601 format (`YYYY-MM-DDThh:mm:ss.fffZ`) .
- **endDate** - (data type `long`) End time of the period. The end time is specified in milliseconds, the starting point for the calculation is `1970-01-01, 00:00`.
- **endDateIsoFormat** - (optional, data type `dateTime`) Same as `endDate`, but specified in ISO 8601 format (`YYYY-MM-DDThh:mm:ss.fffZ`).

Example:

```
<Period startDateIsoFormat="2012-08-31T22:00:00.000Z"
  startDate="1346450400000"
  endDateIsoFormat="2012-09-30T22:00:00.000Z"
  endDate="1349042400000"/>
```

OrganizationDetails

Provides details of the customer for which the billing data have been exported. The details may include a `Udas` element with custom attributes that store additional information on the customer organization.

An `OrganizationDetails` element contains the following subelements:

Email

Specifies the email address of the organization (data type `string`).

Name

Specifies the name of the organization (data type `string`).

Address

Specifies the address of the organization (data type `string`).

Paymenttype

Specifies the payment type used for subscriptions of the organization (data type `string`).

Example:

```
<BillingDetails key="10002" timezone="UTC+01:00">
...
  <OrganizationDetails>
    <Email>info@company.com</Email>
    <Name>company</Name>
    <Address>Street</Address>
    <Paymenttype>INVOICE</Paymenttype>
  </OrganizationDetails>
...
</BillingDetails>
```

Subscriptions

Contains the billing data for the subscriptions of the customer which are relevant for the current billing period.

For every subscription of an organization, the `Subscriptions` element contains a `Subscription` element. In this element, the costs of the affected subscription are specified.

A `Subscription` element has the following attributes:

- **id** - (required, data type `string`) Unique subscription name.

- **purchaseOrderNumber** - (data type `string`) Optional reference number as specified by the customer when subscribing to a service.

A `Subscription` element contains a `PriceModel` element with the billing data for the price model of the subscription (see *PriceModel* on page 51). A `Udas` element with custom attributes that store additional information on the subscription may also be included (see *Udas* on page 50).

If a subscription is assigned to an organizational unit at the end of the billing period, the `Subscription` element also contains an `OrganizationalUnit` element with the following attributes:

- **name** - (required, data type `string`) Required name of the organizational unit to which the subscription is assigned.
- **referenceID** - (optional, data type `string`) Optional reference ID of the organizational unit to which the subscription is assigned.

Be aware that the organizational units to which the subscription may have been assigned before the generation of the billing data are not shown.

Example:

```
<BillingDetails key="10002" timezone="UTC+01:00">
...
  <Subscriptions>
    <Subscription id="Mega Office Basic" purchaseOrderNumber="12345">
      <OrganizationalUnit name="ProjectTeam" referenceID="123abc"/>
      <PriceModels>
        <PriceModel calculationMode="PRO_RATA" id="14001">
...
        </PriceModel>
      </PriceModels>
    </Subscription>
  </Subscriptions>
...
</BillingDetails>
```

Udas

Contains custom attributes that store additional information on an organization or subscription. This could be, for example, the profit center to which a customer's revenue is to be accounted.

A `Udas` element may be included in an `OrganizationDetails` or a `Subscription` element.

For every custom attribute, a `Uda` element is included in the `Udas` element.

A `Uda` element has the following attributes:

- **id** - (required, data type `string`) ID of the custom attribute.
- **value** - (required, data type `string`) Value of the custom attribute.

Example:

```
<BillingDetails key="10002" timezone="UTC+01:00">
...
  <Subscriptions>
    <Subscription id="Mega Office Basic" purchaseOrderNumber="12345">
...
      <Udas>
        <Uda id="Profit Center" value="My Company"/>
      </Udas>
    </Subscription>
  </Subscriptions>
```

```
...
</BillingDetails>
```

PriceModel

Contains the billing data for a price model used to calculate the utilization charges for a subscription.

A `PriceModel` element is included in every subscription element. It contains the following subelements:

- `UsagePeriod` (see *UsagePeriod* on page 51)
- `GatheredEvents` (see *GatheredEvents* on page 52)
- `PeriodFee` (see *PeriodFee* on page 53)
- `UserAssignmentCosts` (see *UserAssignmentCosts* on page 54)
- `OneTimeFee` (see *OneTimeFee* on page 55)
- `PriceModelCosts` (see *PriceModelCosts* on page 55)
- `Parameters` (see *Parameters* on page 56)

A `PriceModel` element has the following attributes:

`id` - (required, data type `string`) Unique name identifying the price model.

`calculationMode` - (required, data type `string`) Cost calculation option of the price model. Can be set to one of the following values: `FREE_OF_CHARGE` (the service is free of charge), `PRO_RATA` (the costs are calculated exactly for the time a service is used, based on milliseconds), `PER_UNIT` (the costs are calculated based on fixed time units).

UsagePeriod

Specifies the actual period in which a price model is used for calculating the charges of a subscription.

A usage period usually begins when a customer subscribes to a service and ends when the subscription is terminated. In case a free trial period is defined for the service, the usage period begins when the free trial period has ended. When the customer upgrades or downgrades the subscription, a new usage period is started in which the price model of the new service is applied. If the customer changes elements that determine how the charges for the service are calculated (e.g. the number of assigned users, the service roles of the assigned users, or the parameter values), a new usage period is started in which the updated elements are applied.

A `UsagePeriod` element is contained in a `PriceModel` element.

A `UsagePeriod` element has the following attributes:

- **`startDate`** - (data type `long`) Start time of the period. The start time is specified in milliseconds, the starting point for the calculation is `1970-01-01, 00:00`.
- **`startDateIsoFormat`** - (optional, data type `dateTime`) Same as `startDate`, but specified in ISO 8601 format (`YYYY-MM-DDThh:mm:ss.fffZ`) .
- **`endDate`** - (data type `long`) End time of the period. The end time is specified in milliseconds, the starting point for the calculation is `1970-01-01, 00:00`.
- **`endDateIsoFormat`** - (optional, data type `dateTime`) Same as `endDate`, but specified in ISO 8601 format (`YYYY-MM-DDThh:mm:ss.fffZ`).

Example:

```
<PriceModel calculationMode="PRO_RATA" id="14001">
```

```
  <UsagePeriod endDate="1306879200000"
   endDateIsoFormat="2011-05-31T22:00:00.000Z"
   startDate="1304755088065"
   startDateIsoFormat="2011-05-07T07:58:08.065Z"/>
...
</PriceModel>
```

GatheredEvents

Specifies the costs for all chargeable events that occurred in the current usage period of the subscription. These include, for example, login and logout by users to the underlying application, the completion of specific transactions, or the creation or deletion of specific data. It depends on the implementation and integration of the underlying application which events are available.

A `GatheredEvents` element is contained in a `PriceModel` element.

A `GatheredEvents` element contains the following subelements:

- `Event`
- `GatheredEventsCosts`

Event

For every event, an `Event` element is included in the `GatheredEvents` element.

An `Event` element has the following attribute:

`id` - (required, data type `string`) Event ID as specified in the technical service definition.

An `Event` element contains the following subelements:

- `Description`
- `SingleCost`
- `NumberOfOccurence`
- `CostForEventType`

Description

Contains the description of the event.

SingleCost

Specifies the price for the event as defined in the price model. If an event has stepped prices, this element is omitted. A `SteppedPrices` element is included instead (see *SteppedPrices* on page 61).

A `SingleCost` element has the following attribute:

`amount` - (required, data type `decimal`) Price for a single event.

NumberOfOccurence

Specifies how often the event occurred.

A `NumberOfOccurence` element has the following attribute:

`amount` - (required, data type `long`) Number of times the event occurred.

CostForEventType

Specifies the total costs for the event in the billing period.

A `CostForEventType` element has the following attribute:

`amount` - (required, data type `decimal`) Total costs for the event. The total costs for an event are calculated from the singular costs (`SingleCost`) multiplied with the number of occurrences (`NumberOfOccurence`). If role-based costs and/or stepped prices are specified for events, these

costs are added (see *RoleCosts* on page 60 and *SteppedPrices* on page 61). The value is rounded to two decimal places.

GatheredEventsCosts

Specifies the total costs for all events in the current `GatheredEvents` element.

A `GatheredEventsCosts` element has the following attribute:

`amount` - (required, data type `decimal`) Total costs for events. The value is rounded to two decimal places.

Example:

```
<PriceModel calculationMode="PRO_RATA" id="14001">
...
  <GatheredEvents>
    <Event id="USER_LOGOUT_FROM_SERVICE">
      <Description xml:lang="en">Logout from the service.</Description>
      <SingleCost amount="100.00"/>
      <NumberOfOccurrence amount="3"/>
      <CostForEventType amount="300.00"/>
    </Event>
...
    <GatheredEventsCosts amount="1200.00"/>
  </GatheredEvents>
...
</PriceModel>
```

PeriodFee

Specifies the costs for using the subscription in the given usage period.

For each subscription, a charge can be defined that a customer has to pay on a recurring basis. Monthly, weekly, daily, or hourly periods are supported. The recurring charge for a subscription is independent of the amount of users, events, or other usage data.

The calculation of the charges depends on the cost calculation option which was chosen for the price model (see *PriceModel* on page 51 for details).

A `PeriodFee` element is contained in a `PriceModel` element.

A `PeriodFee` element has the following attributes:

- **`basePeriod`** - (required, data type `string`) Period on which the charges are based. Can be set to one of the following values: `MONTH`, `WEEK`, `DAY`, `HOUR`.
- **`basePrice`** - (required, data type `decimal`) Recurring charge per base period according to the price model.
- **`factor`** - (required, data type `decimal`) Factor used to calculate the period fee for the subscription. The factor is calculated from the usage period of the subscription divided by the base period (`basePeriod`). The recurring charge is multiplied with this factor to calculate the total costs (`price`).
- **`price`** - (required, data type `decimal`) Total period fee for the subscription. This value is calculated from the recurring charge (`basePrice`) multiplied with the factor (`factor`). The value is rounded to two decimal places.

Example:

```
<PriceModel calculationMode="PRO_RATA" id="14001">
...
  <PeriodFee basePeriod="MONTH" basePrice="10.00"
 factor="0.4020212567204301" price="4.02"/>
```

```
...
</PriceModel>
```

UserAssignmentCosts

Specifies the costs for the user assignments to the subscription.

For the users assigned to a subscription, a charge can be defined that a customer has to pay on a recurring basis. Monthly, weekly, daily, or hourly periods are supported. The charge depends on the amount of time units one or more users are assigned to the subscription. This type of charge can only be defined for services with the login or user access type.

The recurring charge for users is independent of the recurring charge per subscription or other usage data.

For this type of charge, stepped prices can be applied: Recurring charges can be defined that depend on the sum of the time units of all user assignments.

The calculation of the charges depends on the cost calculation option which was chosen for the price model (see *PriceModel* on page 51 for details).

With per time unit calculation, the costs for a time unit in which a user is assigned to a subscription are always fully charged. There is no difference in the costs between a user who is assigned from the start until the end of the time unit and a user who is assigned for a part of the time unit only. A time unit is charged only once if a user is deassigned from and re-assigned to a subscription within the same time unit. Yet, canceling an assignment, deleting the user, and then creating a new user with the same user ID is treated as if two different users are assigned to the subscription. The time unit is charged twice, accordingly.

A `UserAssignmentCosts` element is contained in a `PriceModel` element.

A `UserAssignmentCosts` element has the following attributes:

- **`basePeriod`** - (optional, data type `string`) Period on which the charges are based. Can be set to one of the following values: `MONTH`, `WEEK`, `DAY`, `HOUR`.
- **`basePrice`** - (optional, data type `decimal`) Recurring charge for users per base period according to the price model. If the charge for users has stepped prices, this attribute is omitted.
- **`factor`** - (optional, data type `decimal`) Factor used to calculate the costs for the user assignments. The factor is calculated by summing up the factors for each user specified in the `UserAssignmentCostsByUser` element. The recurring charge (`basePrice`) is multiplied with this factor to calculate the costs (`price`).
- **`numberOfUsersTotal`** - (optional, data type `long`) Number of users assigned to the subscription in the usage period.
- **`total`** - (data type `decimal`) Total costs for the user assignments including role-based costs and stepped prices. The value is rounded to two decimal places. For details on role-based costs and stepped prices, refer to *RoleCosts* on page 60 and *SteppedPrices* on page 61.
- **`price`** - (optional, data type `decimal`) Costs for the user assignments. This value is calculated from the recurring charge (`basePrice`) multiplied with the factor (`factor`). The value is rounded to two decimal places.

A `UserAssignmentCosts` element contains the following subelement. If role-based costs and/or stepped prices are specified, a `RoleCosts` element and/or `SteppedPrices` element is also present (see *RoleCosts* on page 60 and *SteppedPrices* on page 61).

UserAssignmentCostsByUser

Specifies the fraction of the usage period a user was assigned to the subscription.

A `UserAssignmentCostsByUser` element has the following attributes:

- **`factor`** - (required, data type `decimal`) Fraction of the usage period the given user was assigned to the subscription. The factors of the single user assignments are summed up to calculate the total costs for the user assignments.
- **`userId`** - (required, data type `string`) User ID.

Example:

```
<PriceModel calculationMode="PRO_RATA" id="18000">
...
  <UserAssignmentCosts basePeriod="MONTH" basePrice="19.00"
   factor="0.5337726052867383" numberOfUsersTotal="2"
   total ="50.00" price="10.14">
     <UserAssignmentCostsByUser factor="1.0499215949820788E-4"
     userId="admin"/>
     <UserAssignmentCostsByUser factor="0.5336676131272401"
     userId="miller"/>
   </UserAssignmentCosts>
...
</PriceModel>
```

OneTimeFee

Specifies the one-time fee for the subscription.

A one-time fee defines the amount a customer has to pay for a subscription in the first billing period of the subscription. It is added to the total charges for the first billing period. It is independent of the number of users, events, or other usage data.

If a one-time fee is defined for a service to which a customer upgrades or downgrades a subscription, it is added to the total charges for the customer, even if the service from which the customer migrates also defines a one-time fee.

A `OneTimeFee` element is contained in a `PriceModel` element.

A `OneTimeFee` element has the following attributes:

- **`amount`** - (required, data type `decimal`) Total costs for the one-time fee. The value is rounded to two decimal places.
- **`baseAmount`** - (required, data type `decimal`) One-time fee as defined in the price model.
- **`factor`** - (required, data type `long`) Factor used for calculating the one-time fee. Since this charge occurs only once, the factor is 1 for the first billing period, and 0 in case the one-time fee has already been charged in a previous billing period.

Example:

```
<PriceModel calculationMode="PRO_RATA" id="14001">
...
  <OneTimeFee amount="10.00" baseAmount="10.00" factor="1"/>
...
</PriceModel>
```

PriceModelCosts

Specifies the total costs for the subscription in the current usage period.

A `PriceModelCosts` element is contained in a `PriceModel` element.

A `PriceModelCosts` element has the following attributes:

- **`currency`** - (required, data type `string`) ISO code of the currency in which the costs are calculated.
- **`amount`** - (required, data type `decimal`) Total amount of the costs for the subscription. The value is rounded to two decimal places.

Example:

```
<PriceModel calculationMode="PRO_RATA" id="14001">
...
  <PriceModelCosts currency="EUR" amount="990.00"/>
</PriceModel>
```

Parameters

Specifies the costs for parameters defined for the service underlying the subscription.

A price model can define prices for service parameters and options. It depends on the implementation and integration of the underlying application whether and which parameters and options are available.

A price can be defined for every parameter and option, and the price can be charged per subscription or per user assigned to the subscription. Numeric parameters are a multiplier for the price. For boolean parameters, the multiplier is `1` if the value is `true`. In all other cases, the multiplier is `0`.

The calculation of charges for parameters and options depends on the cost calculation option which was chosen for the price model (see *PriceModel* on page 51 for details).

If the charges for a subscription are calculated per time unit and a customer changes a parameter value within a time unit, the affected time unit is charged pro rata. This means that the customer is charged exactly for the time each parameter value is set.

For numeric parameters, stepped prices can be applied per subscription: Different prices can be defined depending on the parameter values.

The prices for parameters and options are independent of other price model elements.

A `Parameters` element is contained in a `PriceModel` element.

A `Parameters` element contains the following subelements:

- `Parameter`
- `ParametersCosts`

Parameter

For every parameter, a `Parameter` element is included in the `Parameters` element.

A `Parameter` element has the following attribute:

`id` - (required, data type `string`) Parameter ID.

A `Parameter` element contains the following subelements:

- `ParameterUsagePeriod`
- `ParameterValue`
- `PeriodFee`
- `UserAssignmentCosts`
- `ParameterCosts`
- `Options`

ParameterUsagePeriod

Specifies the actual usage period for the parameter.

The usage period for a parameter begins when a customer subscribes to the service with the given parameter definition in the price model and ends when the subscription is terminated.

In case a free trial period is defined for the service, the usage period for the subscription begins when the free trial period has ended. If a parameter value is changed, a new usage period is started in which the updated value is applied for calculating the costs.

A `ParameterUsagePeriod` element has the following attributes:

- **`startDate`** - (data type `long`) Start time of the period. The start time is specified in milliseconds, the starting point for the calculation is `1970-01-01, 00:00`.
- **`startDateIsoFormat`** - (optional, data type `dateTime`) Same as `startDate`, but specified in ISO 8601 format (`YYYY-MM-DDThh:mm:ss.fffZ`) .
- **`endDate`** - (data type `long`) End time of the period. The end time is specified in milliseconds, the starting point for the calculation is `1970-01-01, 00:00`.
- **`endDateIsoFormat`** - (optional, data type `dateTime`) Same as `endDate`, but specified in ISO 8601 format (`YYYY-MM-DDThh:mm:ss.fffZ`).

ParameterValue

Specifies the costs and data type for the parameter.

A `ParameterValue` element has the following attributes:

- **`amount`** - (required, data type `string`) Costs for the parameter as defined in the price model.
- **`type`** - (required, data type `string`) Data type of the parameter. Can be set to one of the following values: `BOOLEAN`, `INTEGER`, `LONG`, `STRING`, `ENUMERATION`, `DURATION`.

PeriodFee

Specifies the costs for using the parameter in the given usage period. If a parameter has stepped prices, a `SteppedPrices` element is included in the `PeriodFee` element.

A `PeriodFee` element has the following attributes:

- **`basePeriod`** - (required, data type `string`) Period on which the charges are based. Can be set to one of the following values: `MONTH`, `WEEK`, `DAY`, `HOUR`.
- **`basePrice`** - (optional, data type `decimal`) Recurring charge per base period according to the price model. If a parameter has stepped prices, this attribute is omitted.
- **`factor`** - (required, data type `decimal`) Factor used to calculate the costs for using the parameter. The factor is calculated from the usage period divided by the base period (`basePeriod`). The recurring charge (`basePrice`) is multiplied with this factor to calculate the costs (`price`).
- **`price`** - (required, data type `decimal`) Costs for using the parameter. This value is calculated from the recurring charge (`basePrice`) multiplied with the factors (`factor` and `valueFactor`). The value is rounded to two decimal places.
- **`valueFactor`** - (required, data type `float`) Factor to calculate the total costs for using the parameter depending on the parameter value. The recurring charge (`basePrice`) is multiplied with this factor to calculate the costs (`price`). This factor is set depending on the data type of the parameter. For numeric parameters it is set to the value of the parameter. For boolean parameters, the factor is set to `1` if the value is `true`. In all other cases, the factor is set to `0`.

UserAssignmentCosts

Specifies the costs for the parameter related to the user assignments of the subscription based on the price per user for the parameter as defined in the price model. If costs for service roles are

defined, a `RoleCosts` element is included in the `UserAssignmentCosts` element (see *RoleCosts* on page 60 for details).

A `UserAssignmentCosts` element has the following attributes:

- **`basePeriod`** - (required, data type `string`) Period on which the charges are based. Can be set to one of the following values: `MONTH`, `WEEK`, `DAY`, `HOUR`.
- **`basePrice`** - (required, data type `decimal`) Recurring charge for users per base period for the parameter according to the price model.
- **`factor`** - (required, data type `decimal`) Factor used to calculate the costs for using the parameter. The factor is calculated from the parameter usage period divided by the base period (`basePeriod`) multiplied with the number of users. The recurring charge (`basePrice`) is multiplied with this factor to calculate the costs (`price`).
- **`price`** - (required, data type `decimal`) Costs for using the parameter. This value is calculated from the recurring charge (`basePrice`) multiplied with the factors (`factor` and `valueFactor`). The value is rounded to two decimal places. If stepped prices are defined for user assignments, the costs are given in the `price` attribute.
- **`total`** - (data type `decimal`) Total costs for using the parameter including role-based costs. The value is rounded to two decimal places. For details on role-based costs, refer to *RoleCosts* on page 60.
- **`valueFactor`** - (required, data type `float`) Factor to calculate the total costs for using the parameter depending on the parameter value. The recurring charge (`basePrice`) is multiplied with this factor to calculate the costs (`price`). This factor is set depending on the data type of the parameter. For numeric parameters it is set to the value of the parameter. For boolean parameters, the factor is set to `1` if the value is `true`. In all other cases, the factor is set to `0`.

ParameterCosts

Specifies the total costs for using the parameter.

A `ParameterCosts` element has the following attribute:

`amount` - (required, data type `decimal`) Total costs for the parameter calculated by summing up the costs specified in the `PeriodFee` and the `UserAssignmentCosts` element for the parameter and its options. If role-based costs and/or stepped prices are specified for the parameter, these are added (see *RoleCosts* on page 60 and *SteppedPrices* on page 61). The value is rounded to two decimal places.

ParametersCosts

Specifies the total costs for all parameters.

A `ParametersCosts` element has the following attribute:

`amount` - (required, data type `decimal`) Total costs for the parameters calculated by summing up the costs of the individual parameters as specified in the `ParameterCosts` elements. The value is rounded to two decimal places.

Example:

```
<Parameters>
...
  <Parameter id="MAX_FOLDER_NUMBER2">
    <ParameterUsagePeriod endDate="1306879200000"
     endDateIsoFormat="2011-05-31T22:00:00.000Z"
     startDate="1304755088065"
     startDateIsoFormat="2011-05-07T07:58:08.065Z"/>
    <ParameterValue amount="200" type="INTEGER"/>
    <PeriodFee basePeriod="MONTH" basePrice="0.00"
     factor="0.5337789669205496" price="0.00" valueFactor="200.0"/>
```

```
    <UserAssignmentCosts basePeriod="MONTH" basePrice="0.00"
     factor="0.5337726052867383" total ="0.00"
     price="0.00" valueFactor="200.0"/>
    <ParameterCosts amount="0.00"/>
  </Parameter>

...
  <ParametersCosts amount="600.00"/>
</Parameters>
```

Options

Specifies the costs for parameter options.

An `Options` element is contained in a `Parameter` element.

For every option, an `Option` element is included in the `Options` element.

An `Option` element has the following attribute:

`id` - (required, data type `string`) Option ID.

An `Option` element contains the following subelements:

- `PeriodFee`
- `UserAssignmentCosts`
- `OptionCosts`

PeriodFee

Specifies the costs for using the parameter option in the given usage period.

A `PeriodFee` element has the following attributes:

- **`basePeriod`** - (required, data type `string`) Period on which the charges are based. Can be set to one of the following values: `MONTH`, `WEEK`, `DAY`, `HOUR`.
- **`basePrice`** - (required, data type `decimal`) Recurring charge per base period according to the price model.
- **`factor`** - (required, data type `decimal`) Factor used to calculate the costs for using the parameter option. The factor is calculated from the usage period divided by the base period (`basePeriod`). The recurring charge (`basePrice`) is multiplied with this factor to calculate the total costs (`price`).
- **`price`** - (required, data type `decimal`) Costs for the parameter option. This value is calculated from the recurring charge (`basePrice`) multiplied with the factor (`factor`), and is rounded to two decimal places.

UserAssignmentCosts

Specifies the costs for the parameter option related to the user assignments of the subscription based on the price per user for the option as defined in the price model. If costs for service roles are defined, a `RoleCosts` element is included in the `UserAssignmentCosts` element (see *RoleCosts* on page 60).

A `UserAssignmentCosts` element has the following attributes:

- **`basePeriod`** - (required, data type `string`) Period on which the charges are based. Can be set to one of the following values: `MONTH`, `WEEK`, `DAY`, `HOUR`.
- **`basePrice`** - (required, data type `decimal`) Recurring charge for users per base period for the parameter option according to the price model.
- **`factor`** - (required, data type `decimal`) Factor used to calculate the costs for using the parameter option. The factor is calculated from the usage period divided by the base period

(basePeriod). The recurring charge (basePrice) is multiplied with this factor to calculate the costs (price).

- **total** - (data type decimal) Total costs for using the parameter option including role-based costs. The value is rounded to two decimal places. For details on role-based costs, refer to *RoleCosts* on page 60.
- **price** - (required, data type decimal) Costs for using the parameter option. This value is calculated from the recurring charge (basePrice) multiplied with the factor (factor). The value is rounded to two decimal places. If stepped prices are defined for user assignments, the costs are given in the price attribute.

OptionCosts

Specifies the total costs for using the parameter option.

An OptionCosts element has the following attribute:

amount - (required, data type decimal) Total costs for the parameter option calculated by summing up the costs specified in the PeriodFee and the UserAssignmentCosts element. The value is rounded to two decimal places.

Example:

```
<Parameter id="MEMORY_STORAGE">
...
  <Options>
    <Option id="2">
      <PeriodFee basePeriod="MONTH" basePrice="100.00"
       factor="1.0" price="100.00"/>
      <UserAssignmentCosts basePeriod="MONTH" basePrice="0.00"
       factor="1.0" total ="0.00" price="0.00"/>
      <OptionCosts amount="100.00"/>
    </Option>
  </Options>
...
</Parameter>
```

RoleCosts

Specifies the costs for service roles.

If defined for the underlying application, roles can be used to grant specific privileges to different users. The roles are specified in the technical service definition as service roles. Service roles can be mapped to corresponding permissions in the application.

For each role, a price can be defined. This price is added to the base price per user in the cost calculation for a billing period.

The calculation of the charges for service roles depends on the cost calculation option which was chosen for the price model (see *PriceModel* on page 51 for details).

If the charges are calculated per time unit and the role assignment of a user is changed within a time unit, the affected time unit is charged pro rata. This means that the customer is charged exactly for the time each user role is assigned.

If the charges are calculated per time unit and a user with a specific role is removed from the subscription and assigned to it again with a different role in the same time unit, the customer is also charged for the time during which the user is not assigned to the subscription. This means that he is charged with the price for the first service role until the user is assigned to the subscription with the second service role.

A `RoleCosts` element is contained in a `UserAssignmentCosts` element (as subelement of the `PriceModel`, `Parameters`, or `Option` element).

A `RoleCosts` element has the following attribute:

`total` - (required, data type `decimal`) Total amount of costs for the service roles. The value is rounded to two decimal places.

For every service role, a `RoleCost` element is included in the `RoleCosts` element.

A `RoleCost` element has the following attributes:

- **`id`** - (required, data type `string`) ID of the service role.
- **`basePrice`** - (required, data type `decimal`) Recurring charge for the service role according to the price model.
- **`factor`** - (required, data type `decimal`) Factor used to calculate the costs for the service role. The factor is calculated as a fraction of the actual usage period. The recurring charge (`basePrice`) is multiplied with this factor to calculate the costs (`price`).
- **`price`** - (required, data type `decimal`) Costs for the service role. This value is calculated from the recurring charge (`basePrice`) multiplied with the factor (`factor`). The value is rounded to two decimal places.

Example:

```
<Parameter id="MEMORY_STORAGE">
...
  <RoleCosts total="0.00">
    <RoleCost basePrice="0.00" factor="0.4020087753882915"
     id="USER" price="0.00"/>
    <RoleCost basePrice="0.00" factor="0.8040175186678614"
     id="ADMIN" price="0.00"/>
  </RoleCosts>
...
</Parameter>
```

SteppedPrices

Specifies the stepped prices for a user assignment, event, or parameter.

Stepped prices allow for the definition of ranges for which different price factors apply. Step limits, i.e. the upper limits of ranges, can be set for:

- The **sum of the time units** users are assigned and work with a subscription in a billing period. For example, up to 10 hours one user is assigned to a subscription cost 10.00 € per hour, every additional hour the user is assigned costs 8.00 €.
- The **number of events** occurring in the usage of a subscription. For example, up to 10 file downloads cost 1.00 € per download, any additional download costs 0.50 €.
- **Values of numeric parameters**. For example, uploading up to 100 files costs 1.00 € per file, any additional upload costs 0.50 € per file.

Stepped prices are independent of any other price model elements.

A `SteppedPrices` element is contained in `UserAssignmentCosts` (as subelement of the `PriceModel` element), `Event`, and `PeriodFee` (as subelement of the `Parameters` element) elements.

A `SteppedPrices` element has the following attribute:

`amount` - (required, data type `decimal`) Summed up costs for all steps including the last one.

For every price step, a `SteppedPrice` element is included in a `SteppedPrices` element.

A `SteppedPrice` element has the following attributes:

- **`additionalPrice`** - (required, data type `decimal`) Summed up costs for the previous steps. The costs are calculated from the `limit`, `freeAmount` and `basePrice` attributes of the previous step (`(limit - freeAmount) * price`). The `additionalPrice` attribute of the first step always has a value of `0`. The value is rounded to two decimal places.
- **`basePrice`** - (required, data type `decimal`) Costs for the current step according to the price model.
- **`freeAmount`** - (required, data type `long`) Amount of units for the current step that are considered as a fixed discount, for example, the number of users that are free of charge. The value corresponds to the value of the `limit` attribute in the previous step. The `freeAmount` attribute of the first step always has a value of `0`.
- **`limit`** - (required, data type `string`) Step limit as defined in the price model.
- **`stepAmount`** - (optional, data type `decimal`) Summed up costs for the current step. These costs are calculated from the `basePrice` and `stepEntityCount` attributes of the current step. The value is rounded to two decimal places.
- **`stepEntityCount`** - (optional, data type `decimal`) Factor used to calculate the costs for the current step.

Example:

```
<PriceModel calculationMode="PRO_RATA" id="350001">
...
  <UserAssignmentCosts basePeriod="MONTH" factor="2.707940780619112"
 numberOfUsersTotal="4" price="1283.18">
    <SteppedPrices amount="1283.18">
      <SteppedPrice additionalPrice="0.00" basePrice="500.00"
       freeAmount="0" limit="2" stepAmount="1000.00"
 stepEntityCount="2"/>
      <SteppedPrice additionalPrice="1000.00" basePrice="400.00"
       freeAmount="2" limit="3" stepAmount="283.18"
       stepEntityCount="0.707940780619112"/>
      <SteppedPrice additionalPrice="1400.00" basePrice="300.00"
       freeAmount="3" limit="null" stepAmount="0.00"
 stepEntityCount="0"/>
    </SteppedPrices>
  </UserAssignmentCosts>
...
</PriceModel>
```

OverallCosts

Contains the total amount of the charges to be paid by a customer for all subscriptions in the current billing period. The costs are given in the currency specified in the price model.

If a discount was specified, the net amount of the costs is given in the `Discount` element (see *Discount* on page 64). If a VAT rate was defined, it is given in the `VAT` element (see *VAT* on page 63).

An `OverallCosts` element has the following attributes:

- **`netAmount`** - (required, data type `decimal`) Net costs after the net discount has been deducted from the original net costs (see *Discount* on page 64). The value is rounded to two decimal places.
- **`currency`** - (required, data type `string`) ISO code of the currency in which the costs are calculated.

- **grossAmount** - (required, data type `decimal`) Gross amount of the costs, calculated from the net costs (`netAmount`) plus VAT (see *VAT* on page 63). The value is rounded to two decimal places.

Example:

```
<BillingDetails key="10002" timezone="UTC+01:00">
...
<OverallCosts netAmount="900.00" currency="EUR" grossAmount="1053.00"/>

</OverallCosts>
</BillingDetails>
```

VAT

Specifies the VAT rate to be applied.

A supplier can define a basic VAT rate that applies by default to all prices for his customers. In addition to this basic VAT rate, country-specific or even customer-specific VAT rates can be defined. You can:

- Enable VAT rate support for your organization.
- Define a default VAT rate that applies to all prices for all customers.
- Define a country-specific VAT rate for every country where you want to sell your services.
- Define a customer-specific VAT rate, for example, in case a customer organization has a subsidiary located in another country than its parent organization.

The VAT rate settings have the following effects on the cost calculation for a customer:

- If VAT rate support is disabled, prices are calculated as net prices; no VAT is added to the overall costs.
- A customer-specific VAT rate takes priority over any default or country-specific VAT rate.
- The country-specific VAT rate for the country where the customer organization is located is applied to the cost calculation when no customer-specific VAT rate is defined.
- The default VAT rate is used in all other cases.

The VAT rate does not affect any price model elements. The calculated VAT amount is added to the overall costs and results in the gross price to be paid by a customer.

A `VAT` element is contained in the `OverallCosts` element.

A `VAT` element has the following attributes:

- **percent** - (required, data type `float`) VAT rate in percent, specified as a decimal number.
- **amount** - (required, data type `decimal`) Net amount of VAT to be added to the net costs (`netAmount` attribute of the `OverallCosts` element). The value is rounded to two decimal places.

Example:

```
<BillingDetails key="10002" timezone="UTC+01:00">
...
  <OverallCosts netAmount="900.00" currency="EUR" grossAmount="1053.00">
    <VAT percent="17.0" amount="153.00"/>
  </OverallCosts>
</BillingDetails>
```

Discount

Specifies the discount granted to the customer.

A discount can be defined for a customer which applies to all subscriptions of the customer to services. A discount may be valid as of the current or a future month. It can be restricted to a certain period of time. Before the time expires, the customer is notified by email so that he can react and contact the supplier.

The discount is defined as a percentage that is subtracted from the regular total price for a subscription. It is granted for all costs of a customer that incur in a billing period in which the discount is valid. It does not matter whether the discount is valid for the whole billing period or only a part of it.

A discount is completely independent of what a customer might purchase. If a discount is changed, the new discount is valid the next time the billing data is generated. Usually, a discount is only changed in agreement with the relevant customer.

A `Discount` element is contained in the `OverallCosts` element.

A `Discount` element has the following attributes:

- **`percent`** - (required, data type `float`) Percentage of costs to be deducted from the net costs, specified as a decimal number.
- **`discountNetAmount`** - (required, data type `decimal`) Net discount to be deducted from the original net costs (`netAmountBeforeDiscount`). The value is rounded to two decimal places.
- **`netAmountAfterDiscount`** - (required, data type `decimal`) Net costs after the net discount (`discountNetAmount`) has been deducted from the original net costs (`netAmountBeforeDiscount`). The value is rounded to two decimal places.
- **`netAmountBeforeDiscount`** - (required, data type `decimal`) Net costs before the net discount (`discountNetAmount`) has been deducted. The value is rounded to two decimal places.

Example:

```
<BillingDetails key="10002" timezone="UTC+01:00">
...
  <OverallCosts netAmount="900.00" currency="EUR" grossAmount="1053.00">
    <Discount percent="10.00" discountNetAmount="100.00"
     netAmountAfterDiscount="900.00"
     netAmountBeforeDiscount="1000.00" />
    <VAT percent="17.0" amount="153.00"/>
  </OverallCosts>
</BillingDetails>
```

Appendix B: Revenue Share Data

In extended usage scenarios, suppliers may involve brokers and resellers in selling their services. The brokers and resellers as well as the platform operator and the owners of the marketplaces on which the services are published, usually receive a share of the revenue for the services. ESCM calculates these revenue shares based on the billing data for the customers who use the services.

Suppliers, brokers, resellers, and marketplace owners can generate reports for their revenue shares and export the revenue share data for a specific time. Operators can export the data for all the suppliers, brokers, resellers, or marketplace owners known to their platform installation. The exported data can be forwarded, for example, to an accounting system for further processing.

The result of the export is stored in an XML file, the revenue data file. The file conforms to one of the following schemas, depending on the type of the revenue share data:

- `BrokerRevenueShareResult.xsd` : revenue share data for brokers
- `ResellerRevenueShareResult.xsd` : revenue share data for resellers
- `MPOwnerRevenueShareResult.xsd` : revenue share data for marketplace owners
- `SupplierRevenueShareResult.xsd` : revenue share data for suppliers

Each of these schemas includes the `BillingBase.xsd` with common definitions. All the schemas can be found in the ESCM integration package.

The XML files containing the revenue share data are named `<date>BillingData.xml`, where `<date>` represents the creation date.

This appendix describes the meaning of the elements and attributes that may occur in the different types of revenue share data file. The first section explains elements and attributes that are common to all revenue share data files. The subsequent sections describe the individual files.

B.1 Common Elements

The sections below describe the following elements that are common to all revenue share data files:

- `Period`
- `OrganizationData`

Period

Specifies the billing period for which the data is exported. The start and end time of the billing period are output according to the start day of the billing period which was defined by the supplier or reseller.

A `Period` element has the following attributes:

- **`startDate`** - (data type `long`) Start time of the period. The start time is specified in milliseconds, the starting point for the calculation is `1970-01-01, 00:00`.
- **`startDateIsoFormat`** - (optional, data type `dateTime`) Same as `startDate`, but specified in ISO 8601 format (`YYYY-MM-DDThh:mm:ss.fffZ`) .
- **`endDate`** - (data type `long`) End time of the period. The end time is specified in milliseconds, the starting point for the calculation is `1970-01-01, 00:00`.
- **`endDateIsoFormat`** - (optional, data type `dateTime`) Same as `endDate`, but specified in ISO 8601 format (`YYYY-MM-DDThh:mm:ss.fffZ`).

Example:

```
<Period startDateIsoFormat="2012-08-31T22:00:00.000Z"
  startDate="1346450400000"
  endDateIsoFormat="2012-09-30T22:00:00.000Z"
  endDate="1349042400000"/>
```

OrganizationData

Provides details of an organization.

An `OrganizationData` element has the following attributes:

- **`id`** - (required, data type `string`) ID of the organization.
- **`key`** - (required, data type `positiveInteger`) Internal numeric key of the organization.

An `OrganizationData` element contains the following subelements:

- **`Email`** - (data type `string`) Email address of the organization.
- **`Name`** - (data type `string`) Name of the organization.
- **`Address`** - (data type `string`) Address of the organization.
- **`CountryIsoCode`** - (data type `string`) ISO code of the country where the organization is located.

Example:

```
<OrganizationData id="8e8f596c" key="37003">
  <Email>info@company.com</Email>
  <Name>Company</Name>
  <Address>Postal Address</Address>
  <CountryIsoCode>DE</CountryIsoCode>
</OrganizationData>
```

B.2 Broker Revenue Share Data

The following sections describe the XML elements and attributes that make up the revenue share data for brokers.

BrokerRevenueShareResult

Top-level container element for broker revenue share data. For each broker organization in consideration, a `BrokerRevenueShareResult` element is added to the billing data file.

A `BrokerRevenueShareResult` element has the following attributes:

- **`organizationId`** - (required, data type `string`) ID of the broker organization.
- **`organizationKey`** - (required, data type `positiveInteger`) Internal numeric key of the broker organization.

A `BrokerRevenueShareResult` element contains the following subelements:

- An `OrganizationData` element specifying the details of the broker organization (see *OrganizationData* on page 66).
- A `Period` element specifying the billing period (see *Period* on page 65).
- A `Currency` element for each currency for which broker revenue share data is available (see *Currency* on page 67).

Example:

```
<BrokerRevenueShareResult organizationId="cd9ffaac"
 organizationKey="19000" >
  <OrganizationData> ... </OrganizationData>
  <Period> ... </Period>
  <Currency> ... </Currency>
</BrokerRevenueShareResult>
```

Currency

Contains the broker revenue share data for a specific currency.

A `Currency` element has the following attribute:

`id` - (required, data type `string`) ISO code of the currency.

A `Currency` element contains the following subelements:

- A `Supplier` element for each supplier organization that provides a service for which the current broker organization receives a revenue share (see *Supplier* on page 67).
- A `BrokerRevenue` element specifying the overall revenue for the currency in its `totalAmount` attribute (optional, data type positive `decimal`, scale 2), and the overall broker revenue share for the currency in its `amount` attribute (required, data type positive `decimal`, scale 2).

Example:

```
<Currency id="EUR">
  <Supplier>...</Supplier>
  <BrokerRevenue totalAmount="1000.50" amount="100.05" />
</Currency>
```

Supplier

Contains the broker revenue share data for the services provided by a specific supplier.

A `Supplier` element contains the following subelements:

- An `OrganizationData` element specifying the details of the supplier organization (see *OrganizationData* on page 66).
- A `Service` element for each service provided by the supplier for which the current broker organization receives a revenue share (see *Service* on page 67).
- A `BrokerRevenuePerSupplier` element specifying the overall revenue for the supplier in its `totalAmount` attribute (optional, data type positive `decimal`, scale 2), and the overall broker revenue share in its `amount` attribute (required, data type positive `decimal`, scale 2).

Example:

```
<Supplier>
  <OrganizationData> ... </OrganizationData>
  <Service> ... </Service>
  <BrokerRevenuePerSupplier totalAmount="200.50" amount="20.05" />
</Supplier>
```

Service

Specifies the broker revenue share data for a specific service.

A `Service` element has the following attributes:

- **`id`** - (required, data type `string`) Name of the service.

- **key** - (required, data type positiveInteger) Internal numeric key of the service offered by the broker. Technically, this is a copy of the original service defined by the supplier.
- **templateKey** - (required, data type positiveInteger) Internal numeric key of the original service defined by the supplier.

A Service element contains a ServiceRevenue element which specifies the total revenue for the service and the broker revenue share in its attributes:

- **totalAmount** - (required, data type positive decimal, scale 2) Total revenue for the service in the billing period.
- **brokerRevenueSharePercentage** - (required, data type positive decimal, scale 2) Percentage of the revenue the broker is entitled to.
- **brokerRevenue** - (required, data type positive decimal, scale 2) The broker's revenue share for the service in the billing period.

A ServiceRevenue element contains a ServiceCustomerRevenue subelement for each customer who used the service. It specifies the total revenue for the service generated by the customer and the broker revenue share in its attributes:

- **customerName** - (optional, data type string) Name of the customer organization.
- **customerId** - (optional, data type string) ID of the customer organization.
- **totalAmount** - (optional, data type positive decimal, scale 2) Total revenue for the service generated by the customer in the billing period.
- **brokerRevenueSharePercentage** - (optional, data type positive decimal, scale 2) Percentage of the revenue the broker is entitled to.
- **brokerRevenue** - (optional, data type positive decimal, scale 2) The broker's revenue share for the service generated by the customer in the billing period.

Example:

```
<Service id="Mega Office" key="17005" templateKey="10501">
  <ServiceRevenue totalAmount="200.00"
   brokerRevenueSharePercentage="10.00" brokerRevenue="20.00" />
    <ServiceCustomerRevenue customerName="MyCompany"
     customerId="862cfb94" totalAmount="50.00"
     brokerRevenueSharePercentage="10.00" brokerRevenue="5.00" />
</Service>
```

B.3 Reseller Revenue Share Data

The following sections describe the XML elements and attributes that make up the revenue share data for resellers.

ResellerRevenueShareResult

Top-level container element for reseller revenue share data. For each reseller organization in consideration, a ResellerRevenueShareResult element is added to the billing data file.

A ResellerRevenueShareResult element has the following attributes:

- **organizationId** - (required, data type string) ID of the reseller organization.
- **organizationKey** - (required, data type positiveInteger) Internal numeric key of the reseller organization.

A `ResellerRevenueShareResult` element contains the following subelements:

- An `OrganizationData` element specifying the details of the reseller organization (see *OrganizationData* on page 66).
- A `Period` element specifying the billing period (see *Period* on page 65).
- A `Currency` element for each currency for which reseller revenue share data is available (see *Currency* on page 69).

Example:

```
<ResellerRevenueShareResult organizationId="cd9ffaac"
 organizationKey="19000">
  <OrganizationData> ... </OrganizationData>
  <Period> ... </Period>
  <Currency> ... </Currency>
</ResellerRevenueShareResult>
```

Currency

Contains the reseller revenue share data for a specific currency.

A `Currency` element has the following attribute:

`id` - (required, data type `string`) ISO code of the currency.

A `Currency` element contains the following subelements:

- A `Supplier` element for each supplier organization that provides a service for which the current reseller organization receives a revenue share (see *Supplier* on page 69).
- A `ResellerRevenue` element with the following attributes:

  `totalAmount` - (optional, data type positive `decimal`, scale 2) Overall revenue for the currency.

  `amount` - (required, data type positive `decimal`, scale 2) Overall reseller revenue share.

  `purchasePrice` - (optional, data type positive `decimal`, scale 2) Difference between the `totalAmount` and the `amount` attribute.

Example:

```
<Currency id="EUR">
  <Supplier>...</Supplier>
  <ResellerRevenue totalAmount="1000.50" amount="200.10"
   purchasePrice="800.40"/>
</Currency>
```

Supplier

Contains the reseller revenue share data for the services provided by a specific supplier.

A `Supplier` element contains the following subelements:

- An `OrganizationData` element specifying the details of the supplier organization (see *OrganizationData* on page 66).
- A `Service` element for each service provided by the supplier for which the current reseller organization receives a revenue share (see *Service* on page 70).
- A `ResellerRevenuePerSupplier` element with the following attributes:

  `totalAmount` - (optional, data type positive `decimal`, scale 2) Overall revenue for the supplier.

  `amount` - (required, data type positive `decimal`, scale 2) Overall reseller revenue share for the supplier.

`purchasePrice` - (optional, data type positive `decimal`, scale 2) Difference between the `totalAmount` and the `amount` attribute.

Example:

```
<Supplier>
  <OrganizationData> ... </OrganizationData>
  <Service> ... </Service>
  <ResellerRevenuePerSupplier totalAmount="200.50" amount="40.10"
   purchasePrice="160.40"/>
</Supplier>
```

Service

Specifies the reseller revenue share data for a specific service.

A `Service` element has the following attributes:

- **`id`** - (required, data type `string`) Name of the service.
- **`key`** - (required, data type `positiveInteger`) Internal numeric key of the service offered by the reseller. Technically, this is a copy of the original service defined by the supplier.
- **`templateKey`** - (required, data type `positiveInteger`) Internal numeric key of the original service defined by the supplier.

A `Service` element contains the following subelements:

- A `Subscription` element for each subscription to the service offered by the reseller (see *Subscription* on page 70).
- A `ServiceRevenue` element specifying the overall reseller revenue share for the service (see *ServiceRevenue* on page 71).

Example:

```
<Service id="Mega Office" key="17005" templateKey="10501">
  <Subscription> ... </Subscription>
  <ServiceRevenue> ... </ServiceRevenue>
</Service>
```

Subscription

Specifies the revenue for a specific subscription to a service.

A `Subscription` element has the following attributes:

- **`id`** - (required, data type `string`) Name of the subscription.
- **`key`** - (required, data type `positiveInteger`) Internal numeric key of the subscription.
- **`billingKey`** - (required, data type `positiveInteger`) Unique identifier allowing, for example, accounting systems to relate billing data to an invoice. The billing data key is printed on the invoice.
- **`revenue`** - (required, data type positive `decimal`, scale 2) The total revenue for the subscription in the billing period. Note that there is a `Service` element for each phase of the subscription if the subscription is upgraded or downgraded.

A `Subscription` element contains a `Period` subelement specifying the applicable billing period (see *Period* on page 65).

Example:

```
<Subscription id="Mega Office Basic" key="17005" billingKey="19032"
  revenue="600.00">
  <Period>... </Period>
</Subscription>
```

ServiceRevenue

Specifies the total revenue for a service and the reseller revenue share.

A `ServiceRevenue` element has the following attributes:

- **`totalAmount`** - (required, data type positive `decimal`, scale 2) Total revenue for all subscriptions to the service in the billing period.
- **`resellerRevenueSharePercentage`** - (required, data type positive `decimal`, scale 2) Percentage of the revenue the reseller is entitled to.
- **`resellerRevenue`** - (required, data type positive `decimal`, scale 2) The reseller's revenue share for the service in the billing period.

A `ServiceRevenue` element contains a `ServiceCustomerRevenue` subelement for each customer who used the service. It specifies the total revenue for the service generated by the customer and the reseller revenue share in its attributes:

- **`customerName`** - (optional, data type `string`) Name of the customer organization.
- **`customerId`** - (optional, data type `string`) ID of the customer organization.
- **`totalAmount`** - (optional, data type positive `decimal`, scale 2) Total revenue for the service generated by the customer in the billing period.
- **`resellerRevenueSharePercentage`** - (optional, data type positive `decimal`, scale 2) Percentage of the revenue the reseller is entitled to.
- **`resellerRevenue`** - (optional, data type positive `decimal`, scale 2) The reseller's revenue share for the service generated by the customer in the billing period.
- **`purchasePrice`** - (optional, data type positive `decimal`, scale 2) Difference between the `totalAmount` and the `resellerRevenue` attribute.

Example:

```
<ServiceRevenue totalAmount="200.00"
  resellerRevenueSharePercentage="10.00" resellerRevenue="20.00">
    <ServiceCustomerRevenue customerName="MyCompany"
     customerId="862cfb94" totalAmount="50.00"
     resellerRevenueSharePercentage="10.00" resellerRevenue="5.00"
     purchasePrice="45"/>
</ServiceRevenue>
```

B.4 Marketplace Owner Revenue Share Data

The following sections describe the XML elements and attributes that make up the revenue share data for marketplace owners.

MarketplaceOwnerRevenueShareResult

Top-level container element for marketplace owner revenue share data. For each marketplace owner organization in consideration, a `MarketplaceOwnerRevenueShareResult` element is added to the billing data file.

A `MarketplaceOwnerRevenueShareResult` element has the following attributes:

- **`organizationId`** - (required, data type `string`) ID of the marketplace owner organization.
- **`organizationKey`** - (required, data type `positiveInteger`) Internal numeric key of the marketplace owner organization.

A `MarketplaceOwnerRevenueShareResult` element contains the following subelements:

- An `OrganizationData` element specifying the details of the marketplace owner organization (see *OrganizationData* on page 66).
- A `Period` element specifying the billing period (see *Period* on page 65).
- A `Currency` element for each currency for which marketplace owner revenue share data is available (see *Currency* on page 72).

Example:

```
<MarketplaceOwnerRevenueShareResult organizationId="cd9ffaac"
 organizationKey="19000">
  <OrganizationData> ... </OrganizationData>
  <Period> ... </Period>
  <Currency> ... </Currency>
</MarketplaceOwnerRevenueShareResult>
```

Currency

Contains the marketplace owner revenue share data for a specific currency.

A `Currency` element has the following attribute:

`id` - (required, data type `string`) ISO code of the currency.

A `Currency` element contains the following subelements:

- A `Marketplace` element for each marketplace for which revenue share data for the current marketplace owner organization is available (see *Marketplace* on page 72).
- A `RevenuesOverAllMarketplaces` element summarizing the revenue shares across the marketplaces (see *RevenuesOverAllMarketplaces* on page 76).

Example:

```
<Currency id="EUR">
  <Marketplace>...</Marketplace>
  <RevenuesOverAllMarketplaces> ... </RevenuesOverAllMarketplaces>
</Currency>
```

Marketplace

Contains the revenue share data for a specific marketplace.

A `Marketplace` element has the following attributes:

- **`id`** - (required, data type `string`) ID of the marketplace.
- **`key`** - (required, data type `positiveInteger`) Internal numeric key of the marketplace.

A `Marketplace` element contains the following subelements:

- A `Service` element for each service published on the marketplace for which revenue share data is available (see *Service* on page 73).
- A `RevenuesPerMarketplace` element summarizing the revenue shares for all organizations involved (see *RevenuesPerMarketplace* on page 75).

Example:

```
<Marketplace id="e1828fba" key="17021">
  <Service>...</Service>
  <RevenuesPerMarketplace> ... </RevenuesPerMarketplace>
</Marketplace>
```

Service

Specifies the revenue share data for a specific service published on the current marketplace.

A `Service` element has the following attributes:

- **`id`** - (required, data type `string`) Name of the service.
- **`key`** - (required, data type `positiveInteger`) Internal numeric key of the published service. If the service is offered by a broker or reseller, this is the key of an internal technical copy of the original service defined by the supplier. If the service is offered directly by its supplier, it is the key of the original service.
- **`model`** - (required, data type `string`) String specifying by which type of organization the service is offered. Possible values are:
  - `DIRECT` : The service is offered by its supplier.
  - `BROKER` : The service is offered by a broker.
  - `RESELLER` : The service is offered by a reseller.
- **`templateKey`** - (optional, data type `positiveInteger`) Internal numeric key of the original service defined by the supplier, if the service is published by a broker or reseller.

A `Service` element contains the following subelements:

- A `Supplier` element specifying the supplier organization who defined the service in an `OrganizationData` subelement (see *OrganizationData* on page 66).
- If the service is offered by a broker: A `Broker` element specifying the broker organization in an `OrganizationData` subelement (see *OrganizationData* on page 66).
- If the service is offered by a reseller: A `Reseller` element specifying the reseller organization in an `OrganizationData` subelement (see *OrganizationData* on page 66).
- A `RevenueShareDetails` element specifying the revenue shares for the service (see *RevenueShareDetails* on page 74).

Examples:

The service is offered directly by its supplier:

```
<Service id="Mega Office" key="17005" model="DIRECT">
  <Supplier> <OrganizationData> ... </OrganizationData> </Supplier>
  <RevenueShareDetails> ... </RevenueShareDetails>
</Service>
```

The service is offered by a broker:

```
<Service id="Mega Office" key="17005" model="BROKER"
 templateKey="10501">
  <Supplier> <OrganizationData> ... </OrganizationData> </Supplier>
  <Broker> <OrganizationData> ... </OrganizationData> </Broker>
  <RevenueShareDetails> ... </RevenueShareDetails>
</Service>
```

The service is offered by a reseller:

```
<Service id="Mega Office" key="17005" model="RESELLER"
 templateKey="10501">
  <Supplier> <OrganizationData> ... </OrganizationData> </Supplier>
  <Reseller> <OrganizationData> ... </OrganizationData> </Reseller>
  <RevenueShareDetails> ... </RevenueShareDetails>
</Service>
```

RevenueShareDetails

Specifies the revenue for a specific service and the revenue shares for all organizations involved in selling the service.

A `RevenueShareDetails` element has the following attributes:

- **`serviceRevenue`** - (required, data type `decimal`, scale 2) Total revenue for the service in the billing period.
- **`marketplaceRevenueSharePercentage`** - (required, data type `decimal`, scale 2) Percentage of the revenue the marketplace owner is entitled to.
- **`operatorRevenueSharePercentage`** - (required, data type `decimal`, scale 2) Percentage of the revenue the platform operator is entitled to.
- **`brokerRevenueSharePercentage`** - (optional, data type `decimal`, scale 2) If the service is offered by a broker: Percentage of the revenue the broker is entitled to.
- **`resellerRevenueSharePercentage`** - (optional, data type `decimal`, scale 2) If the service is offered by a reseller: Percentage of the revenue the reseller is entitled to.
- **`marketplaceRevenue`** - (required, data type `decimal`, scale 2) The marketplace owner's revenue share for the service in the billing period.
- **`operatorRevenue`** - (required, data type `decimal`, scale 2) The platform operator's revenue share for the service in the billing period.
- **`brokerRevenue`** - (optional, data type `decimal`, scale 2) If the service is offered by a broker: The broker's revenue share for the service in the billing period.
- **`resellerRevenue`** - (optional, data type `decimal`, scale 2) If the service is offered by a reseller: The reseller's revenue share for the service in the billing period.
- **`amountForSupplier`** - (required, data type `decimal`, scale 2) The supplier's revenue share for the service in the billing period. This is the remaining value of the total service revenue after subtracting the revenue shares for the operator, marketplace owner, broker, and/or reseller.

Examples:

The service is offered directly by its supplier:

```
<RevenueShareDetails serviceRevenue="500.00"
  marketplaceRevenueSharePercentage="15.00" marketplaceRevenue="75.00"
  operatorRevenueSharePercentage="10.00" operatorRevenue="50.00"
  amountForSupplier="375.00">
</RevenueShareDetails>
```

The service is offered by a broker:

```
<RevenueShareDetails serviceRevenue="4000.00"
  marketplaceRevenueSharePercentage="21.00" marketplaceRevenue="840.00"
  brokerRevenueSharePercentage="9.00" brokerRevenue="360.00"
  operatorRevenueSharePercentage="5.00" operatorRevenue="200.00"
  amountForSupplier="2600.00">
```

```
</RevenueShareDetails>
```

The service is offered by a reseller:

```
<RevenueShareDetails serviceRevenue="3000.00"
  marketplaceRevenueSharePercentage="16.00" marketplaceRevenue="480.00"
  resellerRevenueSharePercentage="20.00" resellerRevenue="600.00"
  operatorRevenueSharePercentage="5.00" operatorRevenue="150.00"
  amountForSupplier="1770.00">
</RevenueShareDetails>
```

RevenuesPerMarketplace

Provides an overview of the revenue shares for the different organizations involved in selling services on a specific marketplace.

A `RevenuesPerMarketplace` element contains the following subelements:

- A `Brokers` element listing the relevant broker organizations with their revenue shares in `Organization` subelements.
- A `Resellers` element listing the relevant reseller organizations with their revenue shares in `Organization` subelements.
- A `Suppliers` element listing the relevant supplier organizations with their revenue shares in `Organization` subelements.
- A `MarketplaceOwner` element specifying the revenue share for the marketplace owner in its `amount` attribute (required, data type `decimal`, scale 2).

Each `Brokers`, `Resellers`, or `Suppliers` element included in a `RevenuesPerMarketplace` element has the following attributes:

- **`amount`** - (optional, data type `decimal`, scale 2) Overall revenue share of the listed broker, reseller, or supplier organizations.
- **`marketplaceRevenue`** - (optional, data type `decimal`, scale 2) The marketplace owner's share of the revenue of the listed broker, reseller, or supplier organizations.
- **`totalAmount`** - (optional, data type `decimal`, scale 2) Overall revenue for all services offered by the listed broker, reseller, or supplier organizations on the marketplace.

An `Organization` element included in a `Brokers`, `Resellers`, or `Suppliers` element has the following attributes:

- **`identifier`** - (required, data type `string`) ID of the organization.
- **`amount`** - (required, data type `decimal`, scale 2) Revenue share of the organization.
- **`name`** - (optional, data type `string`) Name of the organization.
- **`marketplaceRevenue`** - (optional, data type `decimal`, scale 2) The marketplace owner's share of the organization's revenue.
- **`totalAmount`** - (optional, data type `decimal`, scale 2) Overall revenue for all services offered by the organization on the marketplace.

Example:

```
<RevenuesPerMarketplace>
  <Brokers amount="250.00" totalAmount="1000.00"
   marketplaceRevenue="200.00">
    <Organization identifier="da3cd3a3" amount="100.00" name="broker"
     marketplaceRevenue="80.00" totalAmount="400.00" />
    <Organization identifier="ea4cd3a3" amount="150.00" name="broker2"
```

```
    marketplaceRevenue="120.00" totalAmount="600.00" />
  </Brokers>
  <Resellers amount="600.00" totalAmount="2000.00"
   marketplaceRevenue="400.00">
    <Organization identifier="bc4cd3a3" amount="240.00" name="reseller"
     marketplaceRevenue="160.00" totalAmount="800.00" />
    <Organization identifier="fg5cd3a3" amount="360.00" name="reseller2"
     marketplaceRevenue="240.00" totalAmount="1200.00" />
  </Resellers>
  <Suppliers />
  <MarketplaceOwner amount="600.00" />
</RevenuesPerMarketplace>
```

RevenuesOverAllMarketplaces

Provides an overview of the revenue shares for the different organizations involved in selling services on any of the marketplaces that belong to a specific marketplace owner.

A `RevenuesOverAllMarketplaces` element contains the following subelements:

- A `Brokers` element listing the relevant broker organizations with their revenue shares in `Organization` subelements.
- A `Resellers` element listing the relevant reseller organizations with their revenue shares in `Organization` subelements.
- A `Suppliers` element listing the relevant supplier organizations with their revenue shares in `Organization` subelements.
- A `MarketplaceOwner` element specifying the revenue share for the marketplace owner in its `amount` attribute (required, data type `decimal`, scale 2).

Each `Brokers`, `Resellers`, or `Suppliers` element included in a `RevenuesOverAllMarketplaces` element has the following attributes:

- **`amount`** - (optional, data type `decimal`, scale 2) Overall revenue share of the listed broker, reseller, or supplier organizations.
- **`marketplaceRevenue`** - (optional, data type `decimal`, scale 2) The marketplace owner's share of the revenue of the listed broker, reseller, or supplier organizations.
- **`totalAmount`** - (optional, data type `decimal`, scale 2) Overall revenue for all services offered by the listed broker, reseller, or supplier organizations on the marketplaces.

An `Organization` element included in a `Brokers`, `Resellers`, or `Suppliers` element has the following attributes:

- **`identifier`** - (required, data type `string`) ID of the organization.
- **`amount`** - (required, data type `decimal`, scale 2) Revenue share of the organization.
- **`name`** - (optional, data type `string`) Name of the organization.
- **`marketplaceRevenue`** - (optional, data type `decimal`, scale 2) The marketplace owner's share of the organization's revenue.
- **`totalAmount`** - (optional, data type `decimal`, scale 2) Overall revenue for all services offered by the organizations on the marketplaces.

Example:

```
<RevenuesOverAllMarketplaces>
  <Brokers amount="250.00" totalAmount="1000.00"
   marketplaceRevenue="200.00">
    <Organization identifier="da3cd3a3" amount="100.00" name="broker"
```

```
    marketplaceRevenue="80.00" totalAmount="400.00" />
   <Organization identifier="ea4cd3a3" amount="150.00" name="broker2"
    marketplaceRevenue="120.00" totalAmount="600.00" />
  </Brokers>
  <Resellers amount="600.00" totalAmount="2000.00"
   marketplaceRevenue="400.00">
   <Organization identifier="bc4cd3a3" amount="240.00" name="reseller"
    marketplaceRevenue="160.00" totalAmount="800.00" />
   <Organization identifier="fg5cd3a3" amount="360.00" name="reseller2"
    marketplaceRevenue="240.00" totalAmount="1200.00" />
  </Resellers>
  <Suppliers />
  <MarketplaceOwner amount="600.00" />
</RevenuesOverAllMarketplaces>
```

B.5 Supplier Revenue Share Data

The following sections describe the XML elements and attributes that make up the revenue share data for suppliers.

SupplierRevenueShareResult

Top-level container element for all revenue share data a supplier has to pay to all participating parties (platform operator, marketplace owners, brokers, resellers) based on his contractual agreements. For each supplier organization in consideration, a `SupplierRevenueShareResult` element is added to the billing data file.

A `SupplierRevenueShareResult` element has the following attributes:

- **`organizationId`** - (required, data type `string`) ID of the supplier organization.
- **`organizationKey`** - (required, data type `positiveInteger`) Internal numeric key of the supplier organization.

A `SupplierRevenueShareResult` element contains the following subelements:

- An `OrganizationData` element specifying the details of the supplier organization (see *OrganizationData* on page 66).
- A `Period` element specifying the billing period (see *Period* on page 65).
- A `Currency` element for each currency for which supplier revenue share data is available (see *Currency* on page 77).

Example:

```
<SupplierRevenueShareResult organizationId="cd9ffaac"
 organizationKey="19000">
  <OrganizationData> ... </OrganizationData>
  <Period> ... </Period>
  <Currency> ... </Currency>
</SupplierRevenueShareResult>
```

Currency

Contains the supplier revenue share data for a specific currency.

A `Currency` element has the following attribute:

`id` - (required, data type `string`) ISO code of the currency.

A `Currency` element contains the following subelements:

- A `Marketplace` element for each marketplace for which revenue share data for the supplier organization is available (see *Marketplace* on page 78).
- A `SupplierRevenue` element specifying the detailed revenue share data for the current supplier organization (see *SupplierRevenue* on page 78).

Example:

```
<Currency id="EUR">
  <Marketplace>...</Marketplace>
  <SupplierRevenue>...</SupplierRevenue>
</Currency>
```

Marketplace

Contains the revenue share data for a specific marketplace.

A `Marketplace` element has the following attributes:

- **`id`** - (required, data type `string`) ID of the marketplace.
- **`key`** - (required, data type `positiveInteger`) Internal numeric key of the marketplace.

A `Marketplace` element contains the following subelements:

- A `MarketplaceOwner` element specifying the marketplace owner in an `OrganizationData` subelement (see *OrganizationData* on page 66).
- A `Service` element for each service published on the marketplace for which revenue share data for the supplier is available (see *Service* on page 80).
- A `RevenuePerMarketplace` element summarizing the revenue shares for all organizations involved (see *RevenuePerMarketplace* on page 84).

Example:

```
<Marketplace id="e1828fba" key="17021">
  <MarketplaceOwner>
    <OrganizationData> ... </OrganizationData>
  </MarketplaceOwner>
  <Service>...</Service>
  <RevenuePerMarketplace> ... </RevenuePerMarketplace>
</Marketplace>
```

SupplierRevenue

Contains the detailed revenue data for a supplier organization.

A `SupplierRevenue` element has the following attributes:

- **`amount`** - (required, data type `decimal`, scale 2) Overall revenue for the supplier.

A `SupplierRevenue` element may contain the following subelements, depending on which organizations offer the supplier's services:

- A `DirectRevenue` element specifying the revenue for the services offered directly by the supplier.
- A `BrokerRevenue` element specifying the revenue for the supplier's services offered by brokers.
- A `ResellerRevenue` element specifying the revenue for the supplier's services offered by resellers.

Example:

```
<SupplierRevenue amount="1500.00">
  <DirectRevenue> ... </DirectRevenue>
  <BrokerRevenue> ... </BrokerRevenue>
  <ResellerRevenue> ... </ResellerRevenue>
</SupplierRevenue>
```

DirectRevenue

Contains the revenue data for services offered directly by their supplier.

A `DirectRevenue` element has the following attributes:

- **`serviceRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the services offered by the supplier.
- **`marketplaceRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the owners of the marketplaces where the services are offered by the supplier.
- **`operatorRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the platform operator who provides the infrastructure for the marketplaces and services.

Example:

```
<SupplierRevenue amount="1500.00">
  <DirectRevenue serviceRevenue="600.00" marketplaceRevenue="75.00"
   operatorRevenue="25.00"/>
  <BrokerRevenue> ... </BrokerRevenue>
  <ResellerRevenue> ...</ResellerRevenue>
</SupplierRevenue>
```

BrokerRevenue

Contains the revenue data for a supplier's services offered by brokers.

A `BrokerRevenue` element has the following attributes:

- **`serviceRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the services offered by the broker.
- **`marketplaceRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the owners of the marketplaces where the services are offered by brokers.
- **`brokerRevenue`** - (required, data type `decimal`, scale 2) Overall revenue share for the broker offering the services.
- **`operatorRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the platform operator who provides the infrastructure for the marketplaces and services.

Example:

```
<SupplierRevenue amount="1500.00">
  <DirectRevenue> ... </DirectRevenue>
  <BrokerRevenue serviceRevenue="700.00" marketplaceRevenue="75.00"
   brokerRevenue="25.00" operatorRevenue="100.00"/>
  <ResellerRevenue> ...</ResellerRevenue>
</SupplierRevenue>
```

ResellerRevenue

Contains the revenue data for the supplier's services offered by resellers.

A `ResellerRevenue` element has the following attributes:

- **`serviceRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the services offered by resellers.
- **`marketplaceRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the owners of the marketplaces where the services are offered by the resellers.
- **`resellerRevenue`** - (required, data type `decimal`, scale 2) Overall revenue share for the resellers offering the services.
- **`operatorRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the platform operator who provides the infrastructure for the marketplaces and services.
- **`overallRevenue`** - (required, data type `decimal`, scale 2) Overall revenue for the services offered by the resellers minus the marketplace owner revenue (`marketplaceRevenue`), the reseller revenue (`resellerRevenue`), and the operator revenue (`operatorRevenue`).

Example:

```
<SupplierRevenue amount="1500.00">
  <DirectRevenue> ... </DirectRevenue>
  <BrokerRevenue> ... </BrokerRevenue>
  <ResellerRevenue serviceRevenue="650.00"
  marketplaceRevenue="75.00"
  resellerRevenue="25.00"
  operatorRevenue="50.00"
  overallRevenue="500.00" />
</SupplierRevenue>
```

Service

Specifies the revenue share data for a specific service.

A `Service` element has the following attributes:

- **`id`** - (required, data type `string`) Name of the service.
- **`key`** - (required, data type `positiveInteger`) Internal numeric key of the published service. If the service is offered by a broker or reseller, this is the key of an internal technical copy of the original service defined by the supplier. If the service is offered directly by its supplier, it is the key of the original service.
- **`model`** - (required, data type `string`) String specifying by which type of organization the service is offered. Possible values are:
  - `DIRECT` : The service is offered by its supplier.
  - `BROKER` : The service is offered by a broker.
  - `RESELLER` : The service is offered by a reseller.
- **`templateKey`** - (optional, data type `positiveInteger`) Internal numeric key of the original service defined by the supplier, if the service is published by a broker or reseller.

A `Service` element contains the following subelements:

- If the service is offered directly by the supplier: A `Subscription` element for each subscription to the service for which revenue share data is available (see *Subscription* on page 81).
- If the service is offered by a broker:
  - A `Subscription` element for each subscription to the service for which revenue share data is available (see *Subscription* on page 81).

    - A `Broker` element specifying the broker organization in an `OrganizationData` subelement (see *OrganizationData* on page 66).
- If the service is offered by a reseller:
    - A `SubscriptionsRevenue` element summarizing the total revenue for all subscriptions to the service in its `amount` attribute (required, data type positive `decimal`, scale 2).
    - A `Reseller` element specifying the reseller organization in an `OrganizationData` subelement (see *OrganizationData* on page 66).
- A `RevenueShareDetails` element specifying the revenue shares for the service (see *RevenueShareDetails* on page 82).

Examples:

The service is offered directly by its supplier:

```
<Service id="Mega Office" key="17005" model="DIRECT">
  <Subscription> ... </Subscription>
  <RevenueShareDetails> ... </RevenueShareDetails>
</Service>
```

The service is offered by a broker:

```
<Service id="Mega Office" key="17005" model="BROKER"
 templateKey="10501">
  <Subscription> ... </Subscription>
  <Broker> <OrganizationData> ... </OrganizationData> </Broker>
  <RevenueShareDetails> ... </RevenueShareDetails>
</Service>
```

The service is offered by a reseller:

```
<Service id="Mega Office" key="17005" model="RESELLER"
 templateKey="10501">
  <SubscriptionsRevenue amount="6000.00" />
  <Reseller> <OrganizationData> ... </OrganizationData> </Reseller>
  <RevenueShareDetails> ... </RevenueShareDetails>
</Service>
```

Subscription

Specifies the revenue for a specific subscription to a service.

A `Subscription` element has the following attributes:

- **`id`** - (required, data type `string`) Name of the subscription.
- **`key`** - (required, data type `positiveInteger`) Internal numeric key of the subscription.
- **`billingKey`** - (required, data type `positiveInteger`) Unique identifier allowing, for example, accounting systems to relate billing data to an invoice. The billing data key is printed on the invoice.
- **`revenue`** - (required, data type positive `decimal`, scale 2) The total revenue for the subscription in the billing period. Note that there is a `Service` element for each phase of the subscription if the subscription is upgraded or downgraded.

A `Subscription` element contains a `Period` subelement specifying the applicable billing period (see *Period* on page 65).

Example:

```
<Subscription id="Mega Office Basic" key="17005" billingKey="19032"
  revenue="600.00">
  <Period>... </Period>
</Subscription>
```

RevenueShareDetails

Specifies the revenue for a specific service and the revenue shares for all organizations involved in selling the service.

A `RevenueShareDetails` element has the following attributes:

- **`serviceRevenue`** - (required, data type `decimal`, scale 2) Total revenue for the service in the billing period.
- **`marketplaceRevenueSharePercentage`** - (required, data type `decimal`, scale 2) Percentage of the revenue the marketplace owner is entitled to.
- **`brokerRevenueSharePercentage`** - (optional, data type `decimal`, scale 2) If the service is offered by a broker: Percentage of the revenue the broker is entitled to.
- **`resellerRevenueSharePercentage`** - (optional, data type `decimal`, scale 2) If the service is offered by a reseller: Percentage of the revenue the reseller is entitled to.
- **`marketplaceRevenue`** - (required, data type `decimal`, scale 2) The marketplace owner's revenue share for the service in the billing period.
- **`brokerRevenue`** - (optional, data type `decimal`, scale 2) If the service is offered by a broker: The broker's revenue share for the service in the billing period.
- **`resellerRevenue`** - (optional, data type `decimal`, scale 2) If the service is offered by a reseller: The reseller's revenue share for the service in the billing period.
- **`operatorRevenueSharePercentage`** - (required, data type `decimal`, scale 2) Percentage of the revenue the operator is entitled to.
- **`operatorRevenue`** - (required, data type `decimal`, scale 2) The operator's revenue share for the service in the billing period.
- **`amountForSupplier`** - (required, data type `decimal`, scale 2) The supplier's revenue share for the service in the billing period. This is the remaining value of the total service revenue after subtracting the revenue shares for the operator, marketplace owner, broker, and/or reseller.

A `RevenueShareDetails` element contains a `CustomerRevenueShareDetails` subelement for each customer who used the service (see *CustomerRevenueShareDetails* on page 83).

Examples:

The service is offered directly by its supplier:

```
<RevenueShareDetails serviceRevenue="500.00"
  marketplaceRevenueSharePercentage="15.00" marketplaceRevenue="75.00"
  operatorRevenueSharePercentage="10.00" operatorRevenue="50.00"
  amountForSupplier="375.00">
  <CustomerRevenueShareDetails> ... </CustomerRevenueShareDetails>
</RevenueShareDetails>
```

The service is offered by a broker:

```
<RevenueShareDetails serviceRevenue="4000.00"
  marketplaceRevenueSharePercentage="21.00" marketplaceRevenue="840.00"
  brokerRevenueSharePercentage="9.00" brokerRevenue="360.00"
```

```
  operatorRevenueSharePercentage="10.00" operatorRevenue="400.00"
  amountForSupplier="2400.00">
  <CustomerRevenueShareDetails> ... </CustomerRevenueShareDetails>
</RevenueShareDetails>
```

The service is offered by a reseller:

```
<RevenueShareDetails serviceRevenue="3000.00"
  marketplaceRevenueSharePercentage="16.00" marketplaceRevenue="480.00"
  resellerRevenueSharePercentage="20.00" resellerRevenue="600.00"
  operatorRevenueSharePercentage="10.00" operatorRevenue="300.00"
  amountForSupplier="1620.00">
  <CustomerRevenueShareDetails> ... </CustomerRevenueShareDetails>
</RevenueShareDetails>
```

CustomerRevenueShareDetails

Specifies the revenue for a specific service generated by a given customer and the revenue shares for all organizations involved in selling the service.

A `CustomerRevenueShareDetails` element has the following attributes:

- **`customerName`** - (required, data type `string`) Name of the customer organization.
- **`customerId`** - (required, data type `string`) ID of the customer organization.
- **`serviceRevenue`** - (required, data type `decimal`, scale 2) Total revenue for the service in the billing period.
- **`marketplaceRevenue`** - (required, data type `decimal`, scale 2) The marketplace owner's revenue share for the service in the billing period.
- **`resellerRevenue`** - (optional, data type `decimal`, scale 2) If the service is offered by a reseller: The reseller's revenue share for the service in the billing period.
- **`brokerRevenue`** - (optional, data type `decimal`, scale 2) If the service is offered by a broker: The broker's revenue share for the service in the billing period.
- **`operatorRevenue`** - (required, data type `decimal`, scale 2) The operator's revenue share for the service in the billing period.
- **`amountForSupplier`** - (required, data type `decimal`, scale 2) The supplier's revenue share for the service generated by the customer in the billing period. This is the remaining value of the total service revenue generated by the customer after subtracting the revenue shares for the operator, marketplace owner, broker, and/or reseller.

Examples:

The service is offered directly by its supplier:

```
<CustomerRevenueShareDetails customerName="MyCompany"
  customerId="8dac00a0" serviceRevenue="316.92"
  marketplaceRevenue="31.69" operatorRevenue="31.69"
  amountForSupplier="223.54"/>
```

The service is offered by a broker:

```
<CustomerRevenueShareDetails customerName="ITCompany"
  customerId="ff48ae0b" serviceRevenue="300.00"
  marketplaceRevenue="63.00" brokerRevenue="27.00"
  operatorRevenue="30.00" amountForSupplier="180.00" />
```

The service is offered by a reseller:

```
<CustomerRevenueShareDetails customerName="YourCompany"
  customerId="859069d7" serviceRevenue="500.00"
  marketplaceRevenue="80.00" resellerRevenue="250.00"
  operatorRevenue="50.00" amountForSupplier="120.00"/>
```

RevenuePerMarketplace

Provides an overview of the revenue shares for the different organizations involved in selling the services of a supplier on a specific marketplace.

A `RevenuePerMarketplace` element has the following attributes:

- **`serviceRevenue`** - (required, data type `decimal`, scale 2) Total revenue for all relevant services in the billing period.
- **`marketplaceRevenue`** - (required, data type `decimal`, scale 2) Total revenue share for the marketplace owner.
- **`brokerRevenue`** - (optional, data type `decimal`, scale 2) Total revenue share for all brokers.
- **`resellerRevenue`** - (optional, data type `decimal`, scale 2) Total revenue share for all resellers.
- **`operatorRevenue`** - (required, data type `decimal`, scale 2) Total revenue share for the operator.
- **`overallRevenue`** - (required, data type `decimal`, scale 2) Total revenue for the supplier. This is the remaining value of the total revenue for all services after subtracting the revenue shares for the operator, marketplace owner, brokers, and/or resellers.

Example:

```
<RevenuePerMarketplace serviceRevenue="80905.00"
  marketplaceRevenue="8090.50"
  resellerRevenue="0.00"
  brokerRevenue="20226.25"
  operatorRevenue="8899.55"
  overallRevenue="43688.70"/>
```

Glossary

Administrator

A privileged user role within an organization with the permission to manage the organization's account and subscriptions as well as its users and their roles. Each organization has at least one administrator.

Application

A software, including procedures and documentation, which performs productive tasks for users.

Billing System

A system responsible for calculating the charges for using a service.

Broker

An organization which supports suppliers in establishing relationships to customers by offering the suppliers' services on a marketplace, as well as a privileged user role within such an organization.

Cloud

A metaphor for the Internet and an abstraction of the underlying infrastructure it conceals.

Cloud Computing

The provisioning of dynamically scalable and often virtualized resources as a service over the Internet on a utility basis.

Customer

An organization which subscribes to one or more marketable services in ESCM in order to use the underlying applications in the Cloud.

Infrastructure as a Service (IaaS)

The delivery of computer infrastructure (typically a platform virtualization environment) as a service.

Marketable Service

A service offering to customers in ESCM, based on a technical service. A marketable service defines prices, conditions, and restrictions for using the underlying application.

Marketplace

A virtual platform for suppliers, brokers, and resellers in ESCM to provide their services to customers.

Marketplace Owner

An organization which holds a marketplace in ESCM, where one or more suppliers, brokers, or resellers can offer their marketable services.

Marketplace Manager

A privileged user role within a marketplace owner organization.

Operator

An organization or person responsible for maintaining and operating ESCM.

Organization

An organization typically represents a company, but it may also stand for a department of a company or a single person. An organization has a unique account and ID, and is assigned one or more of the following roles: technology provider, supplier, customer, broker, reseller, marketplace owner, operator.

Organizational Unit

A set of one or more users within an organization representing, for example, a department in a company, an individual project, a cost center, or a single person. A user may be assigned to one or more organizational units.

OU Administrator

A privileged user role within an organization allowing a user to manage the organizational units for which he has been appointed as an administrator, and to create, modify, and terminate subscriptions for these units.

Payment Service Provider (PSP)

A company that offers suppliers or resellers online services for accepting electronic payments by a variety of payment methods including credit card or bank-based payments such as direct debit or bank transfer. Suppliers and resellers can use the services of a PSP for the creation of invoices and payment collection.

Payment Type

A specification of how a customer may pay for the usage of his subscriptions. The operator defines the payment types available in ESCM; the supplier or reseller determines which payment types are offered to his customers, for example payment on receipt of invoice, direct debit, or credit card.

Platform as a Service (PaaS)

The delivery of a computing platform and solution stack as a service.

Price Model

A specification for a marketable service defining whether and how much customers subscribing to the service will be charged for the subscription as such, each user assigned to the subscription, specific events, or parameters and their options.

Reseller

An organization which offers services defined by suppliers to customers applying its own terms and conditions, as well as a privileged user role within such an organization.

Role

A collection of authorities that control which actions can be carried out by an organization or user to whom the role is assigned.

Seller

Collective term for supplier, broker, and reseller organizations.

Service

Generally, a discretely defined set of contiguous or autonomous business or technical functionality, for example an infrastructure or Web service. ESCM distinguishes between technical services and marketable services, and uses the term "service" as a synonym for "marketable service".

Service Manager

A privileged user role within a supplier organization.

Standard User

A non-privileged user role within an organization.

Software as a Service (SaaS)

A model of software deployment where a provider licenses an application to customers for use as a service on demand.

Subscription

An agreement registered by a customer for a marketable service in ESCM. By subscribing to a service, the customer is given access to the underlying application under the conditions defined in the marketable service.

Subscription Manager

A privileged user role within an organization with the permission to create and manage his own subscriptions.

Supplier

An organization which defines marketable services in ESCM for offering applications provisioned by technology providers to customers.

Technical Service

The representation of an application in ESCM. A technical service describes parameters and interfaces of the underlying application and is the basis for one or more marketable services.

Technology Manager

A privileged user role within a technology provider organization.

Technology Provider

An organization which provisions applications as technical services in ESCM.