FUJITSU

Systemwalker Service Catalog Manager V15.1.0 (Business Support System)

Developer's Guide

B1WS-0985-02ENZ0(00)
July 2012

Table of Contents

About this Manual

This manual describes the public Web service interface of Systemwalker Service Catalog Manager - hereafter referred to as Business Support System (BSS) - and how to use it to integrate applications and external systems with BSS.

The manual is structured as follows:

Chapter	Description
Introduction	Provides an overview of developer's tasks, and describes the basic concepts of Web services.
BSS Platform Services on page 8	Describes the BSS platform services as well as their purpose and usage.
Integrating Applications with BSS on page 12	Describes how to implement the interfaces between an application and BSS.
Integrating External Process Control on page 24	Describes how to integrate an external process control system with BSS.
Integrating Certificates for Trusted Communication on page 26	Provides an introduction to the usage of certificates for securing communication between BSS and an application.
Billing Data File on page 30	Describes the elements of an XML file created by exporting billing data.

Readers of this Manual

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

This manual assumes that you are familiar with the following:

- BSS concepts as explained in the *Overview* manual
- Basic Web service concepts
- XML and the XSD language
- Web services standards SOAP and WSDL
- 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	The names of graphical user interface elements like menu options are shown in boldface.
`init`	System names, for example, command names, and text that is entered from the keyboard are shown in Courier font.
`<variable>`	Variables for which values must be entered are enclosed in angle brackets.

`[option]`	Optional items, for example, optional command parameters, are enclosed in square brackets.
`one | two`	Alternative entries are separated by a vertical bar.
`{one | two}`	Mandatory entries with alternatives are enclosed in curly brackets.

Abbreviations

This manual uses the following abbreviations:

BSS	Business Support System
CA	Certification authority
JAX-WS	Java API for XML Web services
JSP	Java Server Pages
PaaS	Platform as a Service
PSP	Payment service provider
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 BSS is available:

- *Overview:* A PDF manual introducing BSS. It is written for everybody interested in BSS and does not require any special knowledge.
- *Online Help:* Online help pages describing how to work with the user interface of BSS. The online help is directed and available to everybody working at the user interface.
- *Installation Guide:* A PDF manual describing how to install and uninstall BSS. It is directed to operators who set up and maintain BSS in their environment.
- *Operator's Guide:* A PDF manual for operators describing how to administrate and maintain BSS.
- *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 BSS.
- *Supplier's Guide:* A PDF manual for suppliers describing how to define and manage service offerings for applications that have been integrated with BSS.
- *Marketplace Owner's Guide:* A PDF manual for marketplace owners describing how to administrate and customize marketplaces in BSS.
- *Developer's Guide:* A PDF manual for application developers describing the public Web service interface of BSS and how to use it to integrate applications and external systems with BSS.
- Javadoc documentation for the public Web service interface of BSS and additional resources and utilities for application developers.

1 Introduction

Business Support System (BSS) is a set of services which provide all business-related functions and features required for turning on-premise software applications into Software as a Service (SaaS) 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, BSS covers all the business-related aspects of a Platform as a Service (PaaS) or Cloud platform. It 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 BSS framework involves different users and organizations:

- **Technology providers** (e.g. independent software vendors) technically prepare their applications for usage in the Cloud and integrate them with BSS. They register the applications as technical services in BSS.
- **Suppliers** (e.g. independent software vendors or sellers) define service offerings, so-called marketable services, for the technical services in BSS. They publish the services to a marketplace.
- **Customers** register themselves or are registered by a supplier in BSS 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 suppliers publish their services.
- **Operators** are responsible for installing and maintaining BSS.

Developers in different organizations can use the BSS Web service interface for implementing applications that make use of BSS features or for integrating external systems with BSS.

1.1 The Developer's Tasks in BSS

As a developer, you integrate BSS with external applications and systems. 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 BSS. 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 BSS.
- Supplier or customer: You integrate BSS with an external process control system in order to control the execution of specific BSS actions. This involves implementing a notification service which is invoked through appropriate triggers configured in BSS. The triggers are defined and managed by the administrator of the affected organization. For details, refer to the BSS online help.

For performing your tasks, you use the public Web service interface of BSS. This interface, its documentation and additional resources, templates, samples, and utilities are provided in the BSS integration package (`fujitsu-bss-integration-pack.zip` file). A detailed documentation for these resources is provided as Javadoc. By opening the `readme.htm` file of the integration package, you can access the available Javadoc documentation as well as the resources themselves.

1.2 Web Services Concepts

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.

BSS provides its functionality as Web services. The public Web services as well as additional resources, templates, samples, and utilities can be used to integrate applications with BSS in order to make them available as services to customers.

The BSS Web services are written in Java using JAX-WS.

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

- WSDL (Web Service Description Language)

  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`).

2 BSS Platform Services

BSS exposes its basic functionality at a public Web service interface which consists of the so-called platform services. An application that integrates with BSS invokes the functionality of the 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
- Reporting service
- Review service
- SAML service
- Search service
- Service provisioning service
- Session service
- Subscription management service
- Tag service
- Trigger service
- Trigger definition service
- VAT service

The platform services are available in the BSS integration package.

The following sections contain a description of the purpose and usage of each service. For details on the methods of the services and the related data objects, refer to the Javadoc documentation.

Account Management Service

Java class name: `com.fujitsu.bss.intf.AccountService`

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

Billing Service

Java class name: `com.fujitsu.bss.intf.BillingService`

This service is used for exporting billing data.

A supplier can export the billing data from BSS and process it using billing and payment facilities that have already been established in the supplier's organization. This is the typical procedure for customers who decide to pay on receipt of invoice.

The result of the export is stored in an XML file, the billing data file. For detailed information on the elements of the XML file, refer to *Billing Data File* on page 30.

Categorization Service

Java class name: `com.fujitsu.bss.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 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 class name: `com.fujitsu.bss.intf.DiscountService`

This service is used for retrieving discount values.

Event Management Service

Java class name: `com.fujitsu.bss.intf.EventService`

This service is used for recording events which are generated during the operation of applications that are integrated with BSS. 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 BSS event management, refer to *Integrating with BSS Event Management* on page 22.

Identification Service

Java class name: `com.fujitsu.bss.intf.IdentityService`

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

Marketplace Management Service

Java class name: `com.fujitsu.bss.intf.MarketplaceService`

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

Reporting Service

Java class name: `com.fujitsu.bss.intf.ReportingService`

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

BSS 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 BSS user interface and exported to different file formats.

Review Service

Java class name: `com.fujitsu.bss.intf.ReviewService`

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

SAML Service

Java class name: `com.fujitsu.bss.intf.SamlService`

This service is used for the SAML-based single sign-on mechanism. It creates a SAML assertion based on the credentials of the logged-in user, which can be sent to a service provider or application

to grant access to specific resources. To use the SAML-based single sign-on mechanism for an application integrated with BSS, the user access type must be defined and implemented for it. For details on access types, refer to the *Technology Provider's Guide*.

Search Service

Java class name: `com.fujitsu.bss.intf.SearchService`

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

Service Provisioning Service

Java class name: `com.fujitsu.bss.intf.ServiceProvisioningService`

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

Session Service

Java class name: `com.fujitsu.bss.intf.SessionService`

This service is used for storing, retrieving, and deleting BSS session data. If you need to implement your own token handler and logout listener for an application integrated with BSS, you have to implement calls to methods of this service. For details, refer to *Adapting the Login/Logout Implementation* on page 17.

Subscription Management Service

Java class name: `com.fujitsu.bss.intf.SubscriptionService`

This service is used for managing subscriptions. Depending on the instance provisioning mechanism of your applications integrated with BSS, you may have to implement calls to methods of this service in the provisioning service. For details, refer to *Implementation Details* on page 15.

Tag Service

Java class name: `com.fujitsu.bss.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 class name: `com.fujitsu.bss.intf.TriggerService`

This service is used for retrieving and manipulating trigger process data. These data are used in a notification service to approve or reject BSS actions in a process control system. For details on how to integrate external process control systems, refer to *Integrating External Process Control* on page 24 and the online help.

Trigger Definition Service

Java class name: `com.fujitsu.bss.intf.TriggerDefinitionService`

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 BSS online help.

VAT Service

Java class name: `com.fujitsu.bss.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*.

3 Integrating Applications with BSS

Integrating an application with BSS involves the following implementation tasks:

- Implement a provisioning service
- Adapt the login/logout implementation
- Integrate the application with the BSS event management
- Implement service operations

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

3.1 Prerequisites

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

- If your application is written in Java and you are using the Eclipse IDE to develop the integration between the application and BSS, we recommend you to use Eclipse version 3.5 or higher.
- 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.
- You have decided on the authentication type to be used. For details refer to *Introduction* on page 26.

3.2 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 BSS. The provisioning service is called by BSS 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 proxy or external access type:

- With proxy access, any interaction including login takes place through BSS. Your application uses BSS for user authentication only; BSS does not forward login information to your application.
- With external access, 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 client and the application without involving BSS in any way.

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 BSS 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. BSS 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 takes longer, for example, because long-running processes or manual steps are involved. In this case, the provisioning service notifies BSS 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 BSS using the subscription management service when provisioning is either complete or cannot be completed.

- **Application parameters**

  An application may have parameters that are of relevance for the service provisioning in BSS. 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.

  BSS 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 BSS, you need to implement user management operations. These operations are called when a customer assigns users to a subscription in BSS, when users are deassigned from a subscription, 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 client** (see *Implementing the Service as a Java Client* on page 13)
- Implementing the service as a **non-Java client** (see *Implementing the Service as a Non-Java Client* on page 15)

3.2.1 Implementing the Service as a Java Client

It is recommended to use the JAX-WS framework for implementing a provisioning service. BSS ships the Java interfaces for its Web services as a `.jar` file. You can directly write your Java code and include this `.jar` file for calls from BSS to the application (outbound calls), or implement the Java interfaces for calls from the application to BSS (inbound calls). No code generation is required.

Both call directions can be deployed in Java EE environments:

- **Outbound calls**:

For outbound Web service calls, JAX-WS offers APIs to directly address a Web service specified by a WSDL file and to obtain a proxy implementation of the interface.

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

```
final URL wsdlLocation = new
URL("http://myServer:8090/ServiceProvisioningService/v1.2/BASIC?wsdl")
final javax.xml.namespace.QName serviceName = new QName
      ("http://bss.fujitsu.com/xsd/v1.2", "SessionService");
final javax.xml.ws.Service service = Service.create
      (wsdlLocation, serviceName);
final SessionService sessionService = service.
      getPort(SessionService.class);

//now the service can be directly called just as if the implementation
//were locally available.
sessionService.resolveUserToken(...);
```

- **Inbound calls**:

  Providing a Web service implementation that can be used for inbound calls means that you implement the 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 BSS, for example, as shown here:

```
@WebService(serviceName = "ProvisioningService",
          targetNamespace = "http://bss.fujitsu.com/xsd/v1.2",
          endpointInterface =
"com.fujitsu.bss.provisioning.intf.ProvisioningService")
public class MyProvisioningServiceImpl implements ProvisioningService
{
   //...
}
```

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

  If the application is a Java application but not deployed in a Java EE-compliant application server, JAX-WS offers APIs to publish the Web services:

```
import javax.xml.ws.Endpoint;
public static void main(String[] args) {
   Endpoint.publish(
      "http://www.example.com/MyService/provisioningServices",
      new MyProvisioningServiceImpl());
}
```

3.2.2 Implementing the Service as a Non-Java Client

Non-Java clients need to conform to the WSDL and XSD files shipped with BSS: 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.

3.2.3 Implementation Details

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 `modifyParameterSet` method.
3. If users access your application through BSS, implement the following methods:
   - `createUsers`
   - `deleteUsers`
   - `updateUsers`
4. If you are using asynchronous instance provisioning, you need to implement calls to the following methods of the subscription management service (see *BSS Platform Services* on page 8):
   - `abortAsyncSubscription`
   - `completeAsyncSubscription`
5. 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 by specific user actions in BSS. 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

Action at the BSS user interface	Method	Behavior
Import service definition Click a service entry to view the details.	`sendPing()`	`sendPing()` is called for all access types except proxy and external.

Supplier

Action at the BSS user interface	Method	Behavior
Terminate a subscription	`deleteInstance()`	`deleteInstance()` is called for each deleted subscription and for all access types except proxy and external.

Action at the BSS user interface	Method	Behavior
Activate or deactivate services Activate a service.	`sendPing()`	`sendPing()` is called for all access types except proxy and external.
Manage payment types Deactivate a payment type for a customer.	`deactivateInstance()`	`deactivateInstance()` is called for all access types except proxy and external if the customer already uses the payment type to pay for his subscriptions. The method is called for each affected subscription.

Customer

Action at the BSS user interface	Method	Behavior
Subscribe to a service	`createInstance()` `asyncreateInstance()`	`createInstance()` is called for platform, login, direct, and user access in synchronous mode. `asyncreateInstance()` is called for platform, 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 platform, 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 platform, 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 platform, 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.

Action at the BSS user interface	Method	Behavior
Modify a subscription	`modifyParameterSet()`	`modifyParameterSet()` is called for all access types except proxy and external. The method is called only if customizable parameters are changed by the customer.
Up/Downgrade a subscription	`modifyParameterSet()`	`modifyParameterSet()` is called for all access types except proxy and external.
Terminate a subscription	`deleteInstance()`	`deleteInstance()` is called for all access types except proxy and 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 BSS provisioning API shipped in the BSS integration package.

3.3 Adapting the Login/Logout Implementation

If you opt for access through BSS (platform or login access), you need to adapt the login/logout implementation of your application and implement the 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 BSS 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 a custom login module.
- The **custom login module** lets users log in to the application without requesting any further credentials. Users are trusted because they have been authenticated by BSS. 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 BSS, a direct login to your application must be bypassed.
- The **custom logout module** closes user sessions on the application side and redirects users to the logout page of the underlying application. The URL of the logout page is returned by the `deleteServiceSession` method of the BSS session service.
- The **logout listener** notifies BSS when a user logs out or a session timeout occurs.

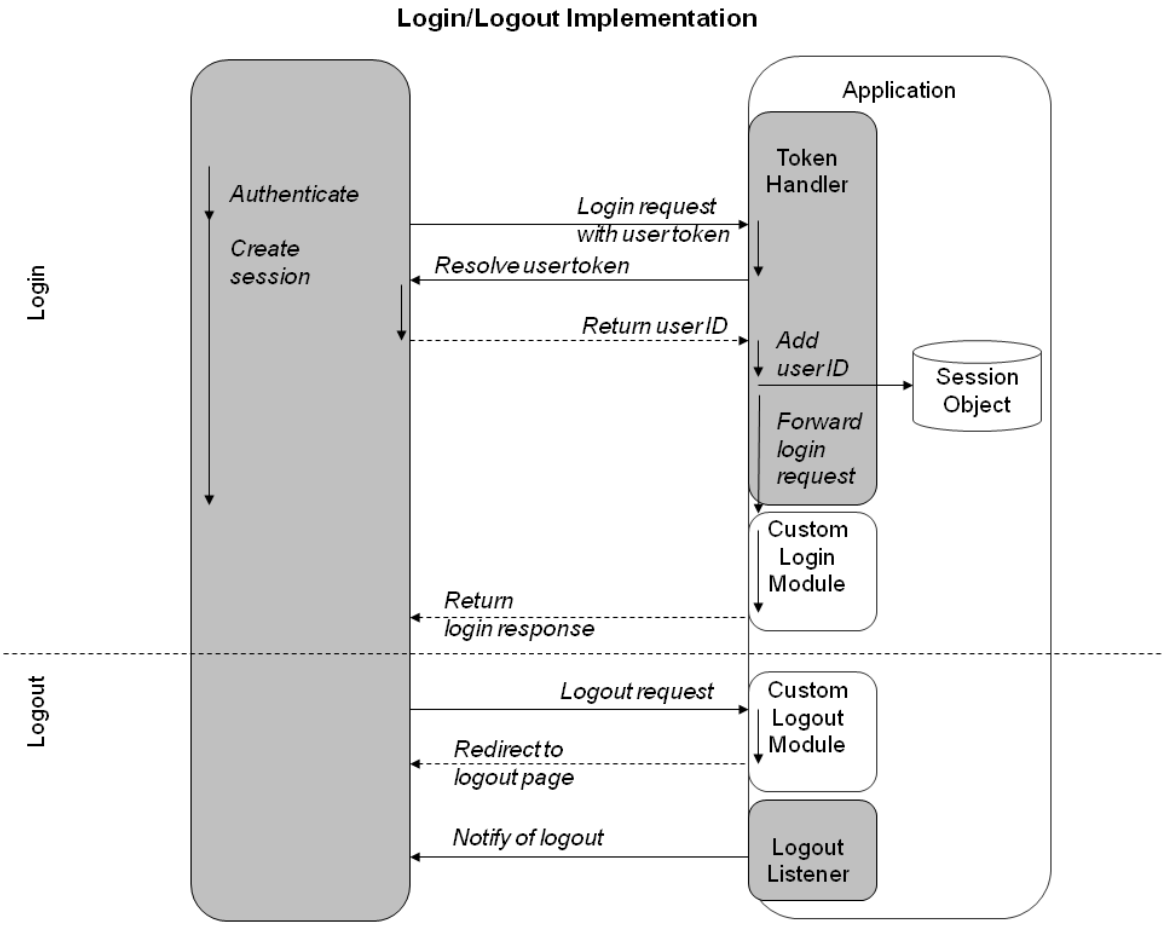
As for the custom login and logout modules, you need to analyze the existing functionality of your application and adapt it to match the required behavior.

As for the remaining functionality, BSS provides a Web application package (`Integrationhelper.war`) implemented in Java, the so-called integration helpers. The package supports you in adapting the login/logout implementation of a Java-based Web application. It contains a token handler and a logout listener ready to use. The integration helpers are available in the BSS integration package.

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

Package Contents	Description
`tokenhandler.jsp`	Token handler implemented as JSP.
`WEB-INF/lib`	A folder containing all JAR files required by the BSS integration helpers.
`WEB-INF/classes/tokenhandler.properties`	A property file specifying the fully qualified URL (e.g. `http://myServer:8090/ServiceProvisioningService/v1.2/BASIC?wsdl`) to the BSS platform services and 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/web.xml`	A sample configuration file with entries for the token handler servlet and the logout listener.
`WEB-INF/classes/com/fujitsu/bes/integrationhelper/LogoutListener.class`	Logout listener class.

The following figure illustrates the functionality of the token handler and the logout listener provided with the integration helpers (components provided by BSS are shown in gray):



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 session service (see *BSS Platform Services* on page 8):

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

HTTP Parameters of the Login Request

The login request sent by BSS contains the following HTTP parameters:

- `usertoken` (String)

  User token that can be resolved to a user ID in conjunction with the `saasId` parameter.

- `saasId` (String)

  ID needed by the application for further communication with BSS. The ID is generated by BSS.

- `instanceId` (String)

  Identifies any items that belong to a subscription on the application side. The ID is generated by the application and returned to BSS by the instance provisioning call. When you implement a provisioning service, the instance ID must be generated by your provisioning service.

- `subKey` (String)

  Identifies a subscription. This key can be used for the `recordEventForSubscripton` method when integrating with the BSS event management.

- `language` (String)

  Language to be used for the interaction with the user.

- `contextPath` (String)

  Optional. Local application-specific context.

To integrate the BSS integration helpers into your Web application:

1. Extract the `Integrationhelper.war` package from the BSS integration package to a location of your choice.
2. Adapt the following information in the `tokenhandler.properties` file to your needs:
   - Server and port where the BSS platform services have been deployed
   - Path to your custom login module
3. Copy the contents of the `WEB-INF/lib` and `WEB-INF/classes` folders to the corresponding folders of your Web application.
4. 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.
5. 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.

In the technical service definition of the application, the path to the token handler must be specified in the `loginPath` attribute for login and platform access.

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

Example

Suppose you implemented a JSP-based or 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 at `/DoAutoLogin`. Your Web application is running on `myserver` at port `7777` using the context `myapplication`. The BSS platform services are listening at port `8082`.

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

- tokenhandler.properties

```
TOKENHANDLER_BSS_HOST=http://localhost:8080/SessionService/v1.2/BASIC?wsdl
TOKENHANDLER_FORWARD=/DoAutoLogin
```

- web.xml

```
...
<servlet>
   <display-name>TokenhandlerServlet</display-name>
   <servlet-name>TokenhandlerServlet</servlet-name>

<servlet-class>com.fujitsu.bss.integrationhelper.TokenhandlerServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>TokenhandlerServlet</servlet-name>
   <url-pattern>/resolveToken</url-pattern>
</servlet-mapping>

<listener>

<listener-class>com.fujitsu.bss.integrationhelper.LogoutListener</listener-class>
</listener>
...
```

- Technical service definition

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

JSP-Based Token Handler

- tokenhandler.properties

```
TOKENHANDLER_BSS_HOST=http://myServer.com:8082
TOKENHANDLER_FORWARD=/DoAutoLogin
```

- web.xml

```
...
<listener>

<listener-class>com.fujitsu.bss.integrationhelper.LogoutListener</listener-class>
</listener>
...
```

- Technical service definition

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

```
    ...
>
...
```

3.4 Integrating with BSS Event Management

The event management service in BSS 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 BSS at runtime through the event management service, which is one of the BSS platform services.

To integrate an application with BSS 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 BSS by calling one of the following methods of the event management service (see *BSS Platform Services* on page 8):
   - `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 BSS platform services.
3. When preparing the service definition, declare the events that your application will send.

Note: BSS comes with two predefined events, which can be used with platform and 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 17.

3.5 Implementing Technical Service Operations

You may wish that your technical service offers additional operations or functions that are to be accessible via BSS without opening the application. 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. Integrating such operations in your technical service simplifies the integration without the need to enhance the application by another interface that is reachable via the Web.

The information about the operations provided by the technical service must be added to its XML definition. For details, refer to the *Technology Provider's Guide*.

To integrate a technical service operation:

1. Implement the `executeServiceOperation` method in a Web service definition using the `OperationService` interface of the operation API, which is part of the BSS integration package.

   The following information is provided as parameter settings:
   - 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.

- The transaction identifier. Currently, this ID is always set to 0; it serves as a reference to a BSS internal transaction.
- 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*.

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 operations.

   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 String.
- Additional parameters are not allowed.
- BSS does not provide for any access control. Every user who can use the subscription can also execute the operations.
- The operations can only be executed in synchronous mode.

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 specific actions are carried out on the user interface or marketplace. 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 in the process control system. If no approval is required, the process control system is informed about the execution of the action.

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. For details on implementing such a service, contact the platform operator.

Users can see all pending actions at the BSS 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 BSS:

- For **customer** organizations:
  - A subscription is to be added (`Subscribe to service`).
  - A subscription is to be terminated (`Terminate subscription`).
  - A subscription is to be changed, for example, renamed (`Modify subscription`).
  - A subscription is to be upgraded or downgraded (`Up/Downgrade subscription`).
  - A user is to be assigned to or removed from a subscription (`Assign users to subscription`).
  - A billing run for a customer is completed, and the billing data for a billing period is calculated (`Start billing run`).
- For **suppliers**:
  - A customer is to be registered (`Register customer`).
  - The payment types for a service or 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 billing run for a customer is completed, and the billing data for a billing period is calculated (`Start billing run`).

Connecting to an External Process Control System

If you want BSS to connect to an external system to handle notifications and thus make use of the trigger processing feature of BSS, 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 for the `com.fujitsu.bss.notification.intf.NotificationService` interface. This interface and the required data objects are provided with the BSS 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 BSS. To do this, you must be an administrator of your organization in BSS.

   When defining a trigger, you need to provide the URL of the notification service WSDL file, for example: `http://myServer:8280/NotificationService?wsdl`.

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

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

   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 call one of the methods provided by the trigger service (see *BSS Platform Services* on page 8), and send its response back to BSS. As a result, the interrupted action is continued or canceled. When rejecting an action, the reason for the rejection can be passed.

5 Integrating Certificates for Trusted Communication

Certificates are required for BSS to authenticate a calling client and to allow for trusted communication between BSS and an application underlying a technical service or a payment service provider (PSP).

The following organizations are involved in handling certificates:

- Operator
- Technology providers integrating their applications with BSS
- PSPs whose services are to be integrated with BSS for invoicing and payment collection.
- Any other organization using Web service calls to or from BSS

5.1 Introduction

Web service calls coming from BSS (e.g. for provisioning and process integration) or sent to it 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, BSS can act as a server (Web service calls to BSS) or as a client (Web service calls from BSS).

The client must provide its authenticating data to the server. Two options are available:

- **Authentication with basic authentication**:

  The caller sends the key and password of a BSS user. The user key can be looked up at the BSS user interface on the **Edit Profile** page. This authentication mechanism is not related to certificates.

  When using basic authentication, the access to Web services should be restricted to SSL/TLS communication and HTTPS. Since the BSS APIs are accessible through JNDI lookups, configure your network's firewall to block JNDI lookups from the outside.

- **Authentication with certificates**:

  The caller provides a certificate to the BSS server. In this case, the following requirements must be fulfilled:

  - The distinguished name (DN) of the client's certificate must correspond to the DN configured and stored in BSS for the corresponding organization.
  - BSS must trust the client's certificate: The BSS truststore must contain a certificate with a valid signing chain to the certificate presented by the client.

BSS 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.

5.2 Requirements for Web Service Calls from BSS

For provisioning and process integration, BSS calls other Web services which can be addressed by HTTPS. In this scenario, BSS 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 must correspond to the server's domain name and it must be valid at the time of calling.
- The client (BSS) must trust the server's certificate. To this end, the client's truststore must contain a certificate with a valid signing chain to the certificate presented by the server.

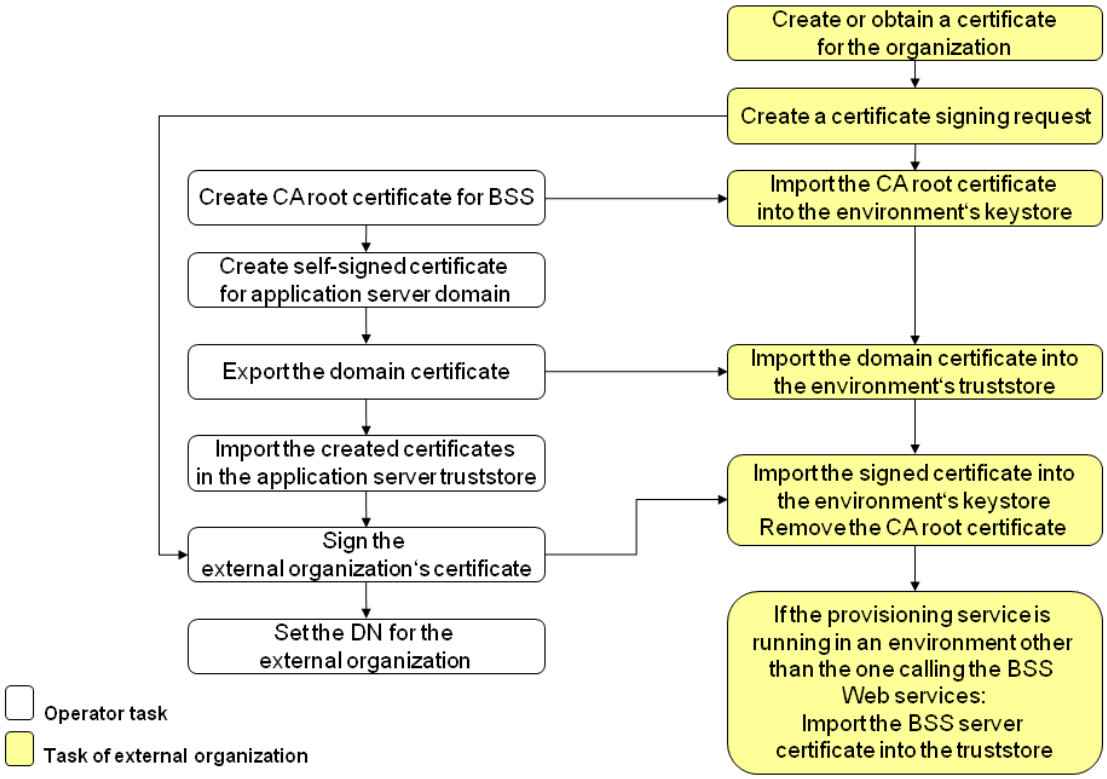
5.3 Requirements for Web Services Calls to BSS

BSS provides its Web services (platform services) that can be called by other systems. In this scenario, BSS is the HTTPS server while the other system is a Web service client. The following requirements must be met to establish a connection to BSS:

- The BSS server must present a valid certificate: The CN must correspond to the server's domain name and it must be valid at the time of calling.
- The Web service client must trust the server's certificate: The client's truststore must contain a certificate with a valid signing chain to the certificate presented by the server.

5.4 Certificate Integration Procedures

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



The tasks of your organization to make use of certificate-based authentication are described in detail in the subsequent sections. The operator tasks are described in the *Operator's Guide*.

5.4.1 Creating a Certificate and a Signing Request

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.

Create a certificate signing request (CSR) by executing the following command:

```
%JAVA_HOME%\bin\keytool -certreq -alias tpcert -file tpcert.csr -keystore
 keystore.jks
```

The alias `tpcert` refers to the created certificate. The result of the above command is a file name `tpcert.csr`. Send this file to the certification authority (CA), for example, the operator.

5.4.2 Importing the Signed Certificates

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

- CA public (root) certificate for BSS
- 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 public 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 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
```

5.4.3 Importing the BSS Server Certificate

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

Execute the following command:

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

Appendix A: Billing Data File

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

A supplier can export the billing data for one or several of his customers for a specific time. The exported data can then be forwarded, for example, to an accounting system which continues to process the data seamlessly.

The result of the export is stored in an XML file, the billing data file. For each customer, a separate billing data file is created. The billing data file conforms to the XML schema `BillingResult.xsd`, which can be found in the BSS 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 30)
- `OrganizationDetails` (see *OrganizationDetails* on page 31)
- `Subscriptions` (see *Subscriptions* on page 31)
- `OverallCosts` (see *OverallCosts* on page 44)

A `BillingDetails` element has the following attribute:

`key` - (optional, data type `long`) Unique identifier allowing, for example, accounting systems to relate the 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.

Period

Specifies the billing period.

The calculation of the charges is based on milliseconds. The billing period, i.e. the period of time for which charges are calculated, is one month.

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:

```
<BillingDetails key="10002">
  <Period endDate="1306879200000"
endDateIsoFormat="2011-05-31T22:00:00.000Z" startDate="1304200800000"
startDateIsoFormat="2011-04-30T22:00:00.000Z"/>
...
</BillingDetails>
```

OrganizationDetails

Provides details of the customer for which the billing data have been exported.

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">
...
  <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 and the custom attributes of the affected subscription are specified. A `Subscription` element also contains the billing data for the price model of the subscription in the `PriceModel` element.

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.

Example:

```
<BillingDetails key="10002">
...
```

```
  <Subscriptions>
    <Subscription id="Mega Office Basic" purchaseOrderNumber="12345">
      <PriceModels>
        <PriceModel 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">
...
  <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 33)
- `GatheredEvents` (see *GatheredEvents* on page 33)
- `PeriodFee` (see *PeriodFee* on page 34)
- `UserAssignmentCosts` (see *UserAssignmentCosts* on page 35)
- `OneTimeFee` (see *OneTimeFee* on page 36)
- `PriceModelCosts` (see *PriceModelCosts* on page 36)
- `Parameters` (see *Parameters* on page 37)

A `PriceModel` element has the following attribute:

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

UsagePeriod

Specifies the actual usage period for a subscription. A usage period begins when the customer subscribes to a service and ends when the subscription is terminated.

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 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 attribute is omitted.

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 41 and *SteppedPrices* on page 42). 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 id="14001">
...
  <GatheredEvents>
    <Event id="USER_LOGOUT_FROM_SERVICE">
      <Description xml:lang="en">Logout of a user 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.

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 `float`) 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 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 every user 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. A charge to be paid on a recurring basis per user can only be defined for services with the platform, login, or user access type.

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 per user and base period according to the price model. If the charge per user has stepped prices, this attribute is omitted.
- **`factor`** - (optional, data type `float`) 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 41 and *SteppedPrices* on page 42.
- **`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:

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 `float`) 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 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. A one-time fee is independent of the number of users, events, or other usage data.

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 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. If a discount was specified, the net amount of the costs is given in the `Discount` element (see *Discount* on page 43).

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.

- **grossAmount** - (required, data type `decimal`) Gross amount of the costs, calculated from the net costs (`amount`) plus VAT (see *VAT* on page 43). The value is rounded to two decimal places.
- **amount** - (required, data type `decimal`) Net amount of the costs for the subscription. The value is rounded to two decimal places.

Example:

```
<PriceModel id="14001">
...
  <PriceModelCosts currency="EUR" grossAmount="990.00" amount="900.00"/>
</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.

You can define a price for every parameter and option, and specify whether this price is to 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 prices for parameters and options are independent of other price model elements.

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

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. A new usage period begins when the parameter value is changed.

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.

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 `float`) 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.

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 per user and base period according to the price model.
- **`factor`** - (required, data type `float`) 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.
- **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 41.
- **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 41 and *SteppedPrices* on page 42). 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 `float`) 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.

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 per user and base period for the parameter option according to the price model.
- **`factor`** - (required, data type `float`) 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 41.
- **`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.

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.

Service roles can be used for role-specific prices: 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.

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 `float`) 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 **number of users** accessing a subscription. For example, 1 to 10 users cost 10.00 € per user, 11 to 100 users cost 8.00 € per user, any user above 100 costs 6.00 €.
- 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 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>
```

Discount

Specifies the discount granted to the customer.

For each customer organization, a discount can be defined which applies to all subscriptions of the customer. 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 for all billing data generated within the discount period.

A `Discount` element is contained in `OverallCosts` and `PriceModelCosts` elements.

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`). This value is identical to the value of the `amount` attribute of the `PriceModelCosts` element (see *PriceModelCosts* on page 36). 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:

```
<PriceModel id="14001">
...
  <PriceModelCosts currency="EUR" grossAmount="990.00" amount="900.00">
    <Discount discountNetAmount="100.00" netAmountAfterDiscount="900.00"
netAmountBeforeDiscount="1000.00" percent="10.00"/>
  </PriceModelCosts>
</PriceModel>
```

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.

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 basic VAT rate set of the supplier organization is used in all other cases.

A `VAT` element is contained in `OverallCosts` and `PriceModelCosts` elements.

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 (`amount` attribute of the `PriceModelCosts` element). The value is rounded to two decimal places.

Example:

```
<PriceModel id="14001">
...
  <PriceModelCosts currency="EUR" grossAmount="990.00" amount="900.00">
    <VAT amount="90.00" percent="10.00"/>
  </PriceModelCosts>
</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 has been specified, the net amount of the costs is given in the `Discount` element (see *Discount* on page 43).

An `OverallCosts` element has the following attributes:

- **`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 43). The value is rounded to two decimal places.
- **`netAmount`** - (required, data type `decimal`) Net costs after the net discount has been deducted from the original net costs (see *Discount* on page 43). The value is rounded to two decimal places.

Example:

```
<BillingDetails key="10002">
...
  <OverallCosts currency="EUR" grossAmount="990.00" netAmount="900.00"/>
  </OverallCosts>
</BillingDetails>
```

Glossary

Administrator

A privileged user role within an organization. Each organization has at least one administrator.

Application

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

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 BSS 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 BSS, based on a technical service. A marketable service defines prices, conditions, and restrictions for using the underlying application.

Marketplace

A virtual platform for suppliers in BSS to provide their services to customers.

Marketplace Owner

An organization which holds a marketplace in BSS, where one or more suppliers 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 BSS.

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, marketplace owner, operator.

Payment Service Provider (PSP)

A company that offers suppliers 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 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 BSS; the supplier 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.

Role

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

Service

Generally, a discretely defined set of contiguous or autonomous business or technical functionality, for example, an infrastructure or Web service. BSS 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 BSS. By subscribing to a service, the customer is given access to the underlying application under the conditions defined in the marketable service.

Supplier

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

Technical Service

The representation of an application in BSS. 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 BSS.