




Sigma's Service Management Platform  
Release 4.2

# ORDER MANAGER XML API GUIDE



[www.sigma-systems.com](http://www.sigma-systems.com)

## **Copyright © 1996-2009 by Sigma Systems Canada Inc. All rights reserved.**

Sigma Systems Canada Inc., Toronto, ON, Canada

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

Sigma, the Sigma logo and Sigma product names are trademarks of Sigma Systems Canada Inc., and may be registered in certain jurisdictions.

Other names may be trademarks of their respective owners.

# Contents

<b>Preface</b>	iii
Who should read this guide . . . . .	iii
Prerequisites Knowledge and Skills . . . . .	iii
Whats this guide cover . . . . .	iii
Notational conventions . . . . .	iii
How to contact Sigma Systems . . . . .	iv
 <b>SMP Overview</b>	 1
SMP Platform Environment . . . . .	1
Glossary of Terms . . . . .	1
SMP Architecture . . . . .	2
Order execution flow . . . . .	4
 <b>Service Definition and Subscriber Profile</b>	 5
Subscriber Structure Overview . . . . .	5
Service Catalog . . . . .	7
Service Definition . . . . .	7
Event Service . . . . .	17
Entities Definition . . . . .	18
Subscriber value example . . . . .	19
Entity state . . . . .	23
 <b>Order Manager Overview</b>	 27
Order Manager Architecture . . . . .	27
Order models and attributes . . . . .	28
Snapshot order . . . . .	29
Action Order . . . . .	30
OSSJ Order . . . . .	32
Batch Order . . . . .	34
Order Type Usage . . . . .	34
Order Life Cycle . . . . .	37
Create order . . . . .	37

Process Order . . . . .	38
Update Order . . . . .	48
Cancel Order . . . . .	49
Repair Order . . . . .	50
Replicate Order . . . . .	51
Query Order . . . . .	52
Order States . . . . .	53
Order Item States . . . . .	58
<b>Tools . . . . .</b>	<b>61</b>
SMP Web . . . . .	61
XmlApiUtil.sh . . . . .	61
SbNotfiy.sh . . . . .	62
Dump XML Order Request to XML File . . . . .	62
Review Order in XML from Database . . . . .	62
SmpDebug . . . . .	62
 <b>XML Order API . . . . .</b>	 <b>63</b>
WSDL Overview . . . . .	63
Client Samples . . . . .	65
XML over JMS . . . . .	65
Web Services . . . . .	66
XML over EJB . . . . .	67
XSD for XML Request . . . . .	67
XmlSmpCommonSchema.xsd . . . . .	69
XmlSmpCBEServiceSchema.xsd . . . . .	75
XmlSmpCBEServiceSchema.xsd . . . . .	82
Request, Response and Exception . . . . .	96
Notification . . . . .	112
Web Services samples in Order Manager SDK . . . . .	118
 <b>Appendix A - Order Parameters . . . . .</b>	 <b>121</b>
 <b>Appendix B - Xbeans Library Examples . . . . .</b>	 <b>123</b>
Constructing XML Objects . . . . .	123
Entity XMLObject . . . . .	123
order xml . . . . .	129
Order Request . . . . .	131

# Preface

This guide outlines the procedure for building the Order Manager XML API.

## *Who should read this guide*

This guide is intended for development teams who wants to build the Order Manager XML API.

## *Prerequisites Knowledge and Skills*

Users of the *Order Manager XML API Guide* are expected to be familiar with:

- J2EE
- Web services
- XML and XSD schema
- SMP High Level Architechure (see the *System Architecture Guide*)
- Service Creation Toolkit User Guide

## *Whats this guide cover*

Topics covered in this guide are as follows:

- Overveiw of the SMP
- Service Defintion and Subscriber Profile
- Overveiw of Order Manager
- XML Order API

## *Notational conventions*

The following notational conventions are used to distinguish elements of text throughout this guide:

- italicized serif text—used for book titles; for example, the *Installation Guide*; also used for *emphasized concepts*
- double quotations—used for chapter and section titles; for example, “Using the Service Catalog Manager”

- bold, sans-serif, title case—used for menu names, tabs, buttons, and fields; for example, click **Next**, click the **Services** tab
- monospaced font—used for codes, command lines, values, file, folder, and directory names; for example, the `Provisioning` folder
- angled brackets—used as placeholders for text supplied by users; for example, `<Last Name>`
- title case—used for subsystem, window, dialog box, and web page names; for example, the Subscriber Registration window
- blue type—used for hyperlinks and cross references in online documentation; for example, [authentication](#)
- uppercase text—used for device names, environment variables, and acronyms; for example, LPT1, IP address
- > — used to indicate a menu choice. For example, select File|Open, means click the File menu, and then click Open.

## *How to contact Sigma Systems*

If you have questions about the Service Creation Toolkit, please contact your Sigma Systems representative, who will be able to assist you in the form of technical support, professional services, and training.

We welcome your feedback on the documentation. Please provide your comments to [documentation@sigma-systems.com](mailto:documentation@sigma-systems.com).

# SMP Overview

*Service Management Platform* (SMP) provides the overall foundation for managing multiple services that are offered by a service provider. SMP delivers key capabilities required for service management including managing subscribers and subscribed service profiles with a detailed view of the service delivery networks capabilities. It integrates with an open architecture to business, operational, and network service platforms in order to facilitate the automation of service management processes.

## SMP Platform Environment

---

Environment of SMP Platform is made up of:

- JDK 1.5
- Weblogic 9.2
- Oracle 10g
- OSSJ Service Activation 1.0
- OSSJ Inventory API 1.0
- SOAP 1.1

## Glossary of Terms

---

Table 1: Glossary of Terms

Acronym	Description
JMF	Java message formatting language
SMP	Service Management Platform
STM	Service Topology Manager
OSS	operation support systems
JEA	Java element adapter
SCM	Service Catalog Manager
SCT	Service Catalog Manager
VoIP	Voice over IP
JVT	Java Value Types

Table 1: Glossary of Terms

Acronym	Description
BSS	Business Support Systems
NamedQuery	A query that is defined and stored, by name, in XML definition file for later retrieval and execution.
JMF Bean	Stateless session bean embed in JMF expression The syntaxes of the expression are: <ul style="list-style-type: none"><li>• jmf_bean[jndiName]methodName(exprListopt) OR</li><li>• jmf_bean.beanName.parameter (jndi name of bean must be like com.sigma.jmf.beanName)</li></ul>

## SMP Architecture

---

Sigma's SMP (Service Management Platform) is a real-time, J2EE-based intelligent service control engine that provides the foundation to manage subscription, on-demand and real-time VoIP, High Speed Internet, Wireless, Commercial Services, IPTV and Video services across any network technology, on any functional device.

The following daigram is a simplified view of SMP internal structure:



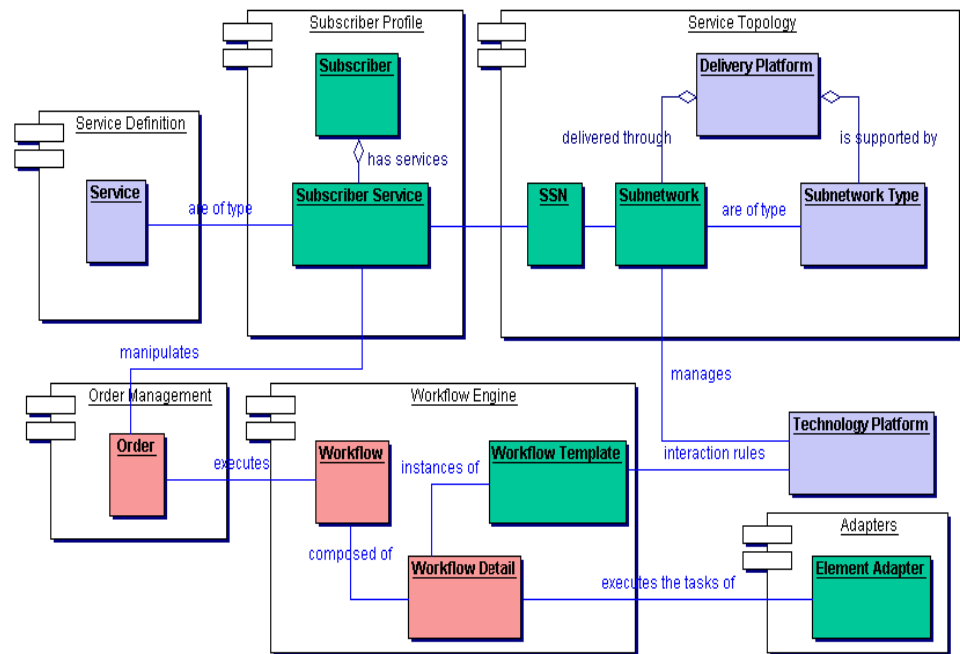


Figure 1: SMP Internal Structure

Where:

**Order Management** validates orders based on rules embedded in the Service and Topology Definitions, and decomposes orders into actions which are then executed via a workflow. *JVTActivationSession* and the respective Web Services present synchronized interfaces to the Order Manager. *JVTActivationSession* is accessible from all SMP or external components that handle Orders. Web Services are designed for external components and used for XML orders only.

**Subscriber Manager** (also called *SubMgr*) provides inventory management for entities like subscribers, services, addresses etc. The main interface is the *SampSubMgr*, which provides all the functionality for querying and making the prototype of subscriber, services and its associated entities.

**Service Topology Manager** (also called STM) maintains an up-to-date view of the Network Topology and its relation to services and subscribers. The topology models the infrastructure network. It is maintained and kept up-to-date by the network management personnel, the Customer Premises Equipment (CPE), and the Platform (SMP). This image of the topology allows the Platform's components to correctly initiate activation and diagnostic workflows, based on the location of the different elements involved and their relationships to other network elements.

**Workflow Engine** manages and executes business processes to provision services or diagnose problems. The engine uses BPEL-based workflow definition language represented in XML format.

**Service Definition** models the products and services that can be offered to subscribers, together with their business rules, in a form that can later be

employed by the components of the Platform. This definition has the richness and flexibility to model the rapidly evolving real word service definitions required by the operator's business.

## Order execution flow

A request from a BSS (Business Support System – such as billing and customer care), SelfCare WebUI, or SMP Web eventually turns into an SMP order. Here is the SMP order execution flow, a dynamic view of SMP:

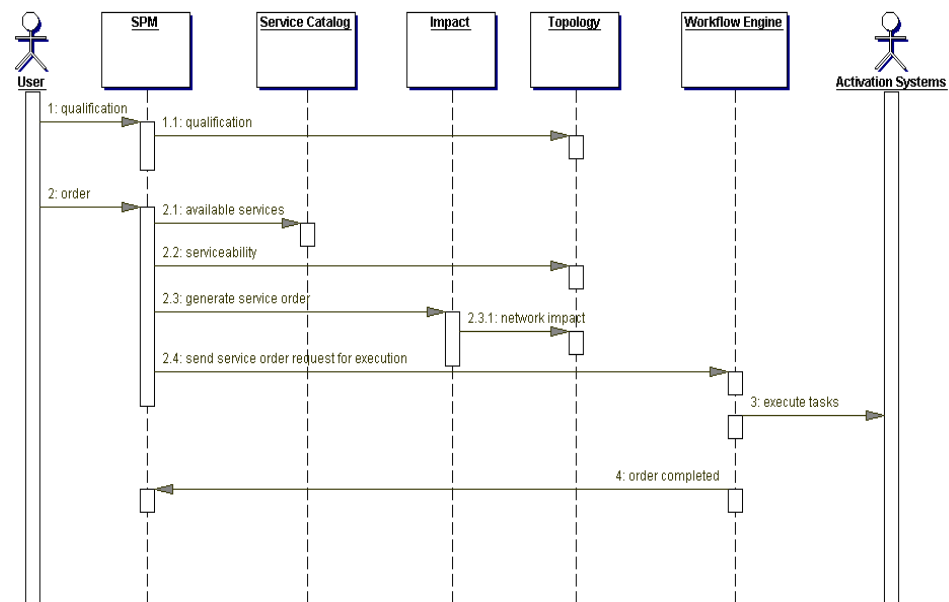


Figure 2: SMP Order Execution Flow

The most common way to execute SMP order is calling the `executeOrder()` function on `JVTActivationSession` Bean or sending `executeOrderRequest` to the web services interface. The call is blocked until Order Manager validates the request, updates the topology, generates the workflow request, and submits it to Workflow Engine.

Workflow Engine follows a BEPL-based request script and communicates with Element Adapters to execute the request, such as creating an email account or activating a cable modem. After processing the request, Workflow Engine notifies Order Manager that the request is done. The result can be completed or aborted. If it is aborted, Order Manager needs to work with other components in SMP to rollback the changes. After that, Order Manager will send out `order status change` notification with the order final state. The notification can be sent to a JMS Queue/Topic or a Web Service.

Other than sending `order status change` notifications, SMP can be configured to send out `line item status change` notifications and `workflow error` notifications.

# Service Definition and Subscriber Profile

An SMP order is always about change in a subscriber profile, for example, adding an email account. Subscriber profile is stored in Subscriber Manager. The physical repository can be SQL database, XML database or LDAP directory. How a profile is stored is transparent to the SMP client. For more details, see the Subscriber Manager SDK as shown below:

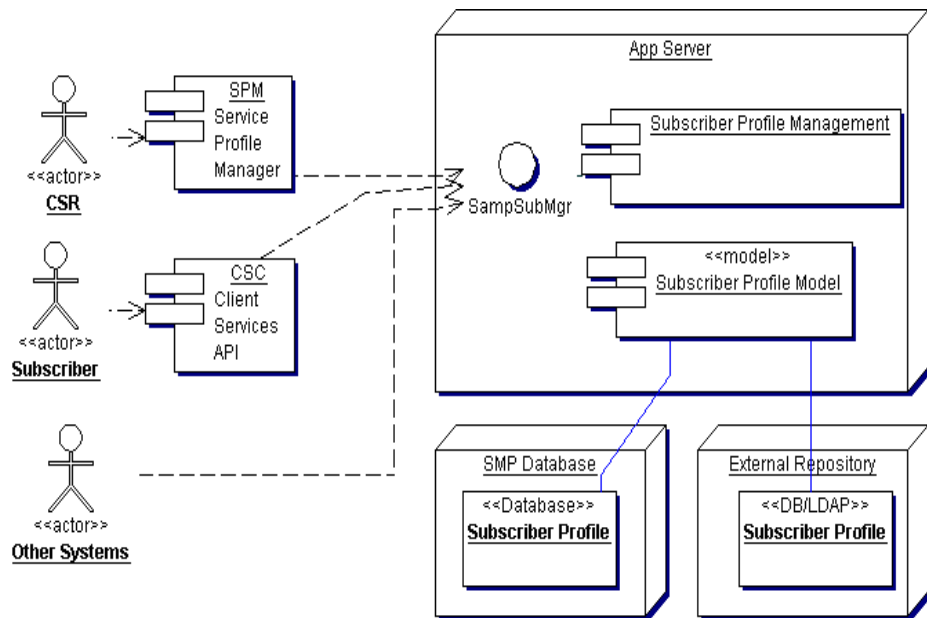


Figure 1: Subscriber Manager SDK

Following OSS Order Manager's design guideline, Subscriber Manager only explores query and get function to external system. All profile manipulation requests go through Order Manager.

## Subscriber Structure Overview

Each subscriber profile includes:

- Subscriber Parameters
- Contact information

Each subscriber has a set of contacts, uniquely identified by a "contact type".

- Subscriber Address

Each subscriber has a set of addresses, uniquely identified by a “preferred name”. Some addresses serve as Service Locations, representing the service delivery point.

- **Subscriber Services**

Subscriber Service can represent package, plan, feature (for example call waiting), device (for example cable modem) or provision service (such as high speed Internet).

- **Subscriber Users**

One subscriber can have a number of users. Each user can use or administer services within the subscriber through self-care WebUI.

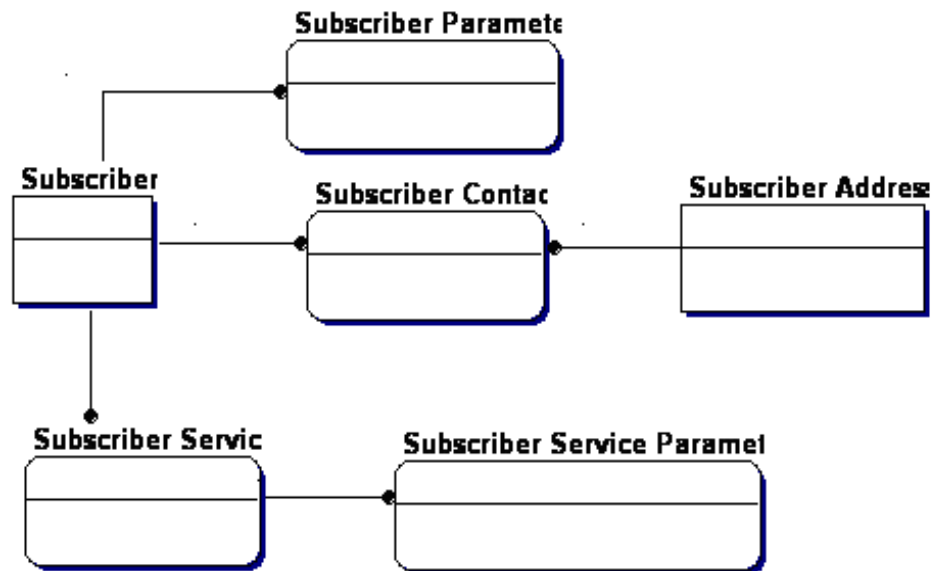


Figure 2: Subscriber Structure Overview

The relation between contact and address, service and service is called association. Here are the associations defined in SMP core:

- Service having children
- Address of Contact
- Service prerequisite
- Service parameter prerequisite
- Address of Service

SMP supports New Entity Type definition, for example, “Site Survey” defined in the SMP demo project. SMP also supports New Association Type definition, for example, “email needs hsd” association defined in the SMP demo project. For more details, see the *Subscriber Profile Manager User Guide* and the *Service Catalog Manager User Guide*.

Subscriber entity is used to represent an entity that belongs to a subscriber, such as subscriber service, contact, address, and any new solution defined type of entity.

The key of the subscriber entity is known as the subscriber entity key or entity key.

## Service Catalog

---

In SMP, subscriber profile definitions and validation rules are stored in Service Catalog Manager (SCM). Service Catalog defines:

- Parameter of subscriber, contact and address, and various validation rules
- Service and service hierarchy structure
- Service parameter and validation rules
- Service-to-service association
- Parameter of new type entity and various validation rules
- Entity-to-entity or entity-to-service association
- Order and order line item parameter
- Service impact and impact parameter
- Service migration rules

There are two UIs working with Service Catalog. Service Creation Toolkit (SCT) is used to view, create, and modify all the catalog definitions. SCM provides flexibility and ease in creating new service bundles, such as:

- viewing existing services
- modifying existing services
- creating services
- migrating services

## Service Definition

The Service Catalog Manager uses the following classification of service definitions:

- Provisionable service

*A provisionable service* is a basic service that may be offered to subscribers as such or bundled with other service offerings. Therefore, a provisionable service is considered the unit of configuration on which other service definitions rely.

- Subtyped services

*Subtyped services* are variations of provisionable services (base services) that inherit all the parameters and properties of the provisionable service. Subtyped services do not have new parameters, but some of the

parameters of the base service may be modified when creating a subtyped service. The parameters of a subtyped service may differ from the underlying base service in the following ways: by adding new validations, by setting default values, by switching Read Only to Read and Write and vice-versa. A Voice Feature Package with Voicemail is a subtyped service that may include Voicemail on Busy and Voicemail Do Not Disturb.

- Composed services

*Basic services* may be bundled together into *composed services*. A composed service consists of one or several components. The composed service takes the role of the “parent” within a composition association, while the component service takes the role of the “child”. Composed services are also called “plans” by broadband providers. For example, the composed service “My Basic Internet Service Package” includes the component services “My Email” and “My Webspaces”.

Provisionable, Subtyped, and composed services can be components of a composed service. Composed services may not be subtyped.

- Orderable services

Once a provisionable/subtyped/composed service is flagged as Orderable, it becomes *orderable service* and available in Service Profile Manager and can be offered to subscribers. Orderable services directly belong to subscriber and do not have composed service on top of it.

Here is an example of Cable Package Service Definition:

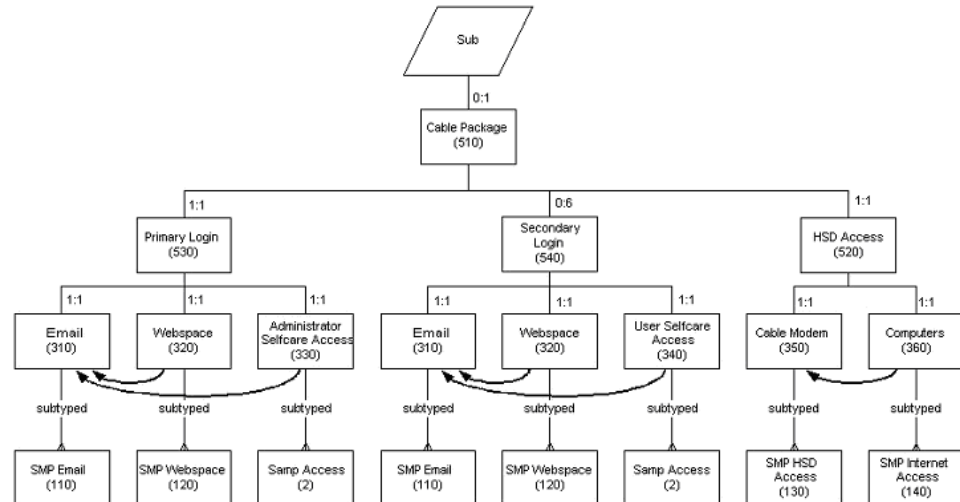


Figure 3: Cable Package Service Definition

In the Cable Package illustrated above, only the Cable Package service is orderable. The Primary Login, Secondary Login and HSD Access services are composed services. The Email, Webspaces, Administrator Selfcare Access, User Selfcare Access, Cable Modem and Computers services are subtyped services. The Provisionable services are: SMP Email, SMP Webspaces, SMP Access, SMP HSD Access and SMP Internet Access services.

## Service Definition to Subscriber Services mapping

Services in a subscriber profile (known as subscriber service) follow service definitions in Service Catalog. The rules for mapping definition to instance are:

- When a service is orderable in the catalog, its corresponding subscriber service is a top-level service; it does not have a parent.
- A provisionable or subtyped service has a corresponding instance in the subscriber if it is an orderable service or direct child service under a composed service.
- A composed service has a corresponding instance in the subscriber if it is an orderable service or child service under a composed service.

The following is an example of mapping:

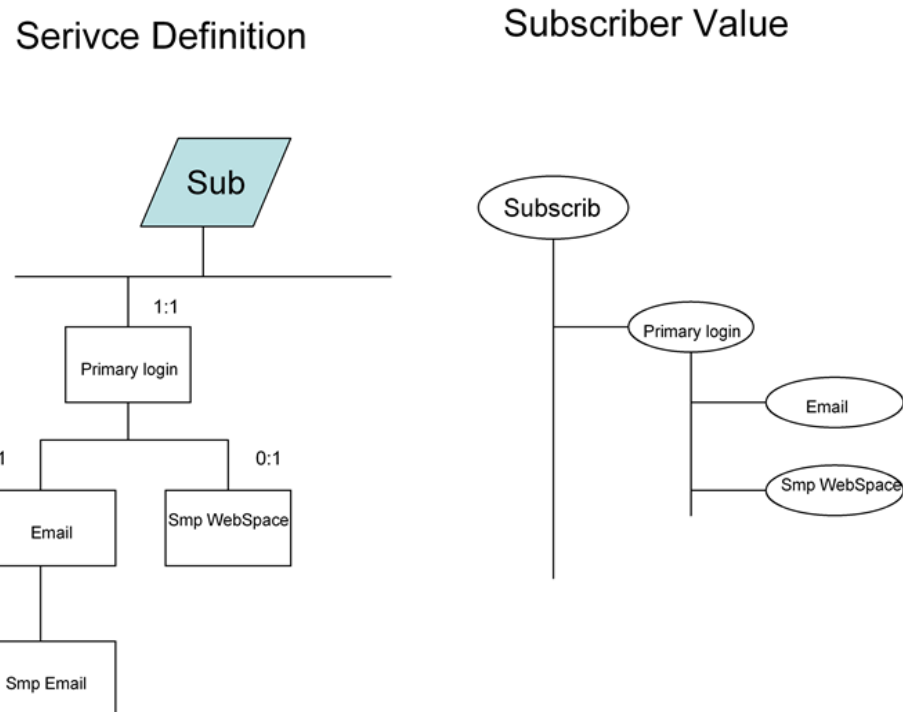


Figure 4: Service Definition & Subscriber Service Mapping

The left side of the diagram displays a service definition and the right side displays the subscriber service that follows the definition on the left side. “Primary Login” is an orderable composed service, so it has a corresponding subscriber service in the subscriber; it is also a top-level service without a parent. Email is a child service of “Primary Login”; it has a corresponding subscriber service on the right side. Smp Email is not a directly member of composed services and not orderable; it does not have a corresponding subscriber service.

In addition to defining the service structure, service definition also defines the maximum and minimum subscriber service instance number. In above example:

- 1:1 of Primary Login indicates that its instance is mandatory in the subscriber profile.
- 1:1 of email indicates that the “Primary Login” service instance must include one and only one Email service instance.
- 0:1 of iSmp WebSpace indicates the instance of iSmp WebSpace is optional to “Primary Login”. Each “Primary Login” can have not more than one instance of “Smp WebSpace”.

## Service Parameter

For provisionable and subtyped services, clicking on the SCT Parameters tab will show the parameters defined for the service:

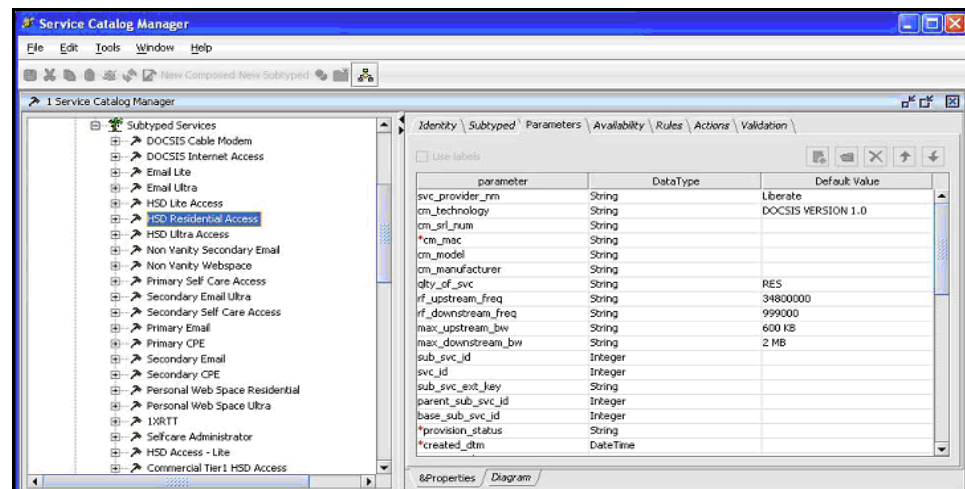


Figure 5: Service Parameter

The parameter definition includes:

- Parameter name
- Type
- Required or not
- Validation rule (optional)
- Permitted values (optional)
- JMF Source, a expression used to calculate parameter value from other parameter or external source

There are two categories of parameter: reserved and non-reserved.

A *non-reserved parameter* is the service’s own attribute, for example cm\_mac in the HSD service. Only a provisionable or subtyped service has a non-reserved parameter definition. Subtyped service and its provisionable service have the



same number, name, and type of non-reserved parameters. The only acceptable differences are:

- Validation rule
- Default value
- Format

A *reserved parameter* is required by the SMP architecture and is not related to the service itself. All the services have the following reserved parameters (with reserved flag as true

Table 1: Reserved Parameters

Parameter Name	Purpose
svc_id	Svc_Id in the Sub_Svc table.
sub_svc_id	Sub_Svc_Id in the Sub_Svc table.
parent_sub_svc_id	Parent_Svc_Id in the Sub_Svc table.
sub_svc_ext_key	External_Key in the Sub_Svc table.
provision_status	Sub_Svc_Status_Id in the Sub_Svc table.
purchase_dt	Purchase_Dt in the Sub_Svc table.
start_svc_dt	Start_Dt in the Sub_Svc table.
end_svc_dt	End_Dt on the Sub_Svc table.
created_dtm	Created_Dtm on the Sub_Svc table.
modified_dtm	Modified_Dtm on the Sub_Svc table.
display_id	Exists as a parameter on Sub_Parm table. Used by SPM UI

As a result of service definition, here are the facts about parameters in a subscriber service (known as service instance):

- Resolved parameters, in general, are read-only to Order Manager Client. The Client does not need to populate the these parameters with values.
- A composed type subscriber service does not have non-reserved parameters.
- Order Manager Client does not need to populate all the service parameters defined in the service catalog. For a non-populated parameter, Order Manager will assign it with a default value.
- A subscriber service returned from Subscriber Manager and Order Manager always includes all the parameters defined in the service definition.
- If a non-reserved parameter is required, but is not populated by Order Manager Client and does not have a default value, then the corresponding subscriber service will be treated as invalid and the order will be rejected.

## Service Association

Service Association is used to define the relationship between two service instances. Service catalog defines the following types of service associations:

- Service having children
- Service prerequisite
- Service parameter prerequisite
- Address of Service
- Implementation service-to-service association

*Service having children association*

It is also known as service composition rule. Here is a screen shot of a Service Composition Rule shown in the SCT UI:

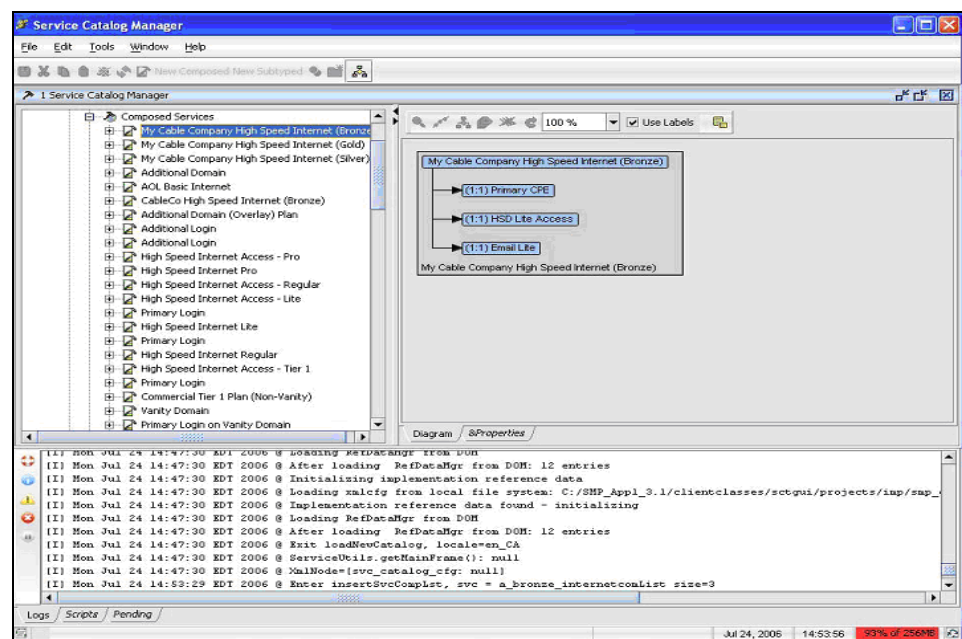


Figure 6: Service Composition Rule

The left pane shows the service definition tree and allows navigating the various service definitions. The right pane displays composition rules for a composed service called “My Cable Company High Speed Internet (Bronze)”. This composed service has three children services:

- Primary CPE (min:1, max:1)
- HSD Lite Access (min:1, max:1)
- Email Lite (min:1, max:1)

*Service prerequisite association*

This association is a service relationship between subtyped or provisionable service to another subtyped or provisionable service. It starts from dependency

service to prerequisite service, which means the instance of prerequisite service must exist before the instance of dependency service can exist.

In most cases, Order Manager Client may not need to populate the prerequisite association. Order Manager will automatically figure it out based on the subscriber profile and dependency rule. If Order Manager cannot figure it out, for example, it finds more than one instance of prerequisite services, it will throw an “illegal data” exception. In this case, Order Manager Client must populate the prerequisite association in the dependency subscriber service.

### Address of Service association

In the SCT service definition panel, Click the Address tab. If “Need Svc Address” is checked, it means this service needs an address.

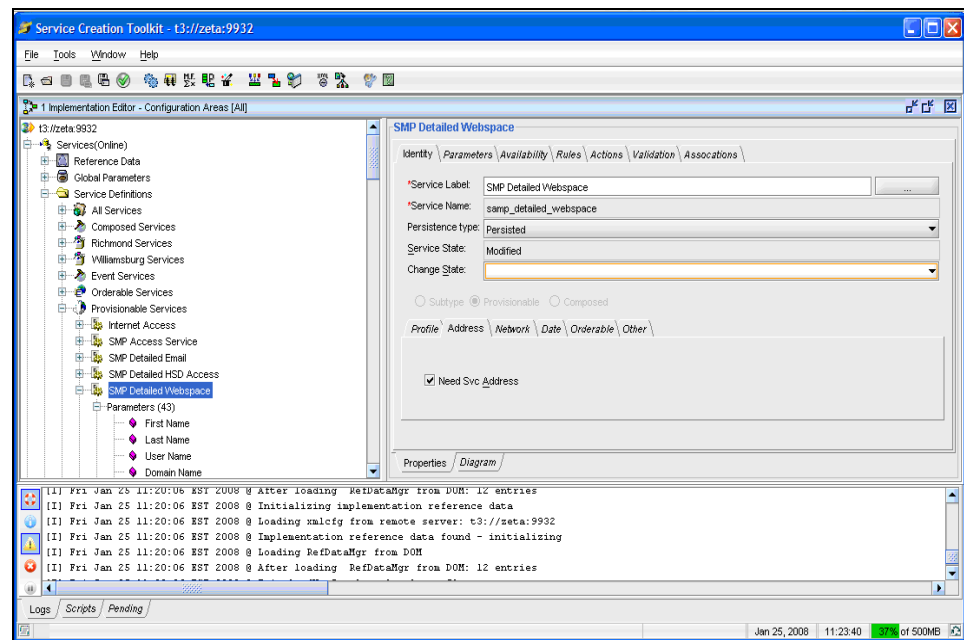


Figure 7: Address of Service Association

When a service needs an address, Order Manager Client needs to populate the “Address of Service Association” value to the subscriber service that it creates.

### Service-to-service association defined in solution

The associations described above are defined by the SMP core. SMP also supports the “solution defined service-to-service” association. Here is the SCT UI screen shot about the association definition:

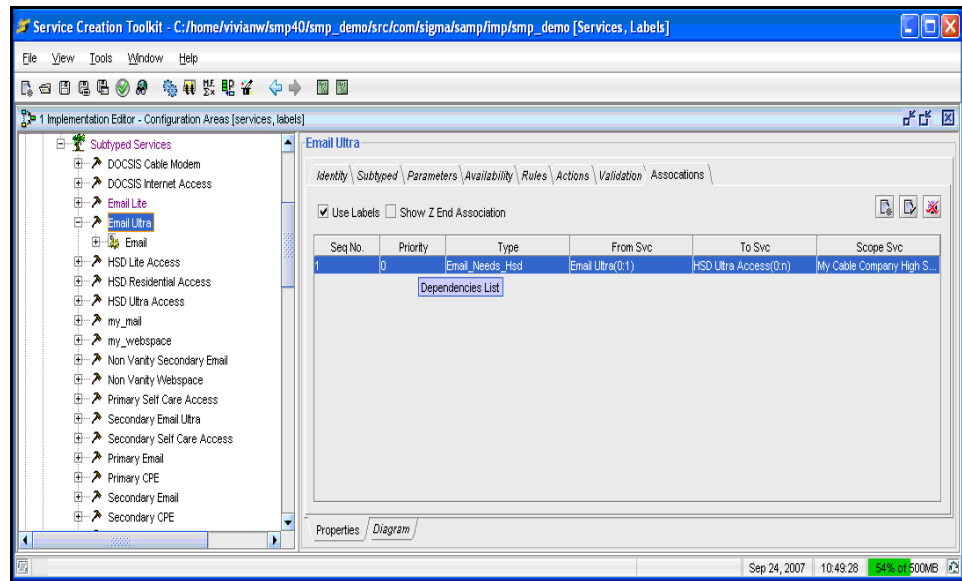


Figure 8: Association Definition

Each association definition includes:

- ZEnd service type (shown as “To Svc”)
- AEnd service type (shown as “From Svc”)
- AEnd cardinality
- ZEnd cardinality

The definition is stored in the service definition xml under the svccat/svc directory. Here is an example in smp\_email.xml taken from SMP DEMO:

```
<cm:assoc_spec_lst>
  <cm:assoc_spec aCard="0-2"
    assocType="Email_Needs_Hsd" captureSeqNum="1"
    name="Email_Needs_Hsd.smp_email.smp_silver_hsd_access"
    scopeClassNm="SubSvcSpec"
    scopeType="cgo_resi_service_plan" zCard="0-n"
    zClassNm="SubSvcSpec"
    zType="smp_silver_hsd_access"/>
</cm:assoc_spec_lst>
```

After defining service-to-service association in Service Catalog, the client can add an association instance in the subscriber service. Association instance is stored in AEnd subscriber service. If the minimum value of ZEnd cardinality equals or is bigger than 1, we say this association is required. For required association, only if the ZEnd cardinality is 1:1, Order Manager tries to figure out the association and populate it into AEnd. Order Manager throws an “Illegal data” exception when the association is not populated by the client and cannot figure out ZEnd from the subscriber profile.

## Subscriber service example

Here is a composed subscriber service example shown in XML format. It includes children subscriber services, service parameters and association between service-to-address and service-to-service.

```

<smpsvc:serviceValue > composed subscriber service
  <smpsvc:serviceKey>
    <cmn:type>SubSvcSpec:cgo_resi_additional_login</cmn:type>
    <smpsvc:externalKey>CP989-cgo_resi_additional_login</smpsvc:externalKey>
  </smpsvc:serviceKey>
  <smpsvc:childServiceList>
    children subscriber services
    <smpsvc:serviceValue> subtyped subscriber service
      <smpsvc:serviceKey>

<cmn:type>SubSvcSpec:smp_str_res_email</cmn:type>
      <smpsvc:externalKey>CP989-smp_str_res_email-1</smpsvc:externalKey>
    </smpsvc:serviceKey>
    <smpsvc:paramList> parameter values
      <smpcbe:param name="password">111111111</smpcbe:param>
      <smpcbe:param name="user_name">aasdfCP989</smpcbe:param>
      <smpcbe:param name="first_name">ASDF</smpcbe:param>
      <smpcbe:param name="domain_type">.ca</smpcbe:param>
    </smpsvc:paramList>
    <smpsvc:associationList>
      <smpcbe:association> implementation service-to-service association

      <smpcbe:associationType>Email_Needs_Hsd</smpcbe:associationType>
      <smpcbe:zEndKey
xsi:type="smpsvc:SubSvcKeyType">
<cmn:type>SubSvcSpec:smp_silver_hsd_access</cmn:type>
      <smpsvc:externalKey>CP989-smp_silver_hsd_access</smpsvc:externalKey>
    </smpcbe:zEndKey>

```

```

        </smpcbe:association>
        <smpcbe:association> address of service
association
        <smpcbe:associationType>service_on_address</
smpcbe:associationType>
        <smpcbe:zEndKey
xsi:type="smpcbe:SubAddressKeyType">
        <cmn:type>SubAddressSpec:-</cmn:type>
        <smpcbe:externalKey>primary</
smpcbe:externalKey>
        </smpcbe:zEndKey>
        </smpcbe:association>
    </smpsvc:associationList>
</smpsvc:serviceValue>
<smpsvc:serviceValue> subtyped subscriber servcie
    <smpsvc:serviceKey>
        <cmn:type>SubSvcSpec:smp_resi_webpace</
cmn:type>
        <smpsvc:externalKey>CP989-smp_resi_webpace-
1</smpsvc:externalKey>
    </smpsvc:serviceKey>
    <smpsvc:paramList> parameter values
        <smpcbe:param name="password" xsi:nil="true"/>
        <smpcbe:param name="web_flag">1</smpcbe:param>
    <smpcbe:param name="first_name" xsi:nil="true"/>
        <smpcbe:param name="domain_type"
xsi:nil="true"/>
    </smpsvc:paramList>
    <smpsvc:associationList>
        <smpcbe:association>
            <smpcbe:associationType>service_on_address</
smpcbe:associationType>
            <smpcbe:zEndKey
xsi:type="smpcbe:SubAddressKeyType">
                <cmn:type>SubAddressSpec:-</cmn:type>
                <smpcbe:externalKey>primary</
smpcbe:externalKey>
            </smpcbe:zEndKey>
            </smpcbe:association>
            <smpcbe:association
xsi:type="smpcbe:ParmAssocType"> service prerequisite
association

```

```

<smpcbe:associationType>service_parm_has_prereq</
smpcbe:associationType>
    <smpcbe:zEndKey
xsi:type="smpsvc:SubSvcKeyType">
    <cmn:type>SubSvcSpec:smp_str_res_email</
cmn:type>
    <smpsvc:externalKey>CP989-
smp_str_res_email-1</smpsvc:externalKey>
    </smpcbe:zEndKey>
    <smpcbe:aParamName>user_name</
smpcbe:aParamName>
    <smpcbe:zParamName>user_name</
smpcbe:zParamName>
    </smpcbe:association>
</smpsvc:associationList>
    </smpsvc:serviceValue>
</smpsvc:childServiceList>
    </smpsvc:serviceValue>

```

## Event Service

SMP supports event type service. It is similar to regular service, has parameter definition, and service association to other event services. The main differences between event and regular service are:

- Event service is not stored in the inventory system  
One example of event service is the VOD service. A VOD subscription only lasts up to one day; the service does not need to be stored in the inventory management system, which would require the creation of a cancel order to remove service in 24 hours.  
SMP assumes that the client contains all the detail information about event services, it simply uses the data and generates impact.
- Event service does not need to belong to a subscriber.  
Event service can represent a network structure. The change is unrelated to the subscriber. In this case, you need to make sure all the required information for impact generation is available from event service itself.

To define an event service, you need to mark the persistence type as “Event”. Here is one example defined in SMP Demo:

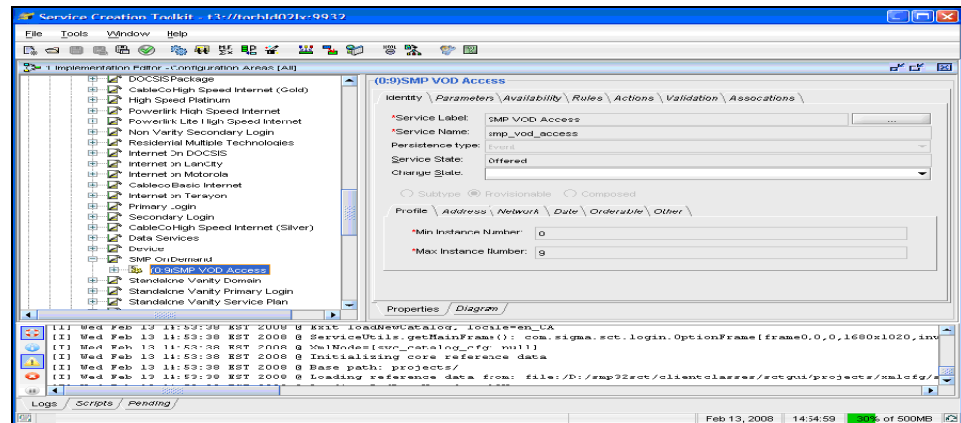


Figure 9: SMP Vode Access

## Entities Definition

### Entity Parameter

Service Catalog also defines parameters for other subscriber entities, such as contact, address and user. Similar to the service parameter definition, the parameter definition includes:

- Parameter name
- Type
- Required or not
- Validation rule (optional)
- Permitted values (optional)
- JMF Source, a expression used to calculate parameter value from other parameter or external source

Here is an SCT screen shot for a subscriber parameter definition:

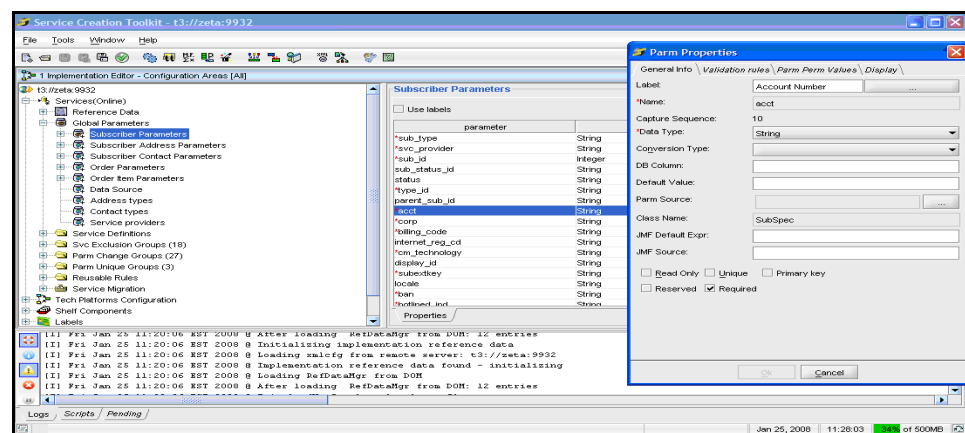


Figure 10: Subscriber Parameter Definition



The client does not need to populate all the non-reserved parameters in the newly created subscriber entity instance. Order Manager will assign an unpopulated parameter with the default value defined in Catalog.

## Entity Association

Service Catalog supports Entity Association definition linked from one entity instance to another entity or service value. This is defined in the XML file called iSubStructure.xml under SMP configuration. So far the SCT UI cannot show or edit the definition. To apply change, you must edit the XML. Here is one example of entity association definition taken from SMP DEMO:

```
<SubAssoc Spec aCard="0-n" aClassNm="SiteSurveySpec"
aType="cable" assocType="SiteSurvey Has Address"
name="SiteSurvey Has Address" zCard="1"
zClassNm="SubAddressSpec" zType="-"/>
```

In a subscriber profile, association value is stored in AEnd entity value. The client needs to explicitly populate the value. Order Manager does not try to figure out the missing association value as it does for service association.

## Core implicit entity Association

The “address of contact” association is an implicit association from the SMP core. This is not defined in SubStructure.xml. The ZEnd cardinality is 1:1. This means each subscriber contact must include this association to link it to a subscriber address. Otherwise, Order Manager will throw an “Illegal Data” exception.

## Subscriber value example

Here is a subscriber value example shown in XML format:

```
<smp:subscriber serviceProvider="Liberate"
subscriberType="residential" locale="en_CA">
  <cmn:lastUpdateVersionNumber>0</cmn:lastUpdateVersionNumber>
  <smp:key xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpCBECORESchema" xsi:type="smp:SubKeyType">
    <cmn:type>SubSpec:-</cmn:type>
    <smp:externalKey>CP991</smp:externalKey>
  </smp:key>
  <smpcbe:state>active</smpcbe:state>
  <smpcbe:paramList>subscriber parameter
    <smpcbe:param xsi:nil="true"
name="billing_subscriber_no"/>
    <smpcbe:param name="hotlined_ind">N</smpcbe:param>
    <smpcbe:param name="acct">888812345</smpcbe:param>
```

```

        <smpcbe:param name="cm_technology">DOCSIS VERSION
1.0</smpcbe:param>
</smpcbe:paramList>
    <smpcbe:entityList>subscriber entities
        <smp:entityValue subscriber address
            xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpCBECoreSchema"
            xsi:type="smp:SubAddressType"
addressType="service">
            <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
            <smp:key xsi:type="smp:SubAddressKeyType">
                <cmn:type>SubAddressSpec:-</cmn:type>
                <smp:externalKey>home</smp:externalKey>
            </smp:key>
            <smp:state>active</smp:state>
            <smp:paramList>subscriber address parameter
                <smp:param name="street_nm">Liberty</smp:param>
                <smp:param
name="serviced_area">BROOKLYN_AMP_1001_12</smp:param>
                <smp:param xsi:nil="true" name="apt_num"/>
                <smp:param name="country">Canada</smp:param>
            <smp:param xsi:nil="true" name="street_dir"/>
                <smp:param name="prov">Ontario</smp:param>
            </smp:paramList>
        </smp:entityValue>
        <smp:entityValue subscriber contact
            xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpCBECoreSchema"
            xsi:type="smp:SubContactType">
            <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
            <smp:key xsi:type="smp:SubContactKeyType">
                <cmn:type>SubContactSpec:-</cmn:type>
                <smp:externalKey>primary</smp:externalKey>
            </smp:key>
            <smp:state>active</smp:state>
            <smp:paramList>subscriber contact parameter
                <smp:param
name="phones.business.extension">2261</smp:param>

```

```

        <smp:param
name="phones.business.number">4169993333</smp:param>
        <smp:param
name="emails.home.address">willys@canada.com</smp:param>
        <smp:param name="suffix">CD</smp:param>
<smp:param name="phones.cell.number">4165554444</
smp:param>
        </smp:paramList>
        <smp:associationList>
<    <smp:association> address of contact association
        <smp:associationType>contact_on_address</
smp:associationType>
        <smp:zEndKey
xsi:type="smp:SubAddressKeyType">
            <cmn:type>SubAddressSpec:-</cmn:type>
            <smp:externalKey>home</smp:externalKey>
        </smp:zEndKey>
        </smp:association>
    </smp:associationList>
</smp:entityValue>
    <smpcbe:entityValue    one orderable subscriber
service
        xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpCBEServiceSchema"
        xsi:type="smp:SubSvcType">
            <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
            <ser1:serviceState xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
provisionState="active"
xsi:type="smp:SubSvcStateType">active</ser1:serviceState>
            <smp:serviceKey>
                <cmn:type>SubSvcSpec:samp_sub</cmn:type>
                <smp:externalKey>CP991svcl</smp:externalKey>
            </smp:serviceKey>
            <smp:paramList> subscriber service parameter
                <smpcbe:param xsi:nil="true"
name="billing_subscriber_no"/>
                <smpcbe:param xsi:nil="true"
name="hotlined_ind"/>
                <smpcbe:param xsi:nil="true" name="acct"/>
<smpcbe:param xsi:nil="true" name="billing_code"/>
                <smpcbe:param xsi:nil="true" name="corp"/>

```

```

    <smpcbe:param xsi:nil="true" name="user_group"/
  >
    </smp:paramList>
    <smp:associationList>
      <smpcbe:association> address of service
association
      <smpcbe:associationType>service_on_address</
smpcbe:associationType>
      <smp:zEndKey xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpCBECORESchema"
xsi:type="smp:SubAddressKeyType">
        <cmn:type>SubAddressSpec:-</cmn:type>
        <smp:externalKey>home</smp:externalKey>
      </smp:zEndKey>
    </smpcbe:association>
  </smp:associationList>
</smpcbe:entityValue>
  <smpcbe:entityValue another orderable subscriber
service
    xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpCBEServiceSchema"
    xsi:type="smp:SubSvcType">
      <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
      <ser1:serviceState xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
provisionState="active"
xsi:type="smp:SubSvcStateType">active</ser1:serviceState>
      <smp:serviceKey>
        <cmn:type>SubSvcSpec:sigma_basic_resi_hsd</
cmn:type>
        <smp:externalKey>CP991svc9</smp:externalKey>
      </smp:serviceKey>
      <smp:associationList>
        <smpcbe:association> address of service
association
        <smpcbe:associationType>service_on_address</
smpcbe:associationType>
        <smp:zEndKey xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpCBECORESchema"
xsi:type="smp:SubAddressKeyType">
          <cmn:type>SubAddressSpec:-</cmn:type>
          <smp:externalKey>home</smp:externalKey>
        </smp:zEndKey>
      </smp:associationList>
    </smpcbe:entityValue>
  </smpcbe:entityValue>
</smpcbe:entityValue>

```

```
        </smpcbe:association>
    </smp:associationList>
    <smp:childServiceList>
</smp:childServiceList>
        </smpcbe:entityValue>
    </smpcbe:entityList>
</smp:subscriber>
```

## *Entity state*

Each type of entity has its own state diagram. All state diagrams start from inactive and end with deleted. The client does not need to populate the entity state in the entity instance value unless it wants to specify a state change.

## Subscriber state

The initial state of subscriber value is inactive. After the “create subscriber” request is successfully handled, the state becomes active. From active, it can move to mso\_block, courtesy\_block or deleted, if there is a state change from the client. When a subscriber parameter gets changed, it becomes change\_in\_progress and moves back to the original state after the corresponding order is completed. Deleted is the final state. After the subscriber is deleted, it cannot be changed to be any other state. Here is the complete state diagram.Subscriber Parameter Definition:

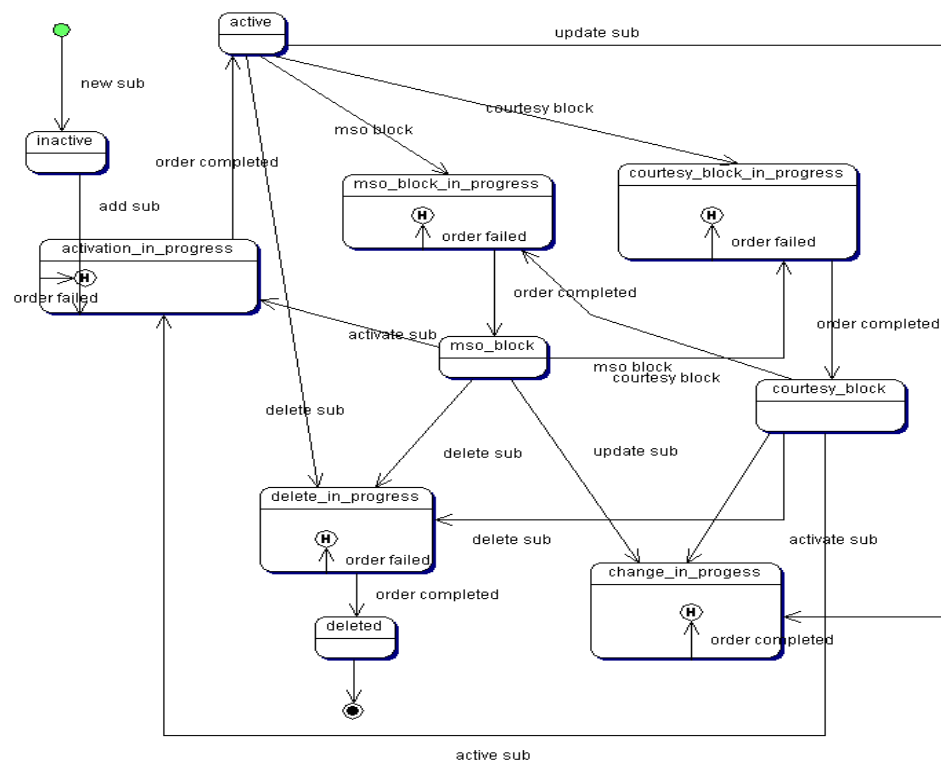


Figure 11: Subscriber State

## Subscriber contact and address state

Subscriber contact and subscriber address have the same state diagram. After they are created successfully, the state is active. When it gets modified, the state becomes **change\_in\_progress** and will move back to active after the corresponding order is completed. Deleted is the final state. After entity value is deleted, it cannot be changed to be any other state. Here is the complete state diagram.

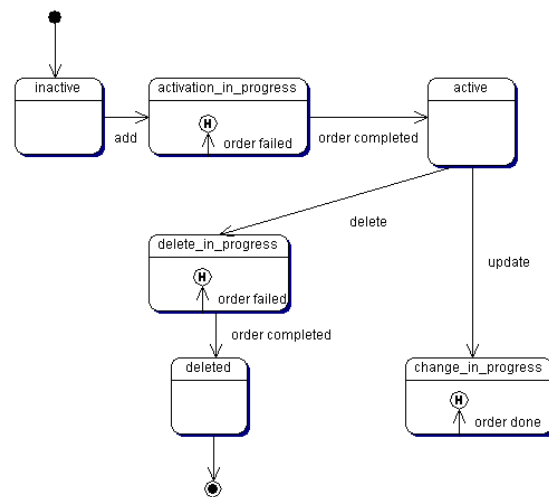


Figure 12: Subscriber Complete State

## Subscriber Service State

Subscriber Service has a state diagram similar to the one for subscriber value. Here are the details.

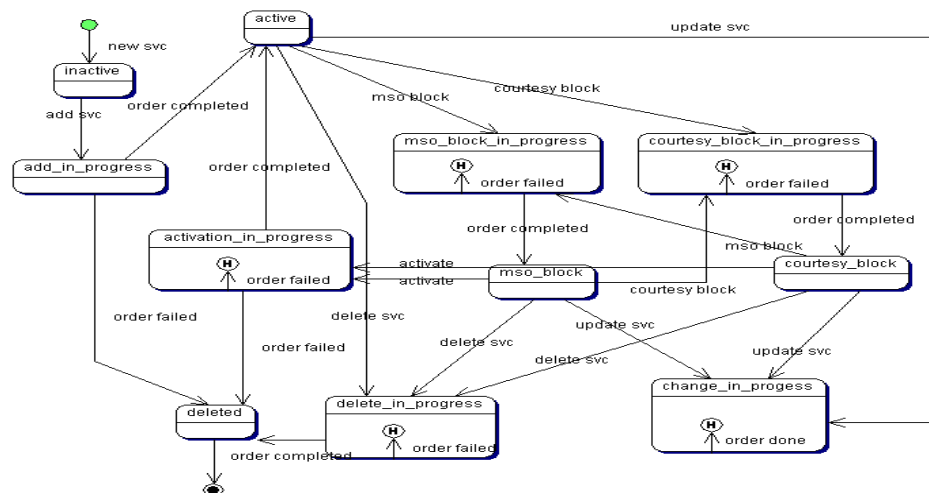


Figure 13: Subscriber Service state





# Order Manager Overview

## Order Manager Architecture

*Order Management* deals with SMP orders; it operates on subscribers, their entities and services. Order Manager provides the following functions on SMP order to its client:

- Accept orders to create subscribers and update subscriber information.
- Accept orders to create, modify, delete subscriber entities (including services).
- Generate workflow requests to reflect changes on external systems
- Query and retrieve orders through OSS/J named queries
- Replicate existing orders
- Repair aborted or partially completed orders
- Manage order life cycle, such as start order, cancel order, and update order.
- Send out notifications for order or line item status changes.
- Send out notifications for errors that happened during workflow processing of order impact.

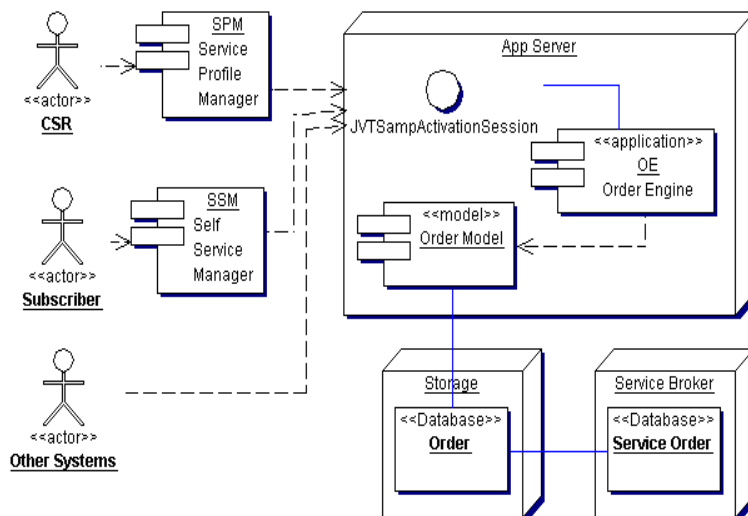


Figure 1: Order Manager Architecture

Order Manager has three types of interface:

- *JVTSampActivationSession*

JVTSampActivationSession is a remote EJB session bean interface. It implements javax.oss.order.JVTActivationSession defined in OSSJ SA (Service Activation) 1.0 specification. It provides all order managers' functions to SMP and external java components that handle Orders.

- *Web Services*

Web Services interface accepts XML request, returns XML response or exception following XSD defined in OSSJ SA 1.0.

- XML Order over JMS Queue

Order Manager also accepts XML request from JMS queue. XML response is sent back to the destination specified in the incoming JMS message.

Notification events are sent from Order Manager to the JMS queue/topic or EJB session bean. An event can be sent as a Java object or in XML format. The format, target, event properties, and notification conditions are defined in ordermgr\_cfg.xml.

## Order models and attributes

---

SMP Order is the top-level object in Order Manager. Each order instance contains:

- Order key
- Order attribute (also known as order level parameter)
- Order state
- Order API Client Id (also known as order source)
- Order priority
- Order completed date time
- Order date (also known as created data time)

Parameter of Order is defined in Service Catalog. Check order\_parm\_defn in SubCfg.xml under xmlcfg/svccat for the detail. For order attributes:

- Client does not need to populate all the order parameters
- Client does not need to populate the order state, key, and read-only parameters. All core-supported order parameters are described in Appendix A
- After receiving the order value, orderManager automatically assigns a default value to an unpopulated parameter

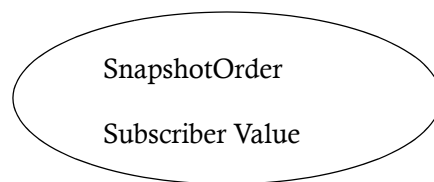
To meet different requirements, there are four types of order supported by Order as

follows:

- Snapshot order
- Action order
- OSSJ order

## Snapshot order

The structure of a snapshot order looks like:



It tells Order Manager to change the subscriber value to the same image included in the order. Order Manager will then:

- If no such subscriber value exists in SMP, persist it to subscriber manager.
- If that subscriber already exists in the inventory, compare incoming profile and the profile from the inventory, figures out the differences and applies the changes.

An example of Snapshot order in XML format:

```

<smp:orderValue xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:SnapshotOrderValue">
  <ser:apiClientId>super_system</ser:apiClientId>
  order source
  <ser:priority>3</ser:priority>
  <ser:orderState order state
smpState="open.not_running.not_started"
xsi:type="smp:SubOrderStateType">open.not_running.n
ot_started</ser:orderState>
  <smp:orderParamList> order parameters
    <smpcmn:param name="queue_request_flag">yes</
smpcmn:param>
    <smpcmn:param name="has_groups">yes</
smpcmn:param>
  </smp:orderParamList>
  <smp:subscriber subscriber snapshot
  
```

```

        serviceProvider="Liberate"
        subscriberType="residential" locale="en_CA">
    </ smp:subscriber>
</smp:orderValue>

```

## Action Order

Action Order contains a collection of LineItems. Each lineItem is a request to manipulate (create/modify/delete) a subscriber entity (including service) as shown below.

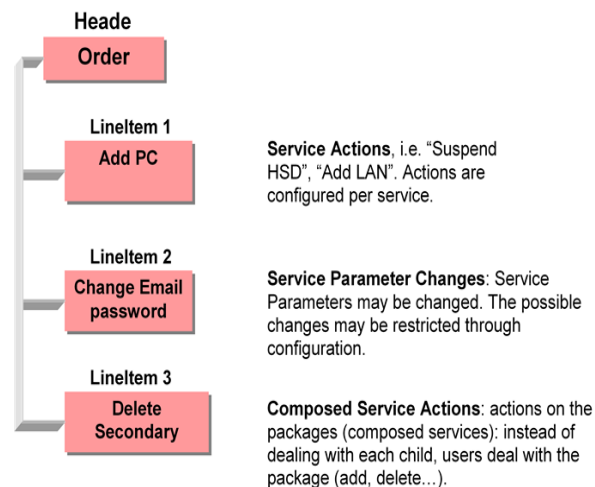


Figure 2: Action Order

Each instance of Action Order contains following attributes on top of standard order structure:

- Subscriber Key (also known as subKey), if this order for subscriber profile changes. Don't need to provide subscriber key if all the line item is for event service not belonging to subscriber.
- List of SampOrderLineItems.

SampOrderLineItem describes change to a subscriber entity.

Each instance contains:

- A lineItemKey, which uniquely identifies this line item
- List of attributes (known as lineItem level parameter)
- Choice of
  - An entity key instance
  - A subscriber service or entity value
- Action (name) against the subscriber entity

Here are important rules about `SampOrderLineItem`:

- Within an order value, each subscriber entity can only have no more than one `LineItem` value
- All the entity key or subscriber entity referred by `SampOrderLineItem(s)` of same order must belong to the same subscriber.
- If entity key and entity value both are populated in a `sampOrderLineItem`, `entityKey` must belong to entity value.

Parameter of `SampOrderLineItem` is defined in Service Catalog. Check `lineitem_parm_defn` in `SubCfg.xml` under `xmlcfg/svccat` for the detail. For `SampOrderLineItem` parameter:

- Client doesn't need to populate all the `LineItem` parameter
- Client doesn't need to populate `LineItem` state, `LineItem` key
- `OrderManager` will automatically assign default value to unpopulated parameter.

An example of action order in XML format is as follows:

```
<smp:orderValue xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:ActionOrderValue">
  <ser:apiClientId>web</ser:apiClientId> order source
  <ser:priority>3</ser:priority>
  <ser:orderState order state
    smpState="open.not_running.not_started"

    xsi:type="smp:SubOrderStateType">open.not_running.n
ot_started</ser:orderState>
  <smp:subKey> subscriber key
    <cmn:type>SubSpec:-</cmn:type>
    <smpcbe:primaryKey>-1</smpcbe:primaryKey>
    <smpcbe:externalKey>CP991</smpcbe:externalKey>
  </smp:subKey>
  <smp:orderParamList> order parameters
    <smpcmn:param name="has_groups">yes</smpcmn:param>
    <smpcmn:param name="queue_request_flag">yes</
smpcmn:param>
  </smp:orderParamList>
  <smp:orderItemList> order LineItems
    <smp:orderItem> order LineItem
      <smp:action>add</smp:action> action name
      <smp:itemParamList> LineItem parameter
      <smpcmn:param name="action_indicator">12</
smpcmn:param>
```

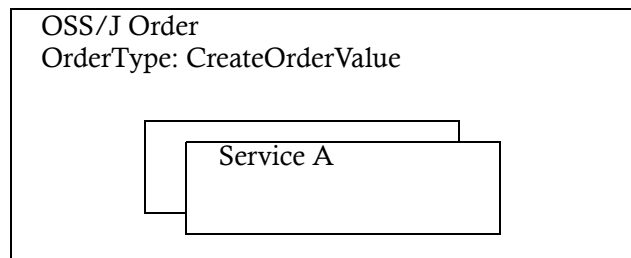
```

        </smp:itemParamList>
        <smpsa:entityValue entity value
xmlns:smp="http://www.sigma-systems.com/schemas/
3.1/SmpCBEServiceSchema"
xsi:type="smp:SubSvcType">
</smpsa:entityValue>
</smp:orderItem>
<smp:orderItemList> order LineItems
<smp:orderItem> order LineItem
<smp:action>delete</smp:action> action name
<smp:itemParamList/> LineItem parameter
<smpsa:entityKey entity key
xmlns:smp=http://www.sigma-systems.com/
schemas/3.1/SmpCBEServiceSchema
xsi:type="smp:SubSvcKeyType">
<cmn:type>SubSvcSpec:samp_ds_hsd_access</
cmn:type>
<smp:externalKey>CP991svc3</smp:externalKey>
</smpsa:entityKey>
</smp:orderItem>
</smp:orderItemList>
</smp:orderValue>

```

## OSSJ Order

OSS/J Service SA 1.0 defines its own order type, known as OSSJ order in this document. Each OSSJ order contains a list of services instance values. Order type attribute represents the action to be applied to those services. For example type `javax.oss.order.CreateOrderValue` means client wants to create all services included in this order.



SMP supports the following OSSJ orders:

- javax.oss.order.CancelOrderValue
- javax.oss.order.ModifyOrderValue
- javax.oss.order.CreateOrderValue
- com.sigma.samp.cmn.order.OssjResumeOrderValue
- com.sigma.samp.cmn.order.OssjSuspendOrderValue
- com.sigma.samp.cmn.order.OssjReprovisionOrderValue

SMP OSSJ Order value need include subscriber key on top of standard order structure and all the services included in the order must belonged to this subscriber.

An example of OSSJ order in XML format is as follows:

```
<smp:orderValue xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:CancelOrderValue"> order type
  <sa:apiClientId>super_system</sa:apiClientId>
  <sa:priority>3</sa:priority>
  <sa:services> included service list
    <sa:item xsi:type="smp:SubSvcType">
      <cmn:lastUpdateVersionNumber>-1</
      cmn:lastUpdateVersionNumber>
      <sa:serviceKey xsi:type="smp:SubSvcKeyType">
        <cmn:type>SubSvcSpec:samp_webpace</cmn:type>
        <smp:externalKey>CP991-sec_webpace</
        smp:externalKey>
      </sa:serviceKey>
      <sa:serviceState>active</sa:serviceState>
      <smp:parentServiceKey>
        <cmn:type>SubSvcSpec:sigma_basic_resi_hsd</
        cmn:type>
        <smp:primaryKey>-1</smp:primaryKey>
        <smp:externalKey>CP991svc9</smp:externalKey>
      </smp:parentServiceKey>
    </sa:item>
  </sa:services>
  <smp:subKey> subscriber key
    <cmn:type>SubSpec:-</cmn:type>
    <smpcbe:primaryKey>-1</smpcbe:primaryKey>
    <smpcbe:externalKey>CP991</smpcbe:externalKey>
  </smp:subKey>
```

```

    <smp:orderParamList> order parameter
      <smpcmn:param name="has_groups">yes</smpcmn:param>
      <smpcmn:param name="queue_request_flag">no</
smpcmn:param>
    </smp:orderParamList>
  </smp:orderValue>

```

## Batch Order

A batch order is a special type of order that is used to group a list of regular orders. The orders that are grouped can be any kind of orders, such as Snapshot order, action order, and OSSJ order, but cannot be the batch type.

A batch order has order parameters, but does not have subkey and subscriber entity.

A batch order is used only in order ownership management. Order ownership is stored as a parameter in each batch order. Ownership management function is provided in the order SMP Web UI.

To use batch order follow these steps:

- Create batch order value; populate parameter roadrunner, which actually is SMP AM user name, for example, samp.csrsl.
- Client submits batch order value to OrderManager by calling createOrderByValue and gets orderKey as returned value.
- Client creates a regular order, for example, action type order; assign batch order id to parameter batch\_ordr\_id.

By linking an action order to a batch order, the owner of the batch order automatically becomes the owner of this action order. Client can change batch\_ordr\_id parameter of action order to move an action order from one batch to another by calling setOrderByValue. Order owner of action order will be changed to the owner of the new batch order.

## Order Type Usage

The following table lists what each order type can be used for:

	Manipulate Subscriber Value	Manipulate subscriber Service	Manipulate Subscriber Entity	Manipulate more than one type of subscriber entity	More than one action	Service Migration
Snapshot Order	Y	Y	Y	Y	Y	Y



OSSJ Order	N	Y	N	N	N	N
Action Order	Y	Y	Y	Y	Y	N



Client need send in whole subscriber profile (of cause need include the changes)

Snapshot orders are generally use for the following tasks:

- Create subscriber value  
A snapshot order includes new subscriber values. A new subscriber may include all its entities and entity associations, such as contacts, addresses, services.
- Service Migration request  
A snapshot order includes migrated subscriber values.
- Delete subscriber  
A snapshot order includes subscriber values with deleted state.
- Modify subscriber (suggest using action order)  
A snapshot order includes modified subscriber values.  
With a snapshot order, you are able to do very complicated profile changes.

It is recommended that you use action type orders in most of the cases. Here is the pattern about how to use an action order to manipulate subscriber, and entity value:

- Update subscriber parameter  
Create a line item with “update” action and modified subscriber value as item entity value. An order accepts change only in the subscriber parameter. If any entity included in the subscriber is also changed, the change will be ignored. Order Manager expects a separate LineItem to explicitly indicate this entity change.
- Update subscriber state  
Create a line item with “update” action and subscriber value with new state as item entity value. State change can be done together with parameter change within the same line item.
- Create service value  
Create a line item with “add” action and new service value as item entity value. To create a composed service and its children service, you only need one line item for the composed subscriber service. Order manager will automatically create children services from this line item.
- Delete service value

Create a line item with “delete” action and service key as item entity key or service value as item entity value. To delete a composed service and its children service, you only need one line item for the composed subscriber service.

- Update service parameter

Create a line item with “update” action and modified service value as item entity value. Make one line item for each service value. For example, to make one change on composed service value and one change on its child service value; you need two line items.

- Update service state

Create a line item with “update” action and service value with new state as item entity value. State change can be done with parameter change within the same line item.

- Create entity value, for example, subscriber contact

Create a line item with “add” action and new entity value as item entity value.

- Delete entity value, for example, subscriber address

Create a line item with “delete” action and entity key as item entity key or entity value as item entity value.

- Update entity parameter, for example, subscriber user

Create a line item with “update” action and modified service value as item entity value.

- Add association

Create a line item with “add” action and association value as item entity value.

Alternatively, add this new association to AEnd entity, create a line item with “update” action, and includes modified AEnd entity as item entity value.

- Delete association

Create a line item with “delete” action and association value as item entity value.

Alternatively, delete this association from AEnd entity, create a line item with “update” action, and includes modified AEnd entity as item entity value.

# Order Life Cycle

---

## Create order

After preparing the order value, there are three ways available from Order Manager to submit an order:

- createOrderByValue then startOrderByKey

These two steps are defined in OSSJ SA. After calling createOrderByValue, order value is accepted and stored in Order Manager with state as open.not\_running.not\_started. Method createOrderByValue returns orderKey back to client.

Client can retrieve, query, update order or cancel order. To ask OrderManager to process this order, client need to explicitly call startOrderByKey.

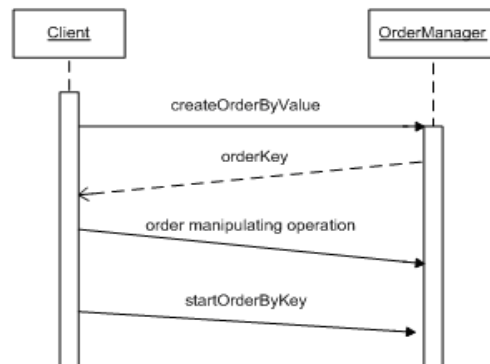


Figure 3: Create Order

- executeOrderAsync

This method is the same as createOrderByValue plus startOrderByKey. Merging two steps reduces the transaction number and remote calling in order to improve overall performance.

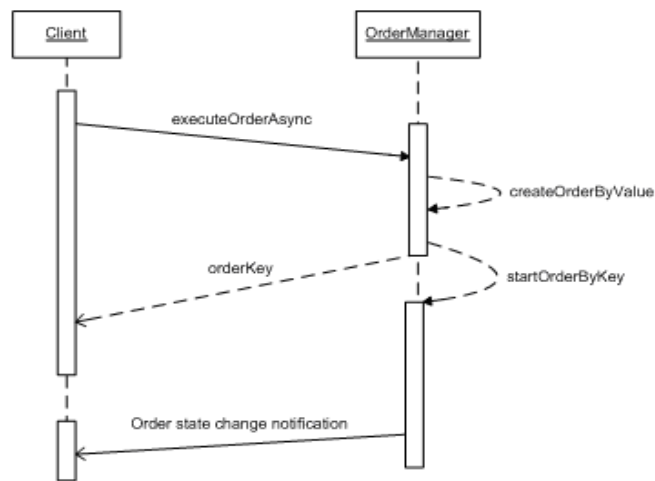


Figure 4: Execute Order Async

- `executeOrder`

Similar to `executeOrderAsync`, it analyzes order, generates workflow request and request to engine before returning `orderKey` back to client.

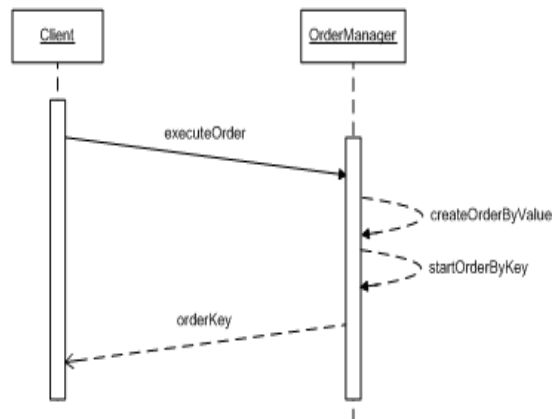


Figure 5: Execute Order

## Process Order

The following diagram presents order processing steps in `startOrderByKeylogic`

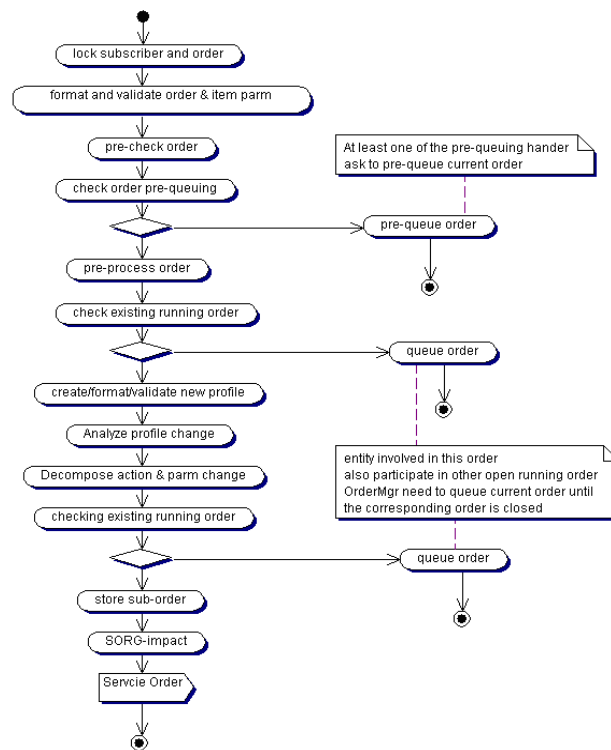


Figure 6: Process Order

After order processing,

- If no workflow request (known as service order) is generated, we say this order doesn't have impact. The profile changes will be applied to the inventory (which is subscriber manager) immediately an order will be marked as "closed.completed.all"
- If a workflow request is generated, the request is sent to the workflow engine. After the engine successfully processes the request, Order Manager will apply profile changes to the inventory and marks the order as "closed.completed.all"
- If the engine cannot process the full request, Order Manager will coordinate other SMP components to roll back changes by this order and mark the order as "closed.aborted.aborted\_byserver"
- <If the engine processes some part of the request, Order Manager applies changes to the line items which have been completed and rolls back update for the line items which have been aborted. The order will be marked as "closed.completed.partially"

The following subsections describe the major steps in the order processing diagram presented on the previous page.

## Lock Subscriber and Order

SMP needs to lock a subscriber to make sure SMP only handles one order at a time for each subscriber, even if it is a multi-node cluster. SMP locks an order to prevent it from getting updated by another client in a different thread.

## Order Pre-queuing

An order needs to be pre-queued since it is not mature/ready for processing. For example, all orders sent from the BSS adapter must be executed in sequence. If No.3 order comes before No.2 order, OrderManager has to pre-queue order No.3. After it processes order No.2, it will go back to execute order No.3.

Every deployment can include a list of “order pre-queuing handler”. A handler makes decision whether:

- the current order needs to be pre-queued
- there is any other pre-queued order that can be triggered after it finished processing the current order?

The following is an example of the session in `ordermgr_cfg.xml` that is used to define the pre-queuing handler.

```
< prequeuing_beans
<bean jndi_name=
    "com.sigma.samp.ordermgr.PreReqOrderHandlerBean"
    check_condition="nzlen order.prereq_ordr_id"
    trigger_condition=
    "substr(order.state,0,20)=='closed.completed.all'"
    queuing_state="prereq"/>
</ prequeuing_beans>
```

All pre-queuing orders have their records in the `sub_ordr` table. All pre-queuing orders have a state with prefix `"open.not_running.pre_queued"`. You can define a specific status after the order is pre-queued by a specific pre-queuing handler. In the above definition sample, the queuing state is “prereq”. This means that when an order is pre-queued by `PreReqOrderHanlderBean`, the state will be `"open.not_running.pre_queued.prereq"`. If no `"queuing_state"` exists in the handler definition, the pre-queued order will be in the default `open.not_running.pre_queued` state.

If Client does not want the order to be pre-queued, it can assign “no” to the **queue\_request\_flag** order parameter. If the pre-queuing handler(s) needs to pre-queue this order, instead of pre-queuing the order, Order Manager throws back `SampIllegalStateException` to Client.

SMP4.0 Core provides the following pre-queuing handlers as listed in Table 1:

Table 1: Pre Queuing Handler

Handler	Description	Required order level parameter(s)
Schedule-order pre-queuing handler	Client submits order and asks OrderManager to execute it at a future time.	<b>start_datetime:</b> String type. Datetime must be specified in the following format: YYYYMMDDhhmmss For example: 20070102081232 for 8am 12:32 in Jan 2 <sup>nd</sup> 2007. This parameter is optional. If it is not provided, this feature will be ignored.
PreReq-order pre-queuing handler	Client can specify a preReq order for the current order. The preReq order must exist in SMP. When preReq order is successfully done, orderManager automatically triggers current order. If preReq order is aborted or partially completed, client can submit a repairing order and when the repairing order is done, ordermgr triggers current order.	<b>prereq_ordr_id:</b> Integer type It is the order id of preReq order. This parameter is optional. If it is not provided, this feature will be ignored.
Hold -rder pre-queuing handler	Client creates order but does not want to execute it until client explicitly releases this order.	<b>hold_ordr_flag:</b> Boolean type If the hold flag is ture, the order will stay in pre-queuing state until this flag value is changed to false. This parameter is optional. If it is not provided, this feature will be ignored.

## Order Pre-checking

There are three types of validation under order pre-checking.

*Order attribute validation*

Before Order Manager starts processing the order, it needs to make sure:

- All required parameters are populated or have default values in the parameter definition
- All parameters belong to the correct types and follow the parameter validation rules.
- `apiClientId`: the attribute value must be defined in `DataRefSrc.xml` under `{implementation configuration dir}/xmlcfg`
- `Priority`: according to OSSJ, the value must be within the range of [1-5]. Priority 5 is highest and 1 is the lowest.

*Rule-based validation*

Order Manager also makes sure that the incoming order conforms to the rules defined in the `order_checking` session of `ordermgr_cfg.xml`. Here is an example:

```
<order_checking>
  <statement expression="jmf_bean.OrderDupChkBean"
    error_info="order external key is not unique!"/>
</order_checking>
```

The expression is written in JMF. The result type of the expression must be Boolean. In the expression, you can refer to the order parameter by using `!order.i` as a prefix.

For more complicated rules, it is better to use a JMF bean. See the *Order Manager Java API Guide* for details.

*Subscriber Entity Version checking*

Each subscriber entity has a version number, which is updated every time the entity instance is modified by an order. Version checking is useful to avoid “stale data” overwriting “recently changed data”.

For Snapshot Order, if the order parameter “`sub_version`” is populated and the value is larger than 0, OrderManager checks to make sure this value is the same as the subscriber version stored in the inventory. If the value is different, a `SmpStaleDataException` will be thrown from Order Manager.

For every subscriber entity in an action order or subscriber service in an OSSJ order, OrderManager extracts the `lastUpdateVersionNumber` attribute from the incoming entity. The value can be either -1 (to bypass version checks) or the current version of this entity loaded from the subscriber manager. Otherwise, the order will be rejected; OrderManager throws back `IllegalArgumentException`.

## Order Pre-Processing

The pre-processing can alter an incoming order just before the order execution. It is a very important feature when the solution heavily uses Snapshot Orders. The basic issue is that when Snapshot Orders are queued, the subscriber model used for the snapshot is incorrect or is not up-to-date. Once this snapshot order with the out of date information is processed, it can “undo” any changes done by queued orders processed ahead of it. If versioning is used, the new order is simply rejected just as it is extracted from the queue, for processing.



You need the ability to have snapshot orders operate with the most up-to-date version of the subscriber model when they begin processing.

The solution has modified the sequence of processing snapshot orders, by introducing the pre-processing bean. The bean is invoked whenever the order is about to be processed (after any possible queuing). Thus, the bean has access to the most up-to-date subscriber profile.

The most frequent scenario is when there is a simple or complex piece of logic to be applied to the subscriber's services. If this applies to the service image at the time a snapshot order is created, then it may either overwrite some other changes (if versioning is bypassed) or get rejected just before execution, if there are versioning conflicts. This piece of logic can be placed inside the pre-processor bean.

Enough information must be passed in with the order (as order parameters for instance) for the bean to know what needs to be done (such as the name of an algorithm to be used, for example, "suspend\_all\_services").

The bean can then process the current subscriber profile, generating a new order which will replace the old one. Given that this is a snapshot order, there are no queuing issues due to changing the order after queuing (all snapshot orders lock the entire profile).

Sample scenarios:

- 1 Subscriber parameter change (*e.g.*, change BAN, change language)
- 2 Suspend subscriber (and all services)
- 3 Hotline subscriber (temporarily redirect all access to a CSR contact)
- 4 Zero Balance (temporarily redirect all access to a CSR contact, when the pre-paid balance reaches 0)

Sample logic in the bean (for hotline):

- 1 Bean checks the order description to determine whether hotline action is occurring.
- 2 Bean replaces snapshot in order with current subscriber model.
- 3 Bean updates subscriber in snapshot to set hotline flag.
- 4 Bean cycles through services in snapshot and sets the appropriate services to "courtesy\_block" (suspended).
- 5 Bean returns order with updated snapshot.

Order Manager also supports pre-processing for action type order or OSSJ order. Pre-processor configuration is stored in `ordermgr-cfg.xml`. Here is an example:

```
<order_pre_processor_cfg>
  <pre_processor order_type="SnapshotOrderValue"

  jndi_name="com.sigma.samp.ordermgr.SamplePreProcessSessionBean"/>
```

```
</order_pre_processor_cfg>
```

## Order queuing

When Order Manager analyzes an order and sends a workflow request, it locks the corresponding entities. The lock(s) will be removed after the order or related line items are closed. Before the lock is removed, Order Manager can process another order for the same subscriber as long as the entities impacted by this order are not locked.

SMP supports two levels locking:

- Subscriber level locking
- Service level locking

The configuration is stored in `ordermgr_cfg.xml`. Here is an example:

```
<policy>
  <rule name="depy_mgr_impl"
    value="com.sigma.samp.ordermgr.depy.SvcLevelLockDepyMgr"/
  >
</policy>
```

The above setting is for service level locking. The rule value for subscriber level locking is `com.sigma.samp.ordermgr.depy.SubLevelLockDepyMgr`.

Those two classes are `DependencyManager` implementations.

`DependencyManager` makes decision about:

- Execute or queue the order.
- After an order is executed successfully (sent to workflow engine), `Dependency Manager` need to figure out executable orders from existing queued orders

### *Subscriber level locking*

If subscriber level locking is selected, one subscriber can only have one open.running order in the whole SMP system, including workflow engine. Any other requests have to wait.

### *Service level locking*

Under service level locking, one subscriber can have more than one open.running orders in the whole SMP system if those orders target different services in different packages. One example is if “order A” updates email password and “order B” updates voice phone number, they can run in parallel. But if “order B” also updates email quota, then “order B” has to wait until “order A” gets closed.

When an order is queued, it has a record in `sub_ordr` table. The status id is 90 and name is `open.not_running.queued`. You can query `sub_ordr` table to looking for queued order. SMP UI also provides search-by-OrderStatus function.

### *Ghost Services*

Consider a “create service in one order, and then update it from GUI” case. The respective service would only be visible in the GUI or available for the BSS adapter to modify only when the “create service” order actually executes.

If order is queued, how can client change the service parameter?

To get around this problem, when creating service order is queued, Order Manager will create a “ghost service”, which is an empty service with an inactive status, to match the service that *will be created*.

The service is then visible in the GUI and available for BSS adapters to modify, even though it is not actually created. A BSS adapter can even submit an order to delete the service while the “create” is still queued. What happens is that they will go through in sequence (because of order queuing logic) and the service will first be activated and then deactivated (deleted).

This feature works for action type order and OSSJ order, but not on the Snapshot order.

## New profile construction and validation

Order Manager has an old profile loaded from the subscriber manager. To make a new profile, Order Manager needs to

- Apply changes from order request to profile loaded from subscriber manager
- Re-calculate parameters with JMF Source definition
- Figure out missing service-to-service association.

In new profile validation, Order Manager checks that:

- Instance number meets minimum requirement and does not exceed maximum limitation.
- Does not have missing association.
- All associations in profile has corresponding rule definition
- All associations AEnd and ZEnd cardinality are correct.
- All entity parameter passes validation rule checking.
- Service parameter also needs passing unique checking

### *Service parameter unique checking*

Order manager does parameter uniqueness checking on subscriber service to make sure the parameter value is unique among all subscribers. The definition is in service catalog on service parameter configuration. Here is a snapshot of SCT UI. Since parameter “cm\_mac” in service “hsd access” has “unique” option checked, Order Manager need do unique checking for this parameter.

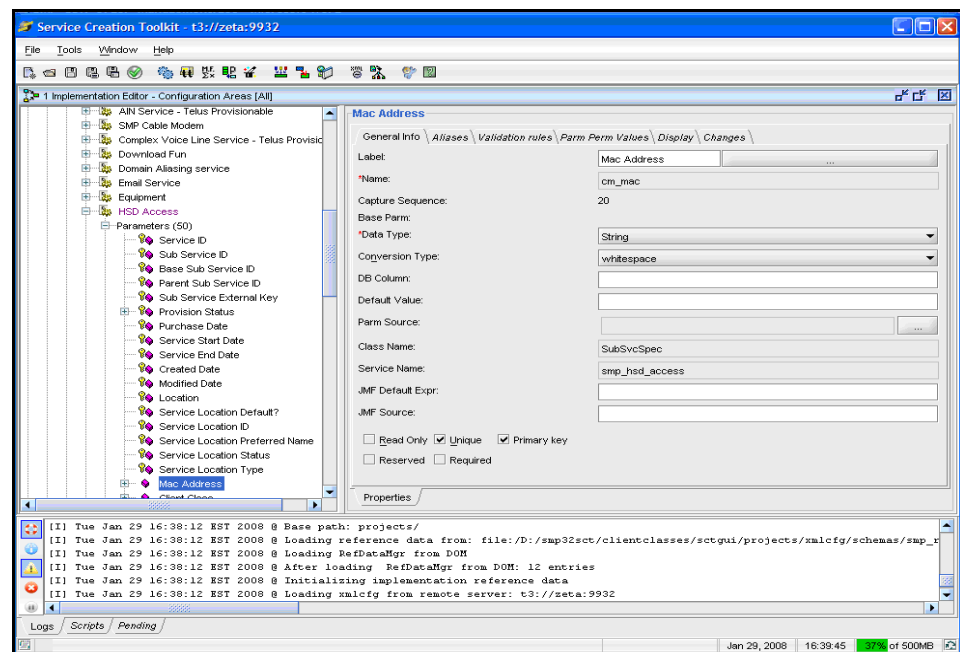


Figure 7: ServiceParmUniqueChecking

Order Manager rejects the order (by throwing back `IllegalArgumentException`) if any service parameter value included in the order breaks the unique constrain.

### Cool Off Period

Some unique value parameters/services require a “cool off period”. That is, even when the service is deleted, the unique parameters on this service will not be taken away. No other new services can reuse the parameter values before the cool off period expires.

A good example is email: when a user deletes his/her email account, it will be “cooling off” for a configurable period of time (30 days) and no other user can use the same account. After the 30 days, any other user can use it.

This works usually in conjunction with external systems, such as email servers that offer some “cool off” functionality as above.

To enable the cool off feature, service definition must define a parameter called `icool_off_period` and define a default value (in number of days) for it. After a subscriber service is deleted, if there is a “cool\_off\_period” defined in service definition, Order Manager will calculate the end date and populate the `endDateTime` attribute with this value.

When Order Manager does service parameter unique checking, it needs to count parameter from deleted service with undue “endDatetime”.

### Impact

After validating the request, Order Manager generates a workflow request (also known as service order or impact) based on the impact definition. A workflow request is a BEPL-alike script. Here is an example:

```

<process xmlns="http://www.sigma-systems.com/sb/jwf_pracs"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sigma-systems.com/sb/
jwf_pracs
../../../../samp/xmlcfg/schemas/jwf-process.xsd"
name="SvcOrderRrq-10001" is_smp_order="true"
smp_version="4.0" syntax_id="0"
effective_dtm="20080212155721" version="1">
  <scope ignoreLateStatus="false" syntax_id="1">
    <variables>
      <variable name="has_groups" type="string"
syntax_id="2">
        <initExpr isConst="true" constType="string">yes</
initExpr>
      </variable>
      <variable name="priority" type="int" syntax_id="3">
        <initExpr isConst="true" constType="int">3</
initExpr>
      </variable>
      <variable name="spm_ordr_id" type="long"
syntax_id="4">
        <initExpr isConst="true" constType="long">10001</
initExpr>
      </variable>
    </variables>
    <flow syntax_id="5">
      <links>
        <link name="fms_add_sub_SID_3_fms_add_cm_SID_2"
syntax_id="6"/>
        <link name
="billing_activate_cm_SID_1_fms_add_cpe_SID_6"
syntax_id="7"/>
      </links>
      <activities>
        <lineItemGroup name="group9910" syntax_id="8">
          <variables>
            <variable name="entity.entity_type" type="string"
syntax_id="9">
              <initExpr isConst="true"
constType="string">SubSvcSpec:samp_sub</initExpr>
            </variable>
            Ö
          </variables>
        </lineItemGroup>
      </activities>
    </flow>
  </scope>
</process>

```

```

        <links>
            <link name
="fms_add_sub_SID_3_fms_add_prod_SID_4" syntax_id="12"/>
        </links>
    </flow>
</activities>
</flow>
</process>

```

The workflow request is then sent to the workflow and the order moves to the `open.running.workflow.processing` status.

#### *Ignores order impact*

In some circumstances, for example, a subscriber migrates from a non-SMP system to SMP, client only wants to create/update the profile and ignore the impact (which means, it does not want to generate a workflow request), because the services have all been provisioned.

SMP order has an order parameter `impact_generation_policy`. In order to ask SMP to ignore order impact, client needs to populate this parameter and assigns it the value “skip\_tmplt”.

#### *Groups workflow request*

Order client can use order parameter “`has_groups`” to control how a workflow request is to be executed in the engine.

- If the value is “no”, request will be processed as one unit. This means the whole order is either fully completed or fully aborted. But if a task in the workflow request needs 10 days to process, all the entities impacted by this order will be locked 10 days; any changes to those entities will be queued.
- If the value is “yes”, the engine sends OrderManager notification when each LineItem is completed or aborted. Order Manager closes the line item after it gets the notification and also releases the corresponding entity lock. But by doing that, an SMP order may have some line items completed and the remaining items aborted. If this happens, the order will end up in `closed.completed.partially` state.

The default value for “`has_groups`” parameter is “no”. If “`has_groups`” is not populated in the SMP order value, Order Manager will execute the order, and perform order-level commit or rollback as one unit.

## *Update Order*

An order can be updated when it is in “`open.not_running`” state. There is a method “`setOrderByValue()`” if you use `JVTSampActivationSession` interface, or use `setOrderByValue` request when using XML API (including web services).

To update an order parameter for better performance, you can use “`updateOrderParm()`” if you choose the `JVTSampActivationSession` interface, or use `OrderParamChangeValue` inside `setOrderByValue` request when selecting XML API.

The major steps involved in order updating are shown below:

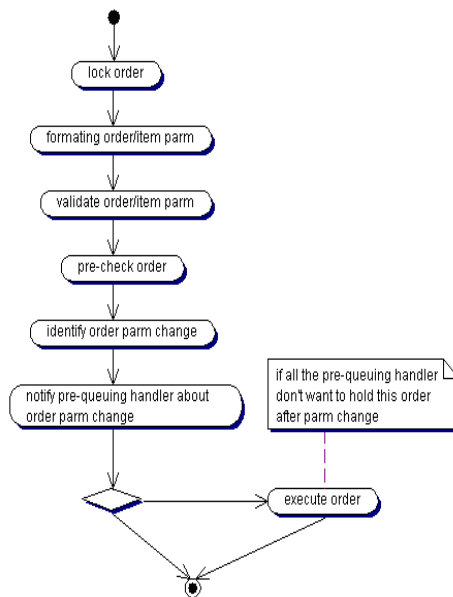


Figure 8: Update Order

Before updating an order, Order Manager needs to lock the order to prevent the order from being processed or updated in another thread at the same time.

After the order is updated, Order Manager performs another pre-queuing check, and executes the order if it passes the pre-queuing check.

Here is an example of why does Order Manager need this checking:

Order B is initially waiting for order C but C is not submitted to SMP yet. So Order B stays in the pre-queued state. Now order B is updated to depend on order A and A has already been successfully executed. SMP needs to execute order B after the dependency is changed.

## Cancel Order

Client can cancel the whole order when the order is in the “open.not\_running” state. An aborted order will be in the state of `iclosed.aborted.aborted_byclient`.

If an order is in “open.running.workflow.processing”, only open.running line item will be cancelled. If an order has a combination of cancelled and completed items, it will be in the state of “closed.completed.partially”. If it is fully cancelled, it will be in the state of “closed.aborted.aborted\_byserver” (not the `abort_byclient` state, since the cancel response is sent from the workflow engine).

To cancel an order from `JVTSampActivationSession` interface, you can use the method `cancelOrderByKey()`, from XML API, use `cancelOrderByKey` request.

## Repair Order

For an aborted or partially completed order, Order Manager now is able to create a repairing order for it. The repairing order contains all the line items which are aborted.

The repairing order carries all order parameters in the original order. It adds one extra order parameter “original\_order\_id”, which is the order id of the problem order.

This new feature is provided by a new method buildRepairingOrder() in OrderMgr’s JVTsmpActivationSession, or repairOrderByKey request in XML API.

The Client repairs an order in the following steps:

- Client finds an aborted or partially completed order
- Client calls buildRepairingOrder() or sends in repairOrderByKey request if using XML API
- Order Manager returns a new order (known as “repairing order”)
- Client reviews and adjust the order (for example, update mac address)
- Client submits the repairing order to Order Manager

After Order Manager receives the “repairing order”, it automatically marks the original order as “resolved”. After the repairing order completes successfully, any orders that are dependent on it or the original order will be triggered and executed.

For example, order B depends on order A. Order C is created to repair order A. After order C is completed executed, OrderManager will trigger order A.

Here is the sequence diagram about order repairing:

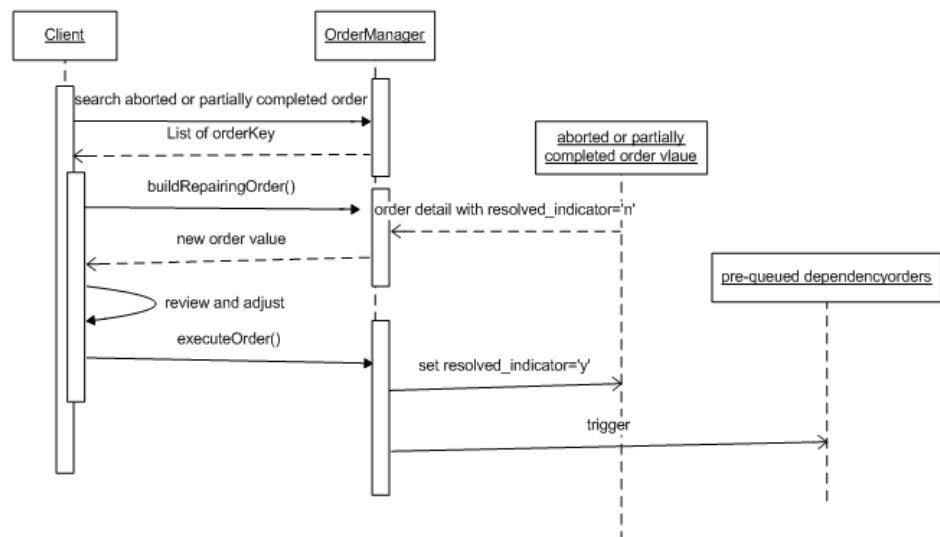


Figure 9: Repair Order



The “repairing order” takes all information from the original request. If the parameter is not correct, this function cannot provide the fix. The user of this feature needs to review the returned order from this function, update it, before submitting it to OrderManager.

Here are the new Order Header Parameters used in this feature:

- original\_order\_id

The “repairing order” includes this order parameter to link to the original order.

Client should not update this parameter change. Order Manager does not accept change on this parameter.

- resolved\_indicator

When an order becomes closed.aborted or closed.completed.partially, OrderManager insert this parameter to order and assign ‘n’ as value. When OrderManager accepts the repairing order, it automatically changes this parameter in the original order to ‘y’.

User can change this indicator from ‘n’ to ‘y’ by doing order header parameter change. But by doing that, there isn’t any new order coming in to fix the problematic order. Any dependency order will stay in “pre-queuing” status forever, unless CSR or BSS adapter call abortOrderByKey to drop the request or remove the order dependency.

## Replicate Order

Replicate Order is to clone an existing active type order. The new order belongs to the same subscriber and is almost the same as the original order:

- Same order type
- Include all order parameters available in the original order
- Similar line items

Here are the differences between the original and cloned order:

- All new added entity values included in the order have new and unique external key.
- New order itself also has new and unique external key

This feature is useful for a situation like:

- 1 Create a phone package for a business subscriber
- 2 Create another phone package for the same subscriber when most of the information is the same as the previous phone package

Client only needs to replicate previous order, update some parameter (for example, phone number and port number) and submit the order.

To replicate an order from JVTsmpActivationSession interface, you can use method replicateOrder(); from XML API, use replicateOrderByKey request.

## Query Order

Order query is based on the query definition stored in `imp/{implementation name}/xmlcfg/named_query_cfg.xml` under element `<query_type nm="order">`. Each query definition includes a query name, sql statement or queryBean JNDI name, supported conditions (known as `searchable_parms`), and possible result set (known as `viewable_parms`).

Here is a definition example:

```
<named_query capture_seq_num="1" chunk_rows_no="150"
nm="OrderQueryByAcct" view_rows_no="15">
  <select_statement><![CDATA[select sub_id,
sub_ordr_id, created_by, ordr_status, acct_no,
created_dtm, modified_dtm from V_ORDR_NQ_BY_ACCT #order by
sub_ordr_id desc]]></select_statement>
  <searchable_parms>
    <cm:parm_lst>
      <cm:parm capture_seq_num="1"
class_nm="Query_Searchable_Parm" data_type_nm="String"
db_col_nm="acct_no" is_pk="false" is_read_only="false"
is_req="true" is_reserved="false" is_unique="false"
parm_nm="acct_no"/>
    </cm:parm_lst>
  </searchable_parms>
  <viewable_parms>
    <cm:parm_lst>
      <cm:parm capture_seq_num="1"
class_nm="Query_Viewable_Parm" data_type_nm="Long"
db_col_nm="sub_ordr_id" is_pk="false" is_read_only="true"
is_req="true" is_reserved="false" is_unique="false"
parm_nm="managedEntityKey"/>
      <cm:parm capture_seq_num="2"
class_nm="Query_Viewable_Parm" data_type_nm="String"
db_col_nm="ordr_status" is_pk="false" is_read_only="true"
is_req="true" is_reserved="false" is_unique="false"
parm_nm="state"/>
    </cm:parm_lst>
  </viewable_parms>
</named_query>
```

To submit a query request, client needs to send a `QueryValue`, which includes:

- The query name
- All condition parameters, or a subset of them, with values
- Expected parameter in result set

A query may return thousands of orders. Normally a user is only interested in the first 50 or even less. There is a set of configuration in `ordermgr_cfg.xml` that could control the query behavior. Here is an example, please check order manage configuration for details.

```

<query_cfg>
  <order_query>
    <property name="maxFetchSize" value="50"/>
    <property name="maxRowSize" value="50"/>
    <property name="defaultImplClass" value="
com.sigma.samp.ordermgr.ejb.QueryImpl"/>
  </order_query>
</query_cfg>

```

## Order States

SMP order at any given time can be in one of the following states:

- NOT\_RUNNING open.not\_running
- NOT\_STARTED open.not\_running.not\_started
- PRE\_QUEUED open.not\_running.pre\_queued
- QUEUED open.not\_running.queued
- WORKFLOW open.running.workflow
- ZOMBIE open.running.zombie
- PROCESSING open.running.workflow.processing
- FAILED open.running.workflow.failed
- WARNING open.running.workflow.warning
- CLOSEED closed
- INVALID closed.aborted.invalid
- ABORTED closed.aborted
- COMPLETED closed.completed
- ABORTED\_BYCLIENT closed.aborted.aborted\_byclient
- ABORTED\_BYSERVER closed.aborted.aborted\_byserver
- ALL\_COMPLETED closed.completed.all
- PARTIALLY\_COMPLETED closed.completed.partially

The following diagram depicts transitions between states:

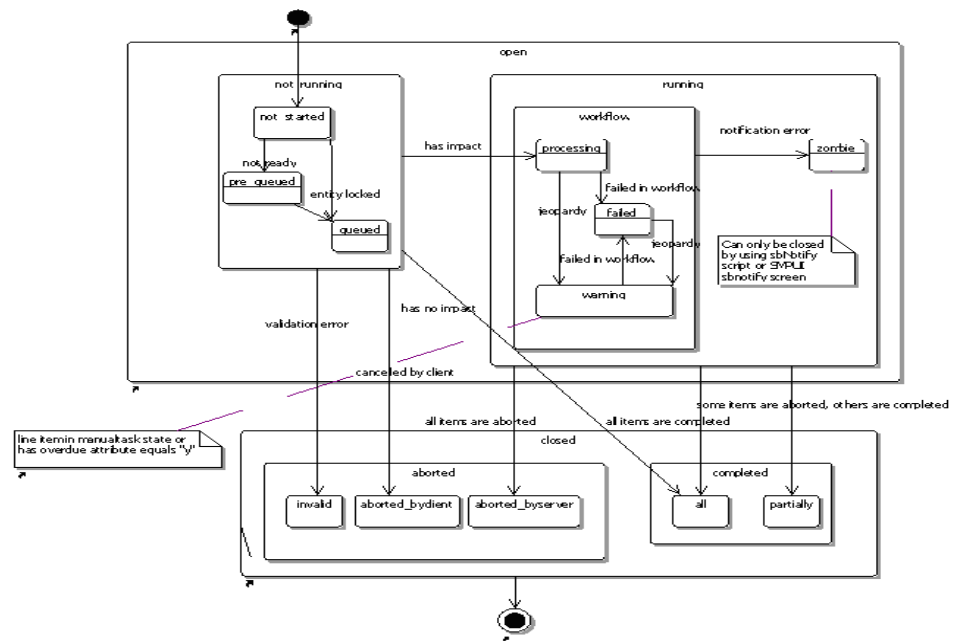


Figure 10: Order States Transition

## State NOT\_RUNNING

NOT\_RUNNING is a parent state. It has three children states:

- NOT\_STARTED
- PRE\_QUEUED
- QUEUED

Order in **NOT\_RUNNING** state can move to:

- INVALID state, when failed in order validation.
- ALL\_COMPLETED state, if there is no workflow request generated; SMP can close order right away.
- PROCESSING state; SMP needs to send workflow request to the engine and wait for a reply.

## State NOT\_STARTED

**NOT\_STARTED** is the initial state of an order when it is created by calling `JVTActivationSession.makeOrderValue(String type)`.

Order in **NOT\_STARTED** state can move to:

- PRE\_QUEUED, when any of Pre-Queuing handlers decides this order is not ready to be executed, for example, schedule time is not due.
- QUEUED, when any related entity is in pending state (for example a subSvc is in `add_in_progress` state).
- Destinations defined in NOT\_RUNNING state.

## State PRE\_QUEUED

**PRE\_QUEUED** means current order does not meet at least one of Pre-Queuing handlers' requirement, for example schedule time is not due. After all the Pre-Queuing handlers agree to let this order go, SMP will execute this order ASAP.

After analyzing the order, Order in **PRE\_QUEUED** state moves to:

- **QUEUED** state, if any related entity is in pending state.
- Destination defined in **NOT\_RUNNING** state.

## State QUEUED

Order stays in **QUEUED** state when any related entity is in the pending state. This also means the corresponding entity is involved in another unfinished order.

After those pending orders are closed, SMP will trigger and execute **QUEUED** order **ASAP**. Order in **PRE\_QUEUED** state can move to:

- Destinations defined in **NOT\_RUNNING** state.

## STATE RUNNING

**RUNNING** is a parent state, it has two children states:

- **WORKFLOW**
- **ZOMBIE**

Order in **RUNNING** state can move to:

- **ABORTED\_BYSERVER**, when all line items are aborted
- **ALL\_COMPLETED**, when all line items are completed
- **PARTIALLY\_COMPLETED**, when at least one of the line items is aborted and reset of them are completed.

## State WORKFLOW

**NOT\_RUNNING** is a parent state. It has three children states:

- **PROCESSING**
- **WARNING**
- **FAILED**

Order in **WORKFLOW** state can move to:

- **ZOMBIE** state, when SMP receiving "order done" reply from engine but can't finish the state transition.
- Destinations defined in **RUNNING** state.

## State PROCESSING

**PROCESSING** means an order is sending a workflow request to the workflow engine and is waiting for a reply.

Order in **PROCESSING** state can move to:

- **WARNING** state, if any of its line items is in an un-healthy state or receives “jeopardy” notification from the engine
- **FAILED** state, when SMP receiving “order failed” notification from the engine.
- Destinations defined in **WORKFLOW** state.

## State WARNING

This state indicates at least one line item is in an un-healthy state (for example MANUALTASK) or got “jeopardy” notification from the engine.

Order in **WARNING** state can move to:

- **FAILED** state, when SMP receives “order failed” notification from the engine
- Destinations defined in **WORKFLOW** state

## State FAILED

Workflow engine cannot continue processing the request. The workflow request is in an unknown state; it needs someone to check the workflow engine and manually rollback/complete the request. Based on what he does, use sbnotify script or SMPUI sbnotify screen to complete or abort the order.

Order in **FAILED** state can move to:

- Destinations defined in **WORKFLOW** state.

## State ZOMBIE

**ZOMBIE** state means SMP receives an “order done” reply from the engine but cannot finish the state transition. This typically means SMP got an exception in doing the state transition or state change notification. It needs someone to resolve the cause of the exception and then use sbnotify script or SMPUI sbnotify screen to complete or abort the request.

Order is in **ZOMBIE** state can move to:

- Destinations defined in **RUNNING** state.

## State CLOSE

**CLOSE** is a parent and final state. It has two children states:

- **ABORTED**
- **COMPLETED**

## State ABORTED

ABORTED is a parent state. It has three children states:

- **INVALID**
- **ABORTED\_BYSERVER**
- **ABORTED\_BYCLIENT**

## State INVALID

This means an order cannot pass validation.

Check order notification or order's `err_reason` attribute for the cause.

## State ABORTED\_BYSERVER

This state indicates that all the order items are aborted by SMP server, workflow engine or adapter. Check order line item notification or line item's `err_reason` attribute for the cause.

## State ABORTED\_CLIENT

This state indicates that an order has been aborted by Order Manager Client by:

- Using method `JVTActivationSession.abortOrderByKey(OrderKey key)`
- Using `abortOrderByKey` XML request
- Deleting subscriber and cause SMP automatically abort all the queued or pre-queued orders.

## State COMPLETED

**COMPLETED** is a parent state. It has three children states:

- **ALL\_COMPLETED**
- **PARTIALLY\_COMPLETED**

## State ALL\_COMPLETED

This means all the order items are completed successfully.

## State PARTIALLY\_COMPLETED

This means some order items are aborted by SMP server, workflow engine or adapter, but at least one of them is completed successfully.

For the aborted line item, check order line item notification or line item's `err_reason` attribute for the cause.

## Order Item States

SMP order Lineitem at any given time can be in one of the following states:

- NOT\_STARTED open.not\_running.not\_started
- WORKFLOW open.running.workflow
- FAILED open.running.workflow.failed
- PROCESSING open.running.workflow.processing
- MANUALTASK open.running.workflow.manualtask
- ABORTED closed.aborted
- ABORTED\_BYSERVER closed.aborted.aborted\_byserver
- ABORTED\_BYCLIENT closed.aborted.aborted\_byclient
- INVALID closed.aborted.invalid
- ABORTED\_ENFORCED closed.aborted.enforced
- COMPLETED closed.completed
- ALL\_COMPLETED closed.completed.all
- COMPLETED\_ENFORCED closed.completed.enforced

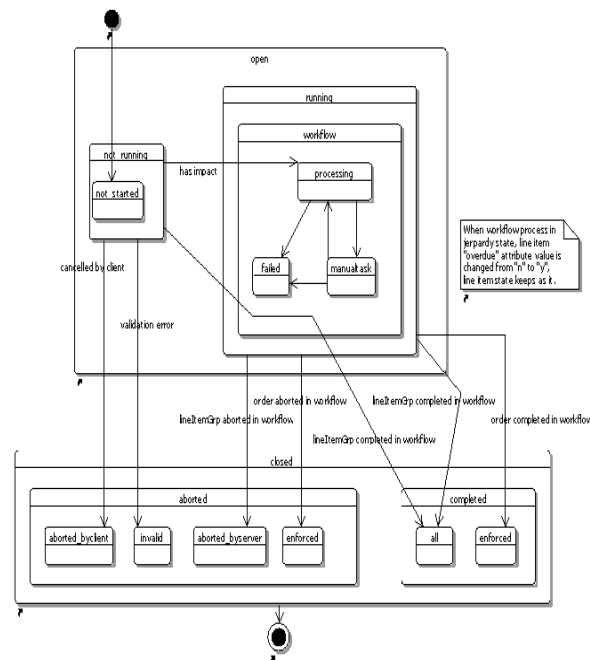


Figure 11: Order Item States Transition

### State not\_started

**NOT\_STARTED** is the initial state of LineItem when it is created by calling `JVTActivationSession.makeLineItemValue()`.



LineItem in **NOT\_STARTED** state can move to:

- **ABORTED\_BYCLIENT** when order is aborted by client
- **INVALID** when order cannot pass the validation
- **ALL\_COMPLETED** when:
  - There is not impact generated from this item
  - And Order “has\_groups” attribute equals ‘y’
  - And having following configuration in ordermgr\_cfg.xml

```
<state_cfg>
  <item_level>
    <line_item_with_noimpact
      action="completed"/>
    </item_level>
  </state_cfg>
```

## State WORKFLOW

This is parent state. It has three children state:

- **PROCESSING**
- **MANUALTASK**
- **FAILED**

LineItem in **WORKFLOW** state can move to:

- **ABORTED\_BYSERVER** after receiving workflow engine’s “line item aborted” notification.
- **ABORTED\_ENFORCED** when order is aborted
- **ALL\_COMPLETED** after receiving workflow engine’s “line item done” notification.
- **COMPLETED\_ENFORCED** when order is completed.

## State PROCESSING

This means LineItem is being processed in the workflow engine.

LineItem in **PROCESSING** state can move to:

- **MANUALTASK** after receiving workflow engine’s “enter mt” notification.
- **FAILED** after order is in **FAILED** state
- Destinations defined in **WORKFLOW** state.

## State MANUALTASK

This means the workflow process related to this LineItem is waiting for human response.

LineItem in **MANUALTASK** state can move to:

- **PROCESSING** after receiving workflow engine's "exit mt" notification.
- **FAILED** after order is in **FAILED** state
- Destinations defined in **WORKFLOW** state.

## State FAILED

This means order is in **FAILED** state, this LineItem target state is known. LineItem in **FAILED** state can move to:

- **ABORTED\_ENFORCED** after order is aborted.
- **COMPLETED\_ENFORCED** after order is done.

## State Closed

This is the parent and final state. It has two children state:

- **ABORTED**
- **COMPLETED**

## State ABORTED

This is a parent state. It has three children state:

- **ABORTED\_BYSERVER**
- **ABORTED\_ENFORCED**
- **INVALID**
- **ABORTED\_BYCLIENT**

## State ABORTED\_BYCLIENT

This means the order has been aborted by client.

## State ABORTED\_BYSERVER

SMP receives "item aborted" notification from the workflow engine.

## State Invalid

This means the order cannot pass validation, for example, missing required service parameter.

## State ABORTED\_ENFORCED

SMP gets the "order aborted" engine reply before receiving "item aborted" notification for this Lineitem.

## State COMPLETED

This is a parent state. It has two children state:

- **ALL\_COMPLETED**
- **COMPLETED\_ENFORCED**

## State ALL\_COMPLETED

SMP receives “item done” notification from the workflow engine.

## State COMPLETED\_ENFORCED

SMP gets the “order done” engine reply before receiving “item done” notification for this LineItem.

# Tools

---

SMP provides some tools for viewing and manipulating an order.

## SMP Web

SMP Web interprets definitions in Service Catalog, automatically generates forms to create subscribers and services. It allows users to specify order parameters, can generate and submit the orders. It also supports order query, subscriber query, order browsing and order update.

See the *Service Profile Manager User Guide* for details.

## XmlApiUtil.sh

This script is located under \$DOMAIN\_HOME/tools/com.sigma.samp.ordermgr. It can generate XML request like:

- executeOrder
- executeOrderAsync
- createOrderByValue
- startOrderByKey
- abortOrderByKey

It takes jndi.properties from the \$DOMAIN\_HOME/test directory. Make sure the URL points to one of the SMP node server.

## *SbNotfiy.sh*

This script is located under `$DOMAIN_HOME/tools/com.sigma.samp.ordermgr`. It is used to simulate the workflow engine by generating responses back to Order Manager.

For example,

```
$. /SbNotify.sh 10000 SUCCESS
```

This tells Order Manager “order 10000” has been successfully executed in the workflow engine.

This script can also be used to generate order parameter change, line item status change. You can execute the following command to get the detail usage:

```
$. /SbNotify.sh -?
```

It takes `jndi.properties` from `$DOMAIN_HOME/test` directory. Make sure the URL points to one of the SMP node server.

## *Dump XML Order Request to XML File*

Order Manager can dump all receiving orders to an XML file under the `$DOMAIN_HOME/test_smp_orders` directory. This feature is turned off by default. To turn it on, you need to update `installation.xml` under `xmlcfg` to:

- Enable debug on class  
`com.sigma.samp.ordermgr.process.ProcessOrder`
- Change trace level to 3
- Enable the debug configuration by running `tools/cfgupd.sh` or restart the domain

## *Review Order in XML from Database*

Order value is also stored in the `ordr_req` column of the `sub_ordr` table. To save database space, a trigger of the `sub_ordr` table will erase the `ordr_req` column after an order has been successfully processed.

This means you can review an order XML saved in the `ordr_req` column for an unfinished order or a non-fully-completed order.

## *SmpDebug*

The Service Management Platform Debug Tool allows the user to view of the entire impact configuration of the system in order to resolve configuration related issues. It helps to analyze the problem and to solve it quickly and easily. See the “Service Management Platform Configuration Troubleshooting Guide” for details.

# XML Order API

OrderManager supports the following protocols for XML order request:

- XML over JMS

Send XML request to JMS queue

“com.sigma.jms.samp.ordermgr.XmlRequestQueue”. Indicate the “replyTo” destination in the request message. Order Manager will send back an XML response/exception there.

XML request will be executed under user defined by

“xml\_over\_jms\_user\_name” policy in `ordermgr_cfg.xml`. Make sure this user has enough permission for all the XML requests.

- Web Services

The Web Services address is at `http://${SERVER_URL}/SmpXmlOrderApi/xmlorder`, where `${SERVER_URL}` is the URL address of Weblogic Cluster, proxy server or Weblogic node server. It is a synchronized type API, XML. The response/exception is replied back to client as operation return.

- XML over EJB

Call `executeXmlRequest()` method on `JVTServiceActivationBean`. This method takes an XML request as String and replies back XML response/exception as String.

The signature is as: `String executeXmlRequest(String xml)` throws `RemoteException`; Since it is an EJB call, it will run under order client transaction context and has better performance result.

## WSDL Overview

---

The WSDL is included in the SMP SDK under the `xml/schemas` directory as shown below:

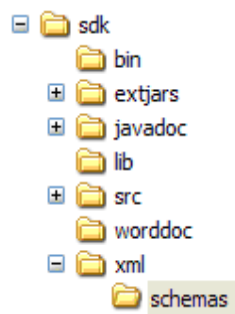


Figure 1: Schema Directory Structure

The details are as follows:

```
<definitions xmlns:tns="http://www.sigma-systems.com/
schemas/3.1/SmpXmlOrderAPI"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://www.sigma-systems.com/
schemas/3.1/SmpXmlOrderAPI">
  <types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/
XMLSchema" elementFormDefault="qualified"
targetNamespace="http://www.sigma-systems.com/
schemas/3.1/SmpXmlOrderAPI" />
  </types>
  <message name="executeXml">
    <part name="xmlRequest"
type="partns:string" xmlns:partns="http://
www.w3.org/2001/XMLSchema" />
  </message>
  <message name="executeXmlResponse">
    <part name="xmlResponse"
type="partns:string" xmlns:partns="http://
www.w3.org/2001/XMLSchema" />
  </message>
  <portType name="xmlorderPort">
    <operation name="executeXml">
      <input message="tns:executeXml" />
      <output
message="tns:executeXmlResponse" />
    </operation>
  </portType>
  <binding name="xmlorderPort"
type="tns:xmlorderPort">
    <soap:binding style="rpc" transport="http:/
/schemas.xmlsoap.org/soap/http"/>
    <operation name="executeXml">
      <soap:operation soapAction="executeXml"/>
    </operation>
  </binding>
</definitions>
<input>
  <soap:body use="encoded" encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.sigma-systems.com/schemas/3.1/
SmpXmlOrderAPI"/>
</input>
<output>
```

```

        <soap:body use="encoded" encodingStyle="http://
        schemas.xmlsoap.org/soap/encoding/"
        namespace="http://www.sigma-systems.com/schemas/
        3.1/SmpXmlOrderAPI"/>
    </output>
</operation>
</binding>
<service name="xmlorder">
    <port name="xmlorderPort"
    binding="tns:xmlorderPort">
        <soap:address location="http://
        localhost:8088/SmpXmlOrderApi/xmlorder"/>
    </port>
</service>
</definitions>

```

Operation `executeXml` takes string as input and returns back string as response. Since XML request uses `RequestType` (for example `abortOrderByKeyRequest`) as root element, Order Manager does not have a problem recognizing the request type and processing it.

## Client Samples

---

The following are client examples written in JAVA.

### *XML over JMS*

Client specifies the response queue and sends an XML request in text message.

```

QueueConnection qConn = EjbUtil.getQueueConnection();
qConn.start();

QueueSession qSession =
qConn.createQueueSession(false,
    Session.AUTO_ACKNOWLEDGE);

QueueSender qSender = qSession.createSender(queue);
Message req = qSession.createTextMessage(xml);
req.setJMSCorrelationID("testing xml");

Queue
replyQueue=EjbUtil.getQueue("com.sigma.jms.samp.oredermgr
.XmlResponseQueue");
req.setJMSReplyTo(replyQueue);
qSender.send(req);
qSender.close();

```

```

        QueueReceiver
receiver=qSession.createReceiver(replyQueue);
        Message msg=receiver.receive();
        String result=((javax.jms.TextMessage)msg).getText();
        receiver.close();
        qSession.close();
qConn.close();

```

## Web Services

An example of standalone client written in JAVA is as follows:

```

ServiceFactory factory = ServiceFactory.newInstance();
String targetNamespace =
    "http://www.sigma-systems.com/schemas/3.1/
SmpXmlOrderAPI/";
QName serviceName =
    new QName(targetNamespace, "xmlorder");
QName portName =
    new QName(targetNamespace, "xmlorderPort");
QName operationName = new QName("xmlorderPort",
"executeXml");
Service service = factory.createService(serviceName);
Call call = service.createCall();
call.setProperty(Call.USERNAME_PROPERTY, "samp.csra1");
    call.setProperty(Call.PASSWORD_PROPERTY, "pwcsra1");
call.setPortTypeName(portName);
call.setOperationName(operationName);
call.addParameter("xmlRequest",
    new QName("http://www.w3.org/2001/XMLSchema",
"string"),
    ParameterMode.IN);
call.setReturnType(new QName("http://www.w3.org/2001/
XMLSchema", "string"));
// set end point address
String address = "http://" + getUrl() + "/SmpXmlOrderApi/
xmlorder";
call.setTargetEndpointAddress(address);
// invoke the remote web service
String xmlResponse = (String) call.invoke(new Object[]
{xmlResquest});

```



You can also use `WebServicesClient.java` provided in the SDK. It includes methods for each type of request supported by `OrderManager`. Here is sample for using it:

```
WebServicesClient client = new
WebServicesClient(hostName,port,userName,passWd);
String resp=client.abortOrderByKey(xmlReq);
```

## XML over EJB

```
String xmlRequest=Ö
Context ctx=new InitialContact();
JVTsmpActivationSession
session=JVTsmpActivationSessionLocator.locate(ctx);
String
xmlResponse=session.executeXmlRequest(xmlRequest);
```

## XSD for XML Request

---

The 15 requests supported in XML Order API are as follows:

- Execute order sync
- Execute order async
- Create order by value
- Start order by key
- Abort order by key
- Get entity by key
- Get service by key
- Get order by key
- Make order value
- Make entity value
- Query order
- Query entity
- Set order by value
- Repair order by key
- Replicate order by key

OSSJ defines schemas for the CBE package and & Service activation package. SMP XML API extends four of them and adds in four extensions.

For example, `XmlSmpCBECORESchema.xsd` extends `SSJXmlCBECORESchema.xsd`.

Table 1: Schema for CBE &amp; Service Activation Package

OSSJ Xml Schemas	SMP extentions
XmlCBECoreSchema.xsd	XmlSmpCBECoreSchema.xsd
XmlCBEServiceSchema.xsd	XmlSmpCBEServiceSchema.xsd
XmlInventorySchema.xsd	
XmlServiceInventorySchema.xsd	Don't support
XmlCBECrossSchema.xsd	
XmlCBECustomerSchema.xsd	
XmlCBEDataTypesSchema.xsd	
XmlCBELocationSchema.xsd	
XmlCBEProductSchema.xsd	
XmlCBEResourceSchema.xsd	
XmlCrossInventorySchema.xsd	
XmlProductInventorySchema.xsd	
XmlResourceInventorySchema.xsd	
XmlCommonSchema.xsd	XmlSmpCommonSchema.xsd
XmlServiceActivationSchema.xsd	XmlSmpServiceActivationSchema.xsd

The following are naming prefixes for schemas used by SMP and SDK:

XML Schema	Naming prefix
XmlCBECoreSchema.xsd	cbecore
XmlSmpCBECoreSchema.xsd	smpcbe
XmlCBEServiceSchema.xsd	cbesvc
XmlSmpCBEServiceSchema.xsd	smpsvc
XmlCommonSchema.xsd	cmn
XmlSmpCommonSchema.xsd	smpcmn
XmlServiceActivationSchema.xsd	sa
XmlSmpServiceActivationSchema.xsd	smps

The 15 XML requests and request contents supported by SMP are defined in the above schemas. For OSSJ XML schema, see OSSJ Specification for details. All OSSJ XML schemas and SMP extension schemas are included in the SMP SDK under the `xml/schemas` directory as shown below:

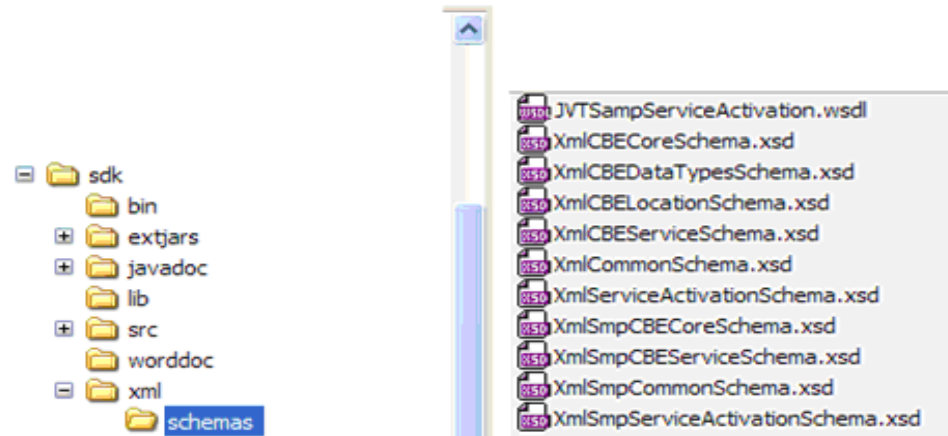


Figure 2: Schema Directory

The following are overviews of SMP extensions.

## *XmlSmpCommonSchema.xsd*

This schema defines Exceptions and some common types, such as:

- NameQuery
- ParamListType
- AuditInfoType

### Name Query

NameQuery Type includes a query name and a list of parameters. NameQuery will be used in order query, service query and NamedQueryKey.

The following diagram shows the Named Query structure:

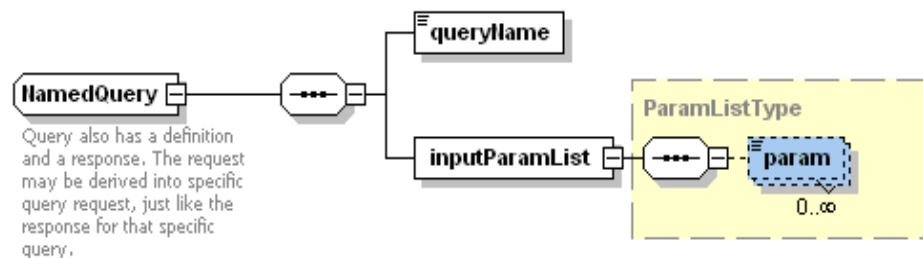


Figure 3: Name Query

An example for nameQuery is as follows:

The query name is called “find\_hsd\_by\_mac” and condition is cm\_mac ==‘CP99122ef’.

```
< nameQuery>
  <queryName>find_hsd_by_mac</smp:queryName>
  < inputParamList>
    < param name="cm_mac">CP991223f</smpcmn:param>
  </ inputParamList>
</nameQuery>
```

## Param Type

ParamType is member of ParamList. The ParamType includes an attribute called name, which is used to store parameter name. Parameter value is the value of node. Use wrap data in CDATA if the value includes any special character. The param type structure is as follows:

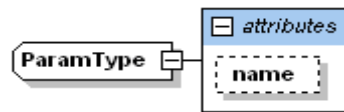


Figure 4: Param Type

ParamType is member of ParamList.

## ParamList Type

The ParamList is a list of ParamType. This is referred by order types and NameQueryType for storing order parameter or namedQuery condition. The param list type structure is as follows:

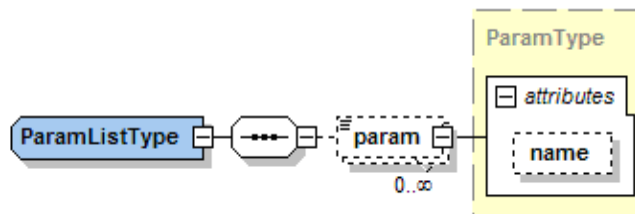


Figure 5: Param List Type

## AuditInfo Type

The AuditInfoType defines createdDateTime, modifiedDataTime, createdBy and modifiedBy. It is used by SubSvcType, SubType, Order and LineItem type definition. The value of this type is normally optional. This means XML (such as XML for Subscriber value) generated by client does not need to populate the value of AuditInfo. But the XML returned from SMP normally has the value populated. The AuditInfo Type syructure is as follows:

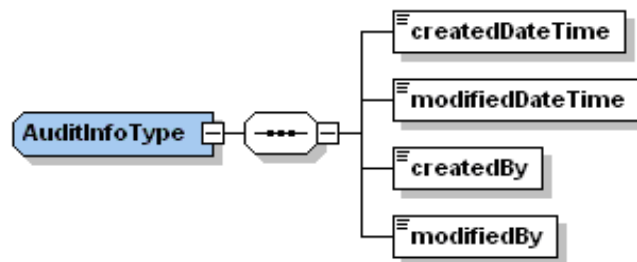


Figure 6: Audit Info Type

## GRefType

The GRefType is used in exception thrown back from SMP to tell which and where entity is. It has three attributes: entityKey, entityClassNm and localAppEnv.

Element appEnv and attribute localAppEnv represent entity location. They are optional and exclusive between each other.

Here is an instance of GRefType:

```

<smpcmn:gRef entityClassNm="SubSvc" entityKey="10100">
  <smpcmn:entityBriefDescription>
    CP16-14-991223D
  </smpcmn:entityBriefDescription>
</smpcmn:gRef>

```

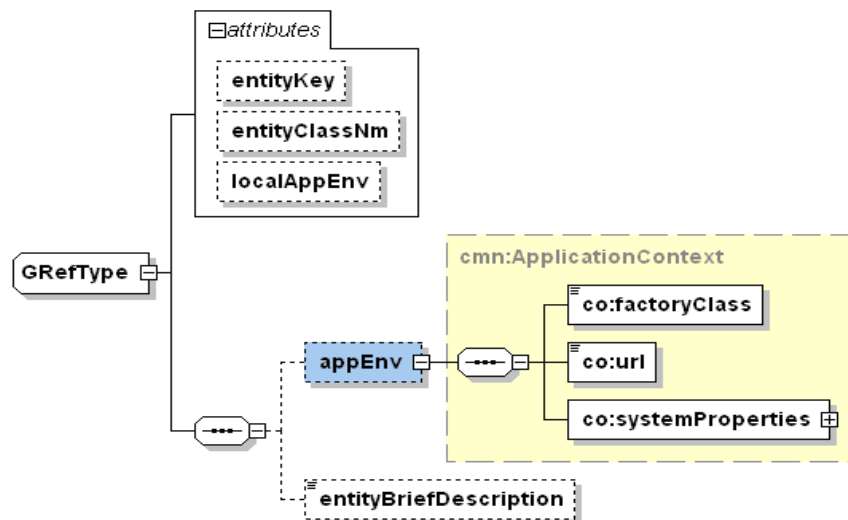


Figure 7: GRef Type

## AppErrorType

the AppErrorType represents a multilingual error message. It includes an error code, list of error parameters, localized error message and an entity reference (type of GRefType). Attribute “xml:lang” tells language of error message.

For example:

```
<smpcmn:appError>
  <smpcmn:errorCode>VLDERR_MAX_CHAR_INVALID</smpcmn:errorCode>
  <smpcmn:errorParamList>
    <smpcmn:param resourceNm="SubSvcSpec">
      samp_ds_hsd_access.cm_mac
    </smpcmn:param>
    <smpcmn:param resourceNm="">CP16-14-991223D</smpcmn:param>
    <smpcmn:param resourceNm="">12</smpcmn:param>
    <smpcmn:param resourceNm="">15</smpcmn:param>
  </smpcmn:errorParamList>
  <smpcmn:errorMessage xml:lang="en CA">The number
    of characters is more than maximum allowed
    number, parmNm = "samp_ds_hsd_access.cm_mac"
    value = "CP16-14-991223D" maxNum = 12 currentNum
    = 15</smpcmn:errorMessage>
  <smpcmn:gRef entityClassNm="SubSvc"
    entityKey="10100">
    <smpcmn:entityBriefDescription>CP16-14-991223D
    </smpcmn:entityBriefDescription>
  </smpcmn:gRef>
</smpcmn:appError>
```

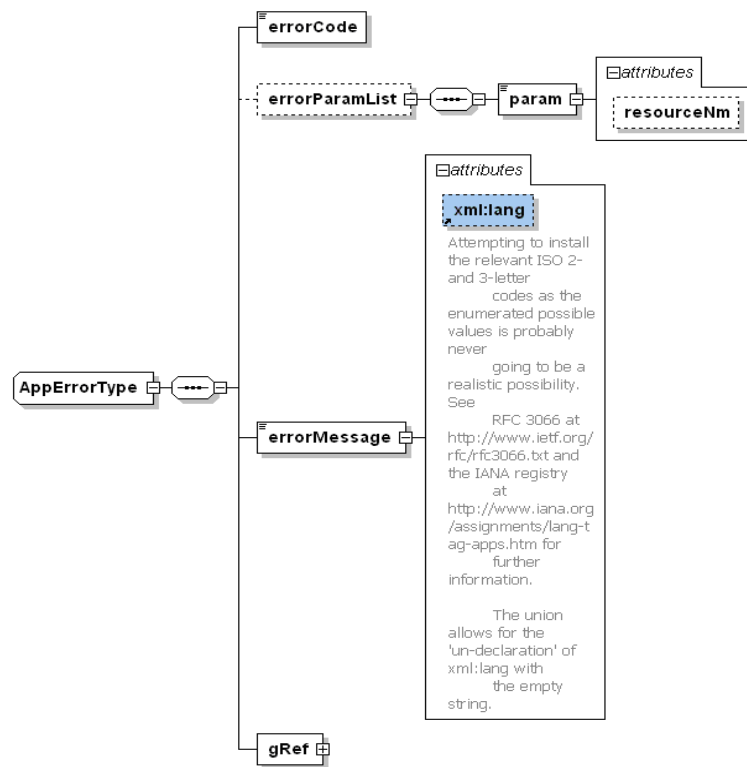


Figure 8: App Error Type

## Exceptions

The list of exception types defined in this schema is as follows:

- SmpIllegalStateException
- SmpObjectNotFoundException
- SmpIllegalArgumentException
- SmpRemoteException
- SmpCreateException

Those types contain a list of AppErrorType on top of OSSJ corresponding exception definition.

An example of SmpIllegalStateException:

```
<smpsa:illegalArgumentException
xsi:type="smpcmn:SmpIllegalArgumentException">
  <cmn:message>Sub profile validation failed.</cmn:message>
  <smpcmn:applicationErrors>
    <smpcmn:item>
      <smpcmn:errorCode>VLDERR_MAX_CHAR_INVALID</smpcmn:errorCode>
```

```

<smpcmn:errorParamList>
  <smpcmn:param resourceNm="SubSvcSpec">
    samp_ds_hsd_access.cm_mac
  </smpcmn:param>
  <smpcmn:param resourceNm="">CP16-14-991223D</smpcmn:param>
  <smpcmn:param resourceNm="">12</smpcmn:param>
  <smpcmn:param resourceNm="">15</smpcmn:param>
</smpcmn:errorParamList>
  <smpcmn:errorMessage xml:lang="en_CA">The number
of characters is more than maximum allowed number,
parmNm = "samp_ds_hsd_access.cm_mac" value = "CP16-
14-991223D" maxNum = 12 currentNum = 15</smpcmn:errorMessage>
  <smpcmn:gRef entityClassNm="SubSvc"
entityKey="10100">
    <smpcmn:entityBriefDescription>
      CP16-14-991223D
    </smpcmn:entityBriefDescription>
  </smpcmn:gRef>
</smpcmn:item>
</smpcmn:applicationErrors>
</smpsai:illegalArgumentException>

```

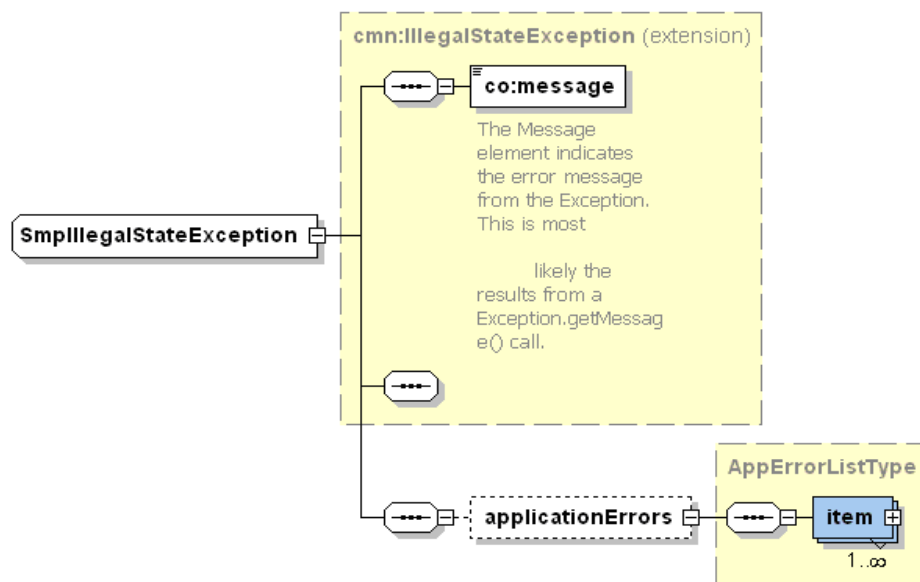


Figure 9: Exception



## XmlSmpCBEServiceSchema.xsd

This schema defines XML type for entity keys such as:

- EntityKeyType (base type of other key types)
- SubUserKeyType
- SubAddressKeyType
- SubContactKeyType
- SubKeyType

And entity types such as:

- EntityType (base type of other entity types)
- SubType
- SubContactType
- SubAddressType
- SubUserType

It also defines getEntityByKey and MakeEntityValue request/response/Exception.

## EntityParamListType

EntityParamList includes a list of parameter (named as param) and old parameter (named as oldParam). Both of param and oldParam are in ParamType defined in XmlSmpCommonSchema.xsd.

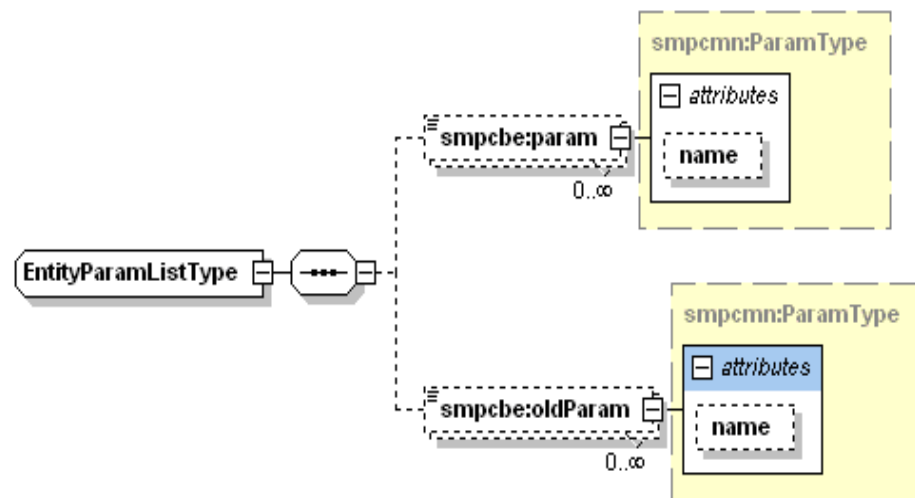


Figure 10: Entity Parm List Type

Param is used to store non-reserved parameters of entity. An entity may have tens of non-reserved parameters. The client only needs to add the param entry into paramList if it wants to overwrite the default value.

The OldParam is used to store the old parameter value. If there is not any oldParam element in EntityParamList, Order Manager assumes order client intends to update all the parameters shown by param element(s). If one oldParam element exists, Order Manager only collects parameter which has oldParam element. For example, ParmList includes four param and two oldParm. Since there are oldParam(s), Order Manager only picks up parameter changes for parameter “movie\_name” and “movie\_minutes”.

```

<smpsvc:paramList>
  <smpcbe:param name="tv_channel">303</smpcbe:param>
    <smpcbe:param name="movie_minutes">140</smpcbe:param>
    <smpcbe:param name="movie_name">Star Wars: Episode III</smpcbe:param>
      <smpcbe:param name="movie_desc">After three years of fighting in the Clone Wars, Anakin Skywalker concludes his journey towards the Dark Side of the Force, putting his friendship with Obi Wan Kenobi and his marriage at risk.</smpcbe:param>
    <smpcbe:oldParam name="movie_minutes">280</smpcbe:oldParam>
    <smpcbe:oldParam name="movie_name">cars</smpcbe:oldParam>
</smpsvc:paramList>

```

For regular subscriber entity, oldParam works as indicator; the value is discarded. Order Manager only uses current value load from inventory as old value. For event type service, parameter is stored outside SMP. The Order Manager uses the value in oldParam as old value for order processing.

## AssocType

AssocType extends AssociationValue defined in XmlCBECoreSchema.xsd, adds associationType, zEndKey and the optional element changeAction.

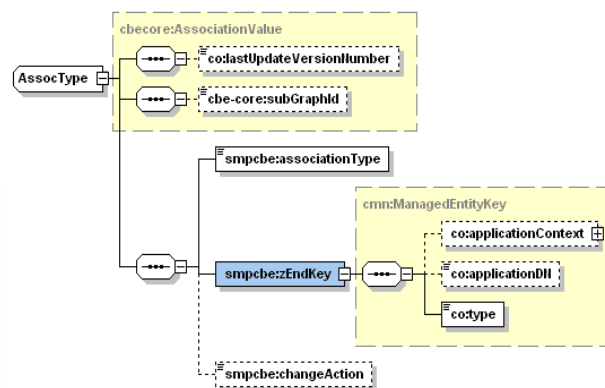


Figure 11: Association Type

ChangeAction works as an indicator. If want to an add/delete association using entity update line item, you need to put “add” for the new association or “delete” for to be deleted the association to be deleted. Otherwise, order manager may ignore the change(s).

An association value example:

```
<smcbe:association>

    <smcbe:associationType>SiteSurvey_Has_Address</smcbe:associationType>

    <smcbe:zEndKey
      xsi:type="smcbe:SubAddressKeyType">
        <cmn:type>SubAddressSpec:-</cmn:type>
        <smcbe:externalKey>home</smcbe:externalKey>
      </smcbe:zEndKey>
    <smcbe:changeAction>add</smcbe:changeAction>
  </smcbe:association>
```

If an association is shown in an association action line item, changeAction is not required and will be ignored if it is presented, since line item action already presents the intension of client.

## AssocListType

AssocListType is type of AssocType list.

## EntityKeyType

EntityKeyType extends EntityKey defined in `XmlCBECoreSchema.xsd`, adds primaryKey, nameQuery and externalKey. Each Entitykey can have primaryKey (known as id key), externalKey or both. nameQuery is only used only when Client doesn't know the id key and external key, but know how to unique identify the entity by NamedQuery.

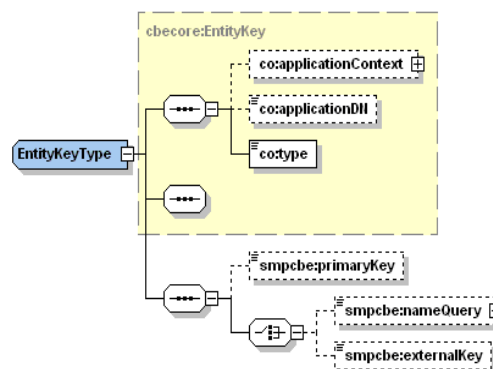


Figure 12: Entity Key Type

An example with external key:

```
<smpcbe:key>
    <cmn:type>SiteSurveySpec:cable</cmn:type>
    <smpcbe:externalKey>surveyCP991</
smpcbe:externalKey>
</smpcbe:key>
```

Another sample with NamedQuery key:

```
<smpsa:entityKey xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpCBECORESchema"
xsi:type="smp:SubUserKeyType">
    <cmn:type>SubUserSpec:primary</cmn:type>
    <smpcbe:nameQuery>
        <smpcmn:queryName>get_id_by_parm</
smpcmn:queryName>
        <smpcmn:inputParamList>
            <smpcmn:param name="user_id">userCP991</
smpcmn:param>
        </smpcmn:inputParamList>
    </smpcbe:nameQuery>
</smpsa:entityKey>
```

## SubKeyType, SubContactKeyType, SubUserKeyType and SubAddressKeyType

They all derive from EntityKeyType. Only types are different.

## EntityType

EntityType extends EntityValue from XmlCBECORESchema.xsd, add key, state, paramList, associationList and auditInfo:

- key  
Type of EntityKeyType, key of current entity
- state  
Type of xs:string, represents entity state.
- paramList  
Type of EntityParamListType, stores <name, value> pairs of for non-reserved parameters.
- auditInfo  
Type of AuditInfoType, stores modifiedDtm, createdDtm, modifiedBy and createdBy of the current entity.  
Client doesn't need to populate those values and will be ignored by OrderManager.
- associationList

Type of [assocListType](#), association values of current entity.

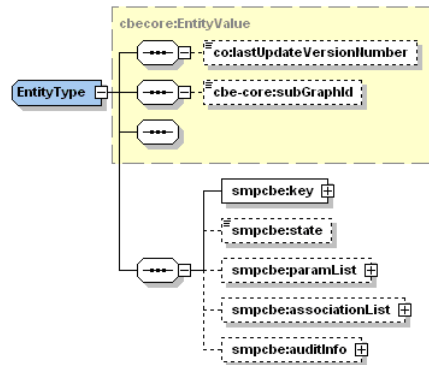


Figure 13: Entity Type

An example of entity value XML:

```
<smpcbe:entityValue>
  <cmn:lastUpdateVersionNumber>0</
cmn:lastUpdateVersionNumber>
  <smpcbe:key>
    <cmn:type>SiteSurveySpec:cable</cmn:type>
    <smpcbe:externalKey>surveyCP991</
smpcbe:externalKey>
  </smpcbe:key>
  <smpcbe:state>inactive</smpcbe:state>
  <smpcbe:paramList>
    <smpcbe:param name="street_nm">55 york</
smpcbe:param>
    <smpcbe:param name="country">CANADA</
smpcbe:param>
    <smpcbe:param name="city">TORONTO</smpcbe:param>
  </smpcbe:paramList>
  <smpcbe:associationList>
    <smpcbe:association>
      <smpcbe:associationType>SiteSurvey_Has_Addres
s</smpcbe:associationType>
      <smpcbe:zEndKey
xsi:type="smpcbe:SubAddressKeyType">
        <cmn:type>SubAddressSpec:-</cmn:type>
        <smpcbe:externalKey>home</
smpcbe:externalKey>
      </smpcbe:zEndKey>
    </smpcbe:association>
  </smpcbe:associationList>
</smpcbe:entityValue>
```

```

        </smpcbe:associationList>
    </smpcbe:entityValue>

```

## SubContactType, SubAddressType and SubUserType

All are derived from the Entitytype. The SubAddressType adds an attribute called isDefault, which is used to identify whether it is a default address of subscriber.

## SubType

SubType extends Entitytype, adds entityList and three attributes and one element as follows:

- entityList

entityList includes all the entities of subscriber, including services.

For subscriber included in Snapshot Order, or “add” action line item of action order, you can include all the subscriber entities in entityList. OrderManager will create the subscriber, as well as included entities. If just want to update a subscriber using an action order, don’t need to include entity in it. Order Manager only takes parameter changes from subscriber parameter list and ignores the entityList presented in the subscriber XML.

- subscriberType

It is type of xs:string, store the type of subscriber. Subscriber type is defined in `cfg/com/sigma/samp/vframe/schemas/smp_ref_data.xml` and `smp_ref_data_imp.xml` under `implementation xmllcfg` directory. Deinition is stored in `ref_data` element with type equals `REF_SUB_TYP`.

- serviceProvider

It is type of xs:string, store the subscriber’s service provider. Permitted service provider value is defined in `svccat/SvcProviders.xml` under `implement xmllcfg` directory.

- Locale

It is type of xs:string, store the subscriber’s locale (in string format).

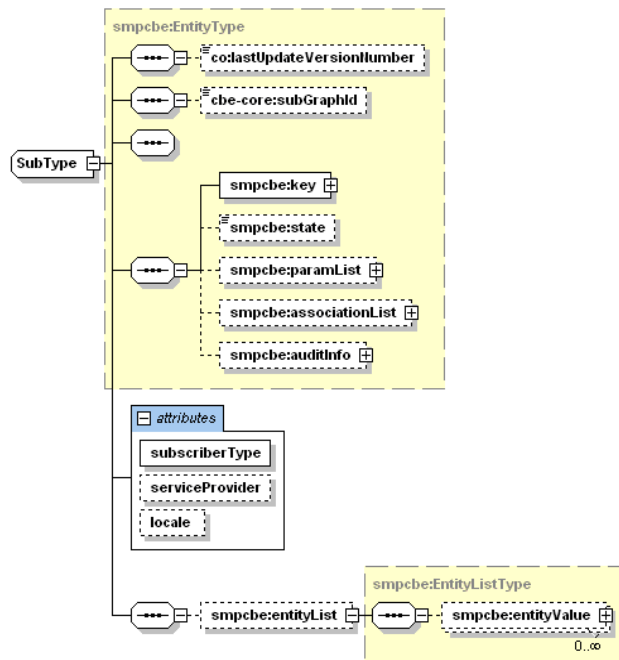


Figure 14: SubType

An XML example of subscriber XML is as follows:

```
<smpsa:subscriber serviceProvider="Liberate"
subscriberType="residential" locale="en_CA">
  <cmn:lastUpdateVersionNumber>0</cmn:lastUpdateVersionNumber>
  <smpcbe:key xsi:type="smpcbe:SubKeyType">
    <cmn:type>SubSpec:-</cmn:type>
    <smpcbe:externalKey>CP40-903</smpcbe:externalKey>
  </smpcbe:key>
  <smpcbe:state>active</smpcbe:state>
  <smpcbe:paramList>
    <smpcbe:param name="hotlined_ind">N</smpcbe:param>
    <smpcbe:param name="acct">903812345</smpcbe:param>
    <smpcbe:param name="user_group">EAST</smpcbe:param>
    <smpcbe:oldParam name="status">inactive</smpcbe:oldParam>
  </smpcbe:paramList>
  <smpcbe:entityList>
  </smpcbe:entityList>
</smpsa>
```

## XmlSmpCBEServiceSchema.xsd

### SubSvcKeyType

The SubSvcKeyType extends ServiceKey defined in XmlCBEServiceSchema.xsd, and adds primaryKey, externalKey and nameQuery.

As in the EntityKeyType, the primary key is for subSvcId and nameQuery key is only to be used when you do not know both id key and external key, and only know how to identify the entity by namedQuery.

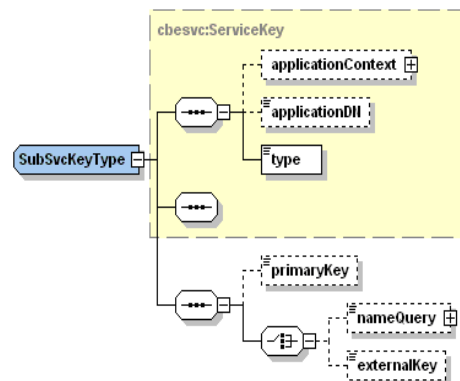


Figure 15: Sub Service Key Type

Here is an example of SubSvcKey with both id key and external key:

```

<smpsvc:serviceKey>
  <cmn:type>SubSvcSpec:smp_device</cmn:type>
  <smpsvc:primaryKey>760309</smpsvc:primaryKey>
  <smpsvc:externalKey>
    Svc_smp_device001056294397386Svrqacore07_adm_svrSeq1
  </smpsvc:externalKey>
</smpsvc:serviceKey>
  
```

### Service Value

The SubSvcType in the SMPCBE schema adds serviceKey and a list of optional elements on top of ServiceValue defined in XmlCBEServiceSchema.xsd:

- serviceKey  
Type of [ServiceKeyType](#), store key of current service.
- parentServiceKey  
Type of [ServiceKeyType](#), specify parentSubSvc.
- paramList



Type of EntityParamListType, store list of <name, value> pairs of non-reserved parameter.

- endDateTime

Type of dateTime, store real end time if the subscriber service has been deleted. See the icool offi feature for details.

- auditInfo

Type of AuditInfoType, store modifiedDtm, createdDtm, modifiedBy, createdBy information of service when it is loaded and returned from SMP. Order client doesn't need to provide those values.

- associationList

Type of assocListType, store a list of association values belonged to current service value.

- childServiceList

Composed services use only. It contains all the children services.

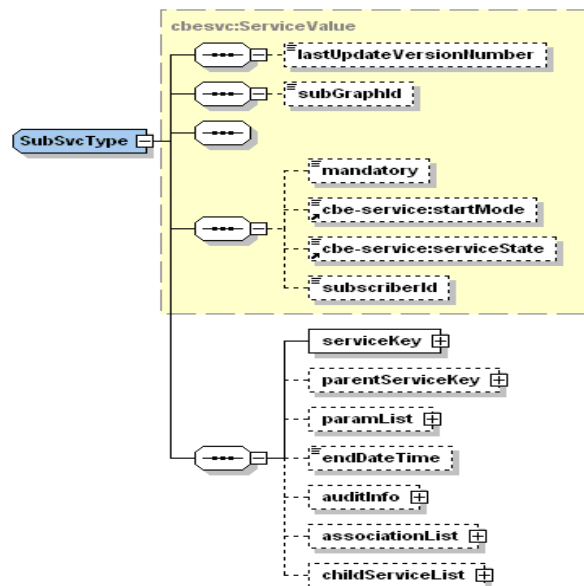


Figure 16: Sub Service Type

SMP doesn't support subGraphId, mandatory, and startMode optional elements defined in CBE ServiceValue. Type of serviceState is ServiceStateType which is defined in XmlSmpServiceActivationSchema.xsd.

The following is an instance of SubSvcType:

```
<smpsvc:serviceValue>
  <cbesvc:serviceState provisionState="active"
xsi:type="smpsa:SubSvcStateType">active</
cbesvc:serviceState>
  <smpsvc:serviceKey>
```

```

cmn:type>
    <cmn:type>SubSvcSpec:hsd_ultra_access</
smpsvc:externalKey>
    <smpsvc:externalKey>CP13-992svc125</
    </smpsvc:serviceKey>
    <smpsvc:parentServiceKey>
cmn:type>
    <cmn:type>SubSvcSpec:high_speed_platinum</
smpsvc:externalKey>
    <smpsvc:externalKey>CP13-992svc12004</
    </smpsvc:parentServiceKey>
    <smpsvc:paramList>
        <smpcbe:param
name="rf_downstream_freq">999000</smpcbe:param>
        <smpcbe:param
name="cm_manufacturer">Motorola</smpcbe:param>
        Ö
        <smpcbe:param name="cm_model">P0222</
smpcbe:param>
        <smpcbe:param name="qlty_of_svc">ultra</
smpcbe:param>
        <smpcbe:param name="max_downstream_bw">2
MB</smpcbe:param>
    </smpsvc:paramList>
    <smpsvc:associationList>
        <smpcbe:association>

<smpcbe:associationType>service_on_address</
smpcbe:associationType>
        <smpcbe:zEndKey
xsi:type="smpcbe:SubAddressKeyType">
            <cmn:type>SubAddressSpec:-</cmn:type>
            <smpcbe:externalKey>primary</
smpcbe:externalKey>
            </smpcbe:zEndKey>
        </smpcbe:association>
    </smpsvc:associationList>
</smpsvc:serviceValue>

```

## XmlSmpServiceActivationSchema.xsd

OSSJ SA specification defines XmlServiceActivationSchema.xsd. It defines OrderValue and OrderKey type. OrderValue includes a list of attributes. SMP supports most of them, including:

- apiClientId

This is a required attribute; it must need to be populated in the order request.

- actualCompletionDate

This is an output attribute. SMP ignores it if it is included in the incoming order. It is always shown in the query result for closed order.

- Description

This is an optional attribute for order description.

- orderKey

This is an optional attribute. SMP ignores it if it is included in the create/execute order request.

- priority

This is an optional incoming attribute.

- lastUpdateVersionNumber

This is an output attribute. SMP ignores it if it is included in the incoming order. It is always shown in the order query result.

- orderState

This is an output attribute. SMP ignores it if it is included in the incoming order. It is always shown in the order query result.

- services

Only used in the OSSJ order.

This schema also defines list of Request/Response/Exception for:

- Create order by value
- Start order by key
- Abort order by key
- Get order by key
- Make order value
- Query order
- SetOrderByValue

SMP XmlSmpServiceActivationSchema.xsd extends this schema, adds definitions like:

- All SMP order values, which extends OSSJ OrderValue
- SubSvcType and SubSvcKeyType for OSSJ type order
- Request/Response/Exception for:
  - Execute order sync
  - Execute order async
  - Get service by key
  - Repair order by key
  - Replicate order by key

**Snapshot Order Value** Snapshot extends OrderValue defined in XmlServiceActivationSchema.xsd to include a full subscriber and add:

- orderParamList

It is type of ParamListType defined in the XmlSmpCommonSchema.xsd, stores order parameters, for example has \_groups. For order attributes like priority, can be stored in orderParamList or element of OrderValue.

- auditInfo

It is of type of AuditInfoType defined in the XmlSmpCommonSchema.xsd, stores order's auditing information, such as created\_dtm, modified\_by. This value is not expected in the incoming order; it always appears in the order value returned from OrderManager.

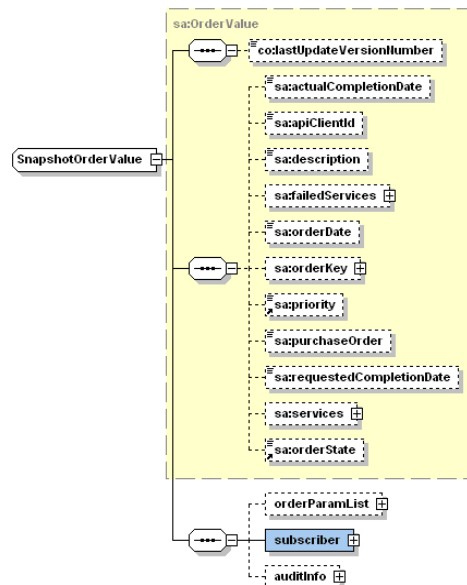


Figure 17: Snap Shot Order Value

*Action order*

ActionOrderValue extends OrderValue defined in XmlServiceActivationSchema.xsd and adds in

subKey, orderparamList, and auditInfo. It also adds orderItemList, which contains list of order items.

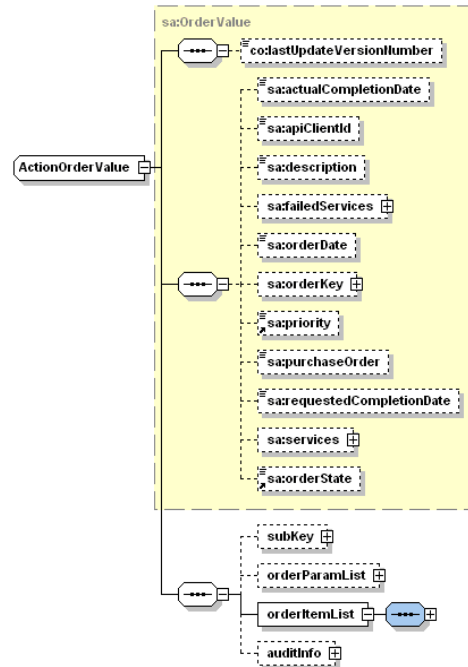


Figure 18: Action Order Value

OrderItemType is LineItem type definition in XML. It includes:

- orderItemKey  
Key of LineItem. Client doesn't need to provide it in new created action order value. It is always included in the LineItem returned from OrderManger.
- action  
For action against subscriber entity.
- itemParamList  
For containing line item parameters.
- entityKey  
It is the subscriber entity key. This is optional when entityValue of LineItem is populated.
- entityValue  
It is the subscriber entity value.

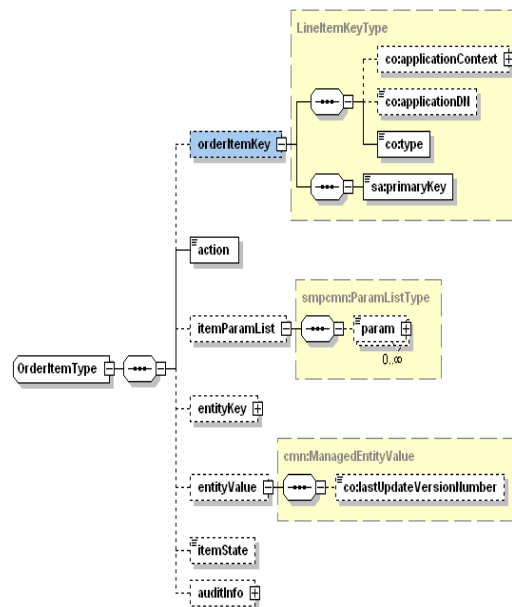


Figure 19: Order Item Type

An example of action order in XML format is as follows:

```
<smpsa:SampOrder xmlns:smpsa="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cmn="http://java.sun.com/products/oss/xml/Common
" xmlns:sa="http://java.sun.com/products/oss/xml/
ServiceActivation" xmlns:smpcbe="http://www.sigma-
systems.com/schemas/3.1/SmpCBECoreSchema"
xmlns:smpcmn="http://www.sigma-systems.com/schemas/3.1/
SmpCommonVa
lues" xmlns:smpsvc="http://www.sigma-systems.com/schemas/
3.1/SmpCBEServiceSchema" xmlns:cbesvc="http://
java.sun.com/products/oss/xml/CBE/Service">
  <smpsa:orderValue xsi:type="smpsa:ActionOrderValue">
    action order
    <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
    <sa:apiClientId>web</sa:apiClientId>
    <sa:orderKey xsi:type="smpsa:OrderKeyType">
      <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
      <sa:primaryKey>14004</sa:primaryKey>
    </sa:orderKey>
    <sa:priority>3</sa:priority>
    <sa:orderState smpState="open.not_running.not_started"
xsi:type="smpsa:SubOrderStateType">open.not_running.not_s
tarted</sa:orderState>
```

```

    <smpsa:subKey> subscriber key
      <cmn:type>Sub</cmn:type>
      <smpcbe:externalKey>CP85-989</smpcbe:externalKey>
    </smpsa:subKey>
    <smpsa:orderParamList> order parameter
      <smpcmn:param name="queue_request_flag">yes</smpcmn:param>
      <smpcmn:param name="has_groups">yes</smpcmn:param>
    </smpsa:orderParamList>
    <smpsa:orderItemList> list of line items
      <smpsa:orderItem>
        <smpsa:action>delete</smpsa:action>
        <smpsa:itemParamList/>
        <smpsa:entityKey xsi:type="smprvc:SubSvcKeyType">
entity key
          <cmn:type>SubSvcSpec:cgo_resi_additional_login</cmn:type>
          <smpsvc:externalKey>CP85-989-
cgo_resi_additional_login</smpsvc:externalKey>
        </smpsa:entityKey>
      </smpsa:orderItem>
    </smpsa:orderItemList>
  </smpsa:orderValue>
</smpsa:SampOrder>

```

*OSSJ Type Order*

SMP defines OSSJ order types, such as:

- CreateOrderValue
- ModifyOrderValue
- CancelOrderValue.
- ResumeOrderValue
- SuspendOrderValue

They are almost same. All derive from OrderValue defined in XmlServiceActivationSchema.xsd and add elements like subKey, orderParamList and auditInfo.

- subKey

It is type of SubKeyType, specify the subscriber.

Event order doesn't need to provide subKey if it doesn't relate to subscriber

- orderParamList

It is of type of ParamListType defined in the XmlSmpCommonSchema.xsd. It is used to store order parameters, for example has\_groups. For order attributes like priority, can be stored in orderParamList or element of OrderValue.

- auditInfo

It is of type of AuditInfoType defined in the XmlSmpCommonSchema.xsd, stores order's auditing information, such as created\_dtm, modified\_by. This value is not expected in the incoming order; it always appears in the order query result.

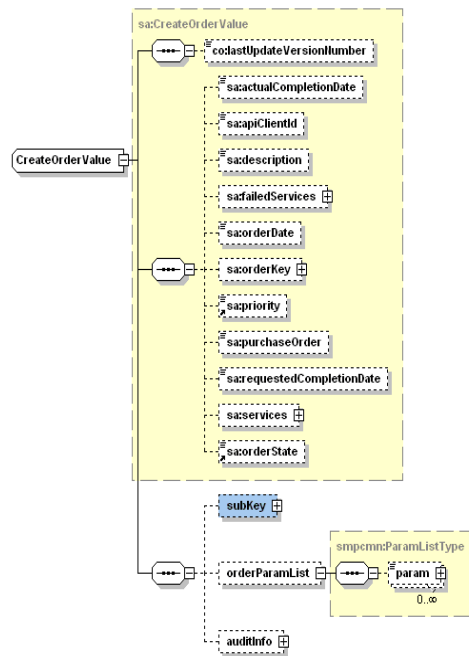


Figure 20: Create Order Value

An example of CancelOrderValue is as follows:

```
<sa:OrderValue xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:CancelOrderValue" >
  <sa:apiClientId>super_system</sa:apiClientId>
  <sa:priority>3</sa:priority>
  <sa:services> services to be deleted
    <sa:item xsi:type="smp:SubSvcType">
      <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
      <sa:serviceKey xsi:type="smp:SubSvcKeyType">
        <cmn:type>SubSvcSpec:samp_webpace</cmn:type>
        <smp:externalKey>CP991-sec_webpace</
smp:externalKey>
```



```

</sa:serviceKey>
<sa:serviceState>active</sa:serviceState>
<smp:parentServiceKey>
  <cmn:type>SubSvcSpec:sigma_basic_resi_hsd</
cmn:type>
  <smp:primaryKey>-1</smp:primaryKey>
  <smp:externalKey>CP991svc9</smp:externalKey>
</smp:parentServiceKey>
</sa:item>
</sa:services>
<smp:subKey> subscriber key
  <cmn:type>SubSpec:-</cmn:type>
  <smpcbe:primaryKey>-1</smpcbe:primaryKey>
  <smpcbe:externalKey>CP991</smpcbe:externalKey>
</smp:subKey>
<smp:orderParamList> order parameters
  <smpcmn:param name="has_groups">yes</smpcmn:param>
  <smpcmn:param name="queue_request_flag">no</
smpcmn:param>
</smp:orderParamList>
</sa:orderValue>

```

### Batch Order

Batch Order is used to group regular order, such as action order. Order itself doesn't include any subscriber entity. It includes subKey, orderParamList and auditInfo on top of OSSJ OrderValue. The XSD definition is as shown below:

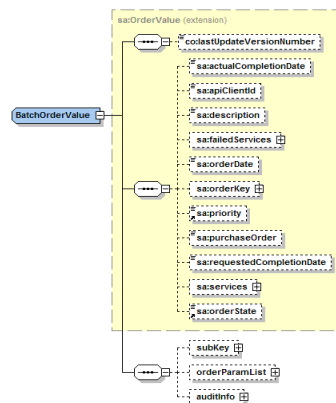


Figure 21: Batch Order

An example of batch order is as follows:

```

<smp:orderValue xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema"

```

```

xsi:type="smp:BatchOrderValue"> batch order type
<ser:apiClientId>web</ser:apiClientId> order source
<ser:priority>3</ser:priority>
<smp:orderParamList> order parameters
    <smpcmn:param name="ordr_owner">samp.csra1</smpcmn:param>
</smp:orderParamList>
</smp:orderValue>

```

### SubSvcType

Type of Service used in OSSJ type order is ServiceValue defined in XmlServiceActivationSchema.xsd not the one defined in XmlCBEServiceSchema.xsd of CBE specification.

They are not compatible.

In order to support OSSJ order, SMP need define another set of SubSvcType and SubSvcKeyType. Snapshot order and Action Type order do not use these types and use the types defined in XmlSmpCBECORESchema.xsd instead.

OSSJ defines lastUpdateVersionNumber, serviceKey, serviceState and subscriberId for its ServiceValue type. To fit the SMP service model, SubSvcType adds the following child elements:

- parentServiceKey  
Type of [ServiceKeyType](#), specify parent's key.
- paramList  
Type of [EntityParamListType](#), store the list of <name, value> pair of non-reserved parameter.
- endDateTime  
Type of dateTime, is used to return value from SMP to telling the "end datetime" of service after service is deleted. For "end datetime", please check the "cool off" feature for detail.
- auditInfo  
Type of [AuditInfoType](#), is used to store the modifiedDtm, createdDtm, modifiedBy, createdBy information of for the service.  
This attribute is read-only to client.
- associationList  
Type of [assocListType](#), is used to store a list of association values belonged to this subscriber service.
- childServiceList  
For composed services use only, include all the children of this subscriber services.

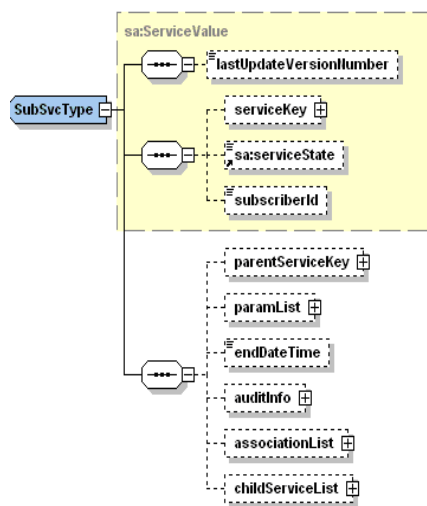


Figure 22: Sub Service Type

Here is an example:

```

    <smpsa:serviceValue>
        <sa:serviceKey
            xsi:type="smpsa:SubSvcKeyType">

            <cmn:type>SubSvcSpec:sigma_primary_email</cmn:type>
                <sa:primaryKey>-1</sa:primaryKey>
                <smpsa:externalKey>CP13-999svc111</
smpsa:externalKey>
                </sa:serviceKey>
                <sa:serviceState provisionState="inactive"

xsi:type="smpsa:SubSvcStateType">inactive</
sa:serviceState>

                <smpsa:parentServiceKey>
<cmn:type>SubSvcSpec:sigma_primary_login</cmn:type>
                <sa:primaryKey>-1</sa:primaryKey>
                <smpsa:externalKey>CP13-999svc110</
smpsa:externalKey>
                </smpsa:parentServiceKey>
                <smpsa:paramList>
                    <smpcbe:param name="fwd_term_dt"
xsi:nil="true"/>
                    <smpcbe:param name="first_nm">JULIA</
smpcbe:param>
                    <smpcbe:param
name="orig_email_pswd">bbbbbbbbbb</smpcbe:param>

```

```

                                <smpcbe:param name="domain">cableco.com</
smpcbe:param>
                                <smpcbe:param name="fwd_eff_dt" xsi:nil="true"/>
                                </smpsa:paramList>
                                <smpsa:associationList>
                                    <smpcbe:association>
                                        <smpcbe:associationType>service_on_address</
smpcbe:associationType>
                                        <smpcbe:zEndKey
xsi:type="smpcbe:SubAddressKeyType">
                                            <cmn:type>SubAddressSpec:-</cmn:type>
                                            <smpcbe:primaryKey>20024</
smpcbe:primaryKey>
                                        </smpcbe:zEndKey>
                                    </smpcbe:association>
                                </smpsa:associationList>
                                </smpsa:serviceValue>

```

### SubSvcKeyType

Type and primary key are request attributes in OSSJ ServiceKey. SMP SubSvcKeyType extends ServiceKey and adds external key and NameQuery.

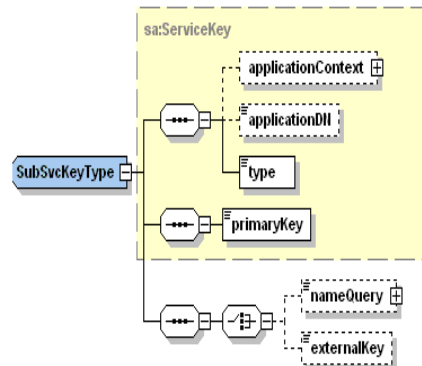


Figure 23: Sub Service Key Type

If the service id is known, you can directly use ServiceKey defined by the OSSJ SA schema. For example:

```

<sa:serviceKey>
    <cmn:type>SubSvcSpec:samp_webpace</cmn:type>
    <sa:primaryKey>12312313</sa:primaryKey>
</sa:serviceKey>

```

If the primary key is not known, you can simply put -1 in the primary key and use externalKey or nameQuery to specify the entity. In this case, you need to use SubSvcKeyType, which is the SMP key extension.

Here is an example of the external key:

```

    <sa:serviceKey xsi:type=ismpsa:SubSvcKeyType1\
      <cmn:type>SubSvcSpec:samp_webpace</cmn:type>
      <sa:primaryKey>-1</sa:primaryKey>
      <smpsa:externalKey>CP991-sec_webpace</
smp:externalKey>
    </sa:serviceKey>

```

And here is another example of the NameQuery Key:

```

    <sa:serviceKey xsi:type=ismpsa:SubSvcKeyType1\
      <cmn:type>SubSvcSpec:samp_webpace</cmn:type>
      <smpsa:nameQuery>
        <smpcmn:queryName>get_id_by_parm</
smpcmn:queryName>
        <smpcmn:inputParamList>
          <smpcmn:param name=icm_mac1\
        </smpcmn:inputParamList>
        </smpsa:nameQuery>
    </sa:serviceKey>

```

### ServiceState

OSSJ ServiceStateType only defines two permitted values:

- active
- inactive

SMP has much more state values than that. In order to bypass this issue, SMP introduces SubSvcStateType which extends OSSJ ServiceStateType. SubSvcStateType adds a property called provisionState.

Here is an example of SubSvcStateType:

```

    <sa:serviceState provisionState="courtesy_block"
      xsi:type="smpsa:SubSvcStateType">active</
sa:serviceState>

```

If the incoming state is an instance of SubSvcStateType, SMP only takes the provisionState attribute value as a state value. We suggest putting using “inactive” as serviceState value when the provisionState is deleted, and iactive1 as serviceState value for other prvisionState values.

If the incoming state is an instance of OSSJ serviceState, SMP takes the value of serviceState even if the value is other than iactive1 or “inactive”.

Since ServiceValue defined in XmlCBEServiceSchema.xsd also uses ServiceStateType defined in OSSJ SA specification, SubSvcType defined in XmlSmpCBEServiceSchema.xsd also uses this state type.

## Request, Response and Exception

Request, Response and Exception are the pieces of information exchanged between XML API client and Order Manager.

Taking the Web Services interface for example, `executeXml` operation sends **ExecuteOrderRequest** to Order Manager. Order Manager processes the request. If the order is executed successfully, **ExecuteOrderResponse** is generated and returned to client; otherwise, **ExecuteOrderException** is sent back.

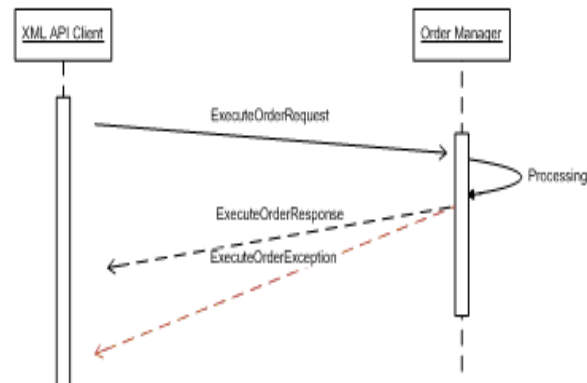


Figure 24: Execute Order

Taking `executeOrder` as an example, here are the request/response/exception schema definitions:

To execute order request find the following schema definition:

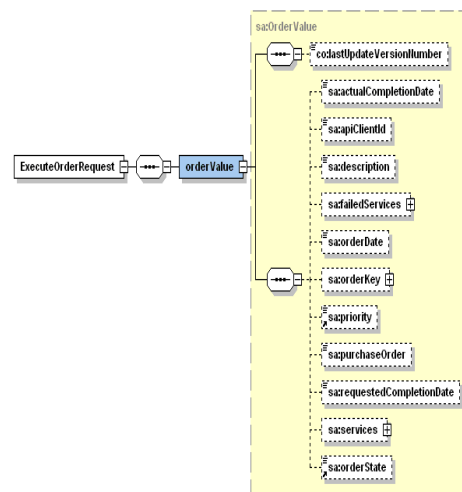


Figure 25: Execute Order Request

To execute order response find the following schema definition:

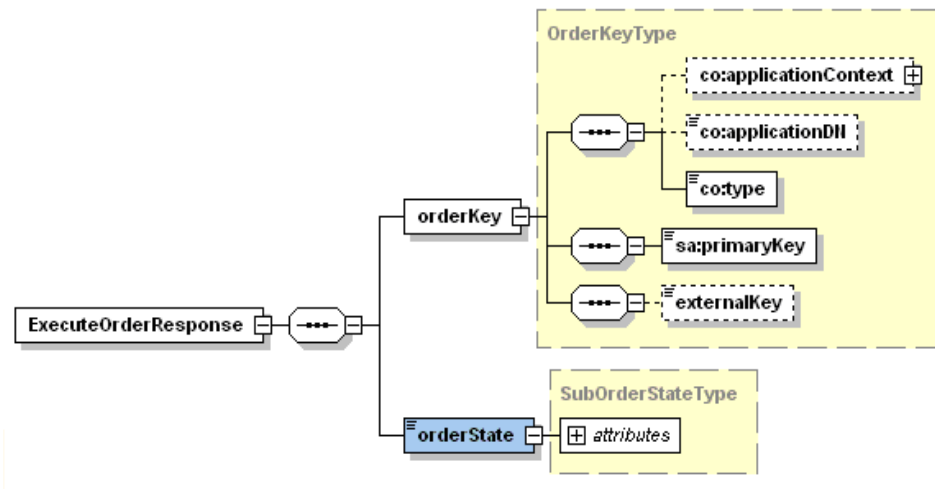


Figure 26: Execute Order Response

To execute order response find the following schema definition:

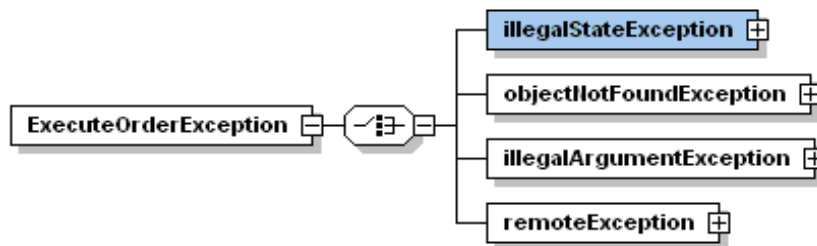


Figure 27: Execute Order Exception

The exception document thrown from Order Manager looks similar. It always includes a detailed exception, such as `illegalStateException`. For more information, see [Exceptions](#). The following is an example of `executeOrderException`:

```

<executeOrderException xmlns="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:cmn="http://java.sun.com/products/oss/xml/Common"
  xmlns:smpcmn="http://www.sigma-systems.com/schemas/3.1/
SmpCommonValues">
    <illegalArgumentException
      xsi:type="smpcmn:SmpIllegalArgumentException">
      <cmn:message>Sub profile validation failed.</
      cmn:message>
      <smpcmn:applicationErrors>
    </illegalArgumentException>
  </smpsa:executeOrderException>
  
```

The following are OSSJ request/response/exceptions are supported by SMP:

In schema XmlServiceActivationSchema.xsd

- createOrderByValue Request/Response/Exception

The OrderValue embedded in request can be Snapshot, ActionType or OSSJ order types defined in XmlSmpServiceActivationSchema.xsd. The key included in response returned from Order Manager is in type of OrderKeyType of this same schema.

Here is example of createOrderByValueRequest:

```
<sa:createOrderByValueRequest xmlns:smpsa="http://
www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema">
  <smp:orderValue
    xmlns:smp="http://www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema"
    xsi:type="smp:ActionOrderValue">
    <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
    <sa:apiClientId>web</sa:apiClientId>
    <sa:priority>3</sa:priority>
    ...
  </smp:orderValue>
</sa:createOrderByValueRequest>
```

Here is example of createOrderByValueResponse:

```
<sa:createOrderByValueResponse xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:cmn="http://java.sun.com/products/oss/xml/Common"
xmlns:smpsa="http://www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema">
  <sa:orderKey xsi:type="smpsa:OrderKeyType">
    <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
    <sa:primaryKey >20011</sa:primaryKey>
  </sa:orderKey>
</sa:createOrderByValueResponse>
```

- startOrderByKey Request/Response/Exception

The key included in the request is in type of OrderKeyType or its' super type OrderKey. The response does not include any detailed information.

```
<sa:startOrderByKeyRequest xmlns:sa="http://java.sun.com/
products/oss/xml/ServiceActivation" xmlns:cmn="http://
java.sun.com/products/oss/xml/Common" xmlns:smpsa="http://
/www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance">
```



```

    <sa:orderKey xsi:type="smpsa:OrderKeyType">
      <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
      <sa:primaryKey>20012</sa:primaryKey>
    </sa:orderKey>
  </sa:startOrderByKeyRequest>
  <sa:startOrderByKeyResponse xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation">
</sa:startOrderByKeyResponse>

```

- **getOrderByKey Request/Response/Exception**

The key included in a request is in type of OrderKeyType or its' super type OrderKey. The response returns back Snapshot, action type or OSSJ order defined in XmlSMPServcieActivationSchema.xsd.

```

<sa:getOrderByKeyRequest xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:cmn="http://java.sun.com/products/oss/xml/
Common" xmlns:smpsa="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sa:orderKey xsi:type="smpsa:OrderKeyType">
    <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
    <sa:primaryKey>20012</sa:primaryKey>
  </sa:orderKey>
</sa:getOrderByKeyRequest>
<sa:getOrderByValueResponse xmlns:smpsa="http://
www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema">
  <smp:orderValue
    xmlns:smp="http://www.sigma-systems.com/schemas/
3.1/SmpServiceActivationSchema"
    xsi:type="smp:ActionOrderValue">
    <cmn:lastUpdateVersionNumber>1</
cmn:lastUpdateVersionNumber>
    <sa:apiClientId>web</sa:apiClientId>
    <sa:priority>3</sa:priority>
    ...
  </smp:orderValue>
</sa: getOrderByValueResponse >

```

- **abortOrderByKey Request/Response/Exception**

The key included in the request is in type of OrderKeyType or its' super type OrderKey. The response does not include any concrete information.

```
<sa:startOrderByKeyRequest xmlns:sa="http://java.sun.com/products/oss/xml/ServiceActivation" xmlns:cmn="http://java.sun.com/products/oss/xml/Common" xmlns:smpsa="http://www.sigma-systems.com/schemas/3.1/SmpServiceActivationSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
  <sa:orderKey xsi:type="smpsa:OrderKeyType">
    <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</cmn:type>
    <sa:primaryKey>20012</sa:primaryKey>
  </sa:orderKey>
</sa:startOrderByKeyRequest>
```

```
<sa:startOrderByKeyResponse xmlns:sa="http://java.sun.com/products/oss/xml/ServiceActivation">
</sa:startOrderByKeyResponse>
```

- **makeOrderValue Request/Response/Exception**

For performance concerns, it is recommended that you do not use this request even though SMP supports it and it is defined in the OSSJ SA specification. XML Order client should be able to create order XML on the client side according to the schema definition.

The request passes an OrderType in string format. The response includes order value proto-type.

```
<sa:makeOrderValueRequest xmlns:sa="http://java.sun.com/products/oss/xml/ServiceActivation">
  <sa:orderType>SnapshotOrderValue</sa:orderType>
</sa:makeOrderValueRequest>
```

```
<sa:makeOrderValueResponse xmlns:sa="http://java.sun.com/products/oss/xml/ServiceActivation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cmn="http://java.sun.com/products/oss/xml/Common" xmlns:smpcmn="http://www.sigma-systems.com/schemas/3.1/SmpCommonValues">
```

```
  <sa:orderValue xmlns:smp="http://www.sigma-systems.com/schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:SnapshotOrderValue">
    <cmn:lastUpdateVersionNumber>-1</cmn:lastUpdateVersionNumber>
    <sa:orderKey xsi:type="smp:OrderKeyType">
      <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</cmn:type>
      <sa:primaryKey>-1</sa:primaryKey>
      <smp:externalKey>117e673184fadmin_svr26</smp:externalKey>
    </sa:orderKey>
    <sa:priority>3</sa:priority>
```

```

    <sa:orderState
      smpState="open.not_running.not_started"
      xsi:type="smp:SubOrderStateType">open.not_running.not_
      started</sa:orderState>
    <smp:orderParamList>
      <smpcmn:param name="has_groups">no</smpcmn:param>
    </smp:orderParamList>
  </sa:orderValue>
</sa:makeOrderValueResponse>

```

- **makeServiceValue Request/Response/Exception**

For performance concerns, it is recommended that you do not use this request even though SMP supports it and it is defined in the OSSJ SA specification. XML Order client should be able to create service xml on the client side according to the schema definition and service catalog.

The request passes the service name in string format. The response includes service value with default parameter values.

```

<sa:makeServiceValueRequest xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation">
  <sa:serviceType>SubSvcSpec:smp_email</
sa:serviceType>
</sa:makeServiceValueRequest>

<sa:makeServiceValueResponse xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:cmn="http://java.sun.com/
products/oss/xml/Common" xmlns:smpcbe="http://
www.sigma-systems.com/schemas/3.1/SmpCBECo
reSchema">
  <sa:serviceValue xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:SubSvcType">
    <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
    <sa:serviceKey xsi:type="smp:SubSvcKeyType">
      <cmn:type>SubSvcSpec:smp_email</cmn:type>
      <smp:externalKey>117ea440a14admin_svr1a</
smp:externalKey>
    </sa:serviceKey>
  <sa:serviceState provisionState="inactive"
xsi:type="smp:SubSvcStateType">inactive</
sa:serviceState>
  <smp:paramList>
    <smpcbe:param name="password" xsi:nil="true"/>
    <smpcbe:param name="user_name" xsi:nil="true"/>
    <smpcbe:param name="alpha_tag" xsi:nil="true"/>
  </smp:paramList>
</sa:makeServiceValueResponse>

```

```

        <smpcbe:param name="transfer_content">n</smpcbe:param>
        <smpcbe:param name="forward_for_days_for_migr" xsi:nil="true"/>
        <smpcbe:param name="domain_type" xsi:nil="true"/>
    </smp:paramList>
</sa:serviceValue>
</sa:makeServiceValueResponse>

```

- **queryOrders Request/Response/Exception**

The request sends a query name, condition attributes, and expected result attributes. All those definitions must appear in the NamedQuery definition. The response returns attribute <name, value> pair list.

```

<sa:queryOrdersRequest xmlns:sa="http://java.sun.com/products/oss/xml/ServiceActivation" xmlns:cmn="http://java.sun.com/products/oss/xml/Common" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sa:queryValue xmlns:smp="http://www.sigma-systems.com/schemas/3.1/SmpCommonValues" xsi:type="smp:SmpQueryValue">
    <smp:queryName>OrderQueryBySubId</smp:queryName>
    <smp:paramList>
      <smp:param name="smp_sub_key">10000</smp:param>
    </smp:paramList>
  </sa:queryValue>
  <sa:attributeNames>
    <cmn:item>managedEntityKey</cmn:item>
    <cmn:item>state</cmn:item>
    <cmn:item>order_type</cmn:item>
  </sa:attributeNames>
</sa:queryOrdersRequest>
<sa:queryOrdersResponse>
</sa:queryOrdersResponse>

```

- **setOrderByValue Request/Response/Exception**

The request sends a modified order value, which can be Snapshot, ActionType or OSSJ order types defined in XmlSmpServiceActivationSchema.xsd. The response does not contain any concrete information.

```

<sa:setOrderByValueRequest xmlns:smpsa="http://www.sigma-systems.com/schemas/3.1/SmpServiceActivationSchema">
  <smp:orderValue

```

```

        xmlns:smp="http://www.sigma-systems.com/schemas/
3.1/SmpServiceActivationSchema"
        xsi:type="smp:ActionOrderValue">
        <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
        <sa:apiClientId>web</sa:apiClientId>
        <sa:priority>3</sa:priority>
        ...
    </smp:orderValue>
</sa:setOrderByValueRequest>
<sa:setOrderByValueResponse xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation">
</sa:setOrderByValueResponse>

```

If you just want to send an order parameter change, you can also use **OrderParamChangeValue** as defined in **XmlSmpServiceActivationSchema.xsd**. Here is an example of changing the order parameter `start_datetime`:

```

<sa:setOrderByValueRequest xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cmn="http://java.sun.com/products/oss/xml/
Common" xmlns:smpcmn="http://www.sigma-systems.com/
schemas/3.1/SmpCommonValues">
    <sa:orderValue xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:OrderParamChangeValue">
        <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
        <sa:orderKey xsi:type="smp:OrderKeyType">
            <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
            <sa:primaryKey>10001</sa:primaryKey>
        </sa:orderKey>
        <smp:orderParamList>
            <smpcmn:param
name="start_datetime">20070308152700</smpcmn:param>
        </smp:orderParamList>
    </sa:orderValue>
    <sa:checkConcurrentAccess>false</
sa:checkConcurrentAccess>
</sa:setOrderByValueRequest>

```

In schema **XmlCommonSchema.xsd**

- **QueryManagedEntities Request/Response/Exception**

The request sends a query value which includes query name, condition attributes, and expected result attributes. All those definitions must appear in the NamedQuery definition. The response returns the attribute <name, value> pair list.

```
<cmn:queryManagedEntitiesRequest xmlns:cmn="http://
java.sun.com/products/oss/xml/Common"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <cmn:query xmlns:smp="http://www.sigma-systems.com/
schemas/3.1/SmpCommonValues"
xsi:type="smp:SmpQueryValue">
    <smp:queryName>nq_get_email_usage_bean</
smp:queryName>
    <smp:paramList>
      <smp:param name="domain">catv9.ne.jp</smp:param>
      <smp:param name="email_login">1</smp:param>
    </smp:paramList>
  </cmn:query>
  <cmn:attrNames>
    <cmn:item>usage_quota</cmn:item>
  </cmn:attrNames>
</cmn:queryManagedEntitiesRequest>

<cmn:queryManagedEntitiesResponse xmlns:cmn="http://
java.sun.com/products/oss/xml/Common"
xmlns:smpcbe="http://www.sigma-systems.com/schemas/
3.1/SmpCBECoreSchema" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance">
  <cmn:value>
    <cmn:item xsi:type="smpcbe:EntityType">
      <smpcbe:paramList>
        <smpcbe:param name="usage_quota">1050</smpcbe:param>
      </smpcbe:paramList>
    </cmn:item>
  </cmn:value>
</cmn:queryManagedEntitiesResponse>
```

SMP adds the following Request/Response/Exception:

In Schema XmlSmpServiceActivationSchema.xsd

- executeOrder Request/Response/Exception

The request sends an order value, which can be Snapshot, ActionType or OSSJ order types defined in XmlSmpServiceActivationSchema.xsd. The response includes a generated order key and order state.

```
<executeOrderRequest xmlns="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema">
```

```

        <smps:orderValue xmlns:smps="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cmn="http://java.sun.com/products/oss/xml/
Common" xmlns:sa="http://java.sun.com/products/oss/
xml/ServiceActivation" xmlns:smpcbe="http://www.sigma-
systems.com/schemas/3.1/SmpCBECORESchema"
xmlns:smpcmn="http://www.sigma-systems.com/schemas/
3.1/SmpCommonValues" xmlns:smpsvc="http://www.sigma-
systems.com/schemas/3.1/SmpCBEServiceSchema"
xmlns:cbevc="http://java.sun.com/products/oss/xml/
CBE/Service" xsi:type="smps:ActionOrderValue">

        <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>

        <sa:apiClientId>web</sa:apiClientId>

        <sa:orderKey xsi:type="smps:OrderKeyType">

        ...

        </smps:orderValue>
</executeOrderRequest>

<smps:executeOrderResponse xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:cmn="http://java.sun.com/products/oss/xml/
Common" xmlns:smps="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema">

<smps:orderKey xmlns:smps="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema">

        <cmn:type xmlns:cmn="http://java.sun.com/
products/oss/xml/Common">

                com.sigma.samp.cmn.order.SampOrderKey

        </cmn:type>

<sa:primaryKey xmlns:sa="http://java.sun.com/products/
oss/xml/ServiceActivation">

        20009

        </sa:primaryKey>

</smps:orderKey>

<smps:orderState

xmlns:smps="http://www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema"

        smpState="open.running.workflow">

        open.running

        </smps:orderState>

</smps:executeOrderResponse>

```

- **ExecuteOrderAsync Request/Response/Exception**

The request sends an order value, which can be Snapshot, ActionType or OSSJ order types defined in XmlSmpServiceActivationSchema.xsd. The response

includes a generated order key. State is not included in the result, since the order has not yet been executed.

```
<ExecuteOrderAsyncRequest xmlns="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema">
  <smpsa:orderValue xmlns:smpsa="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:cmn="http://
java.sun.com/products/oss/xml/Common" xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:smpcbe="http://www.sigma-systems.com/schemas/3.1/
SmpCBECoreSchema" xmlns:smpcmn="http://www.sigma-systems.com/
schemas/3.1/SmpCommonValues" xmlns:smpsvc="http://www.sigma-
systems.com/schemas/3.1/SmpCBEServiceSchema"
xmlns:cbesvc="http://java.sun.com/products/oss/xml/CBE/Service"
xsi:type="smpsa:ActionOrderValue">
    <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
    <sa:apiClientId>web</sa:apiClientId>
    <sa:orderKey xsi:type="smpsa:OrderKeyType">

<cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
    <sa:primaryKey>-1</sa:primaryKey>
    </sa:orderKey>
    <smpsa:subKey>
      <cmn:type>SubSpec:-</cmn:type>
      <smpcbe:externalKey>test20</smpcbe:externalKey>
    </smpsa:subKey>
    <smpsa:orderParamList>
      <smpcmn:param name="has_groups">yes</
smpcmn:param>
      <smpcmn:param name="queue_request_flag">yes</
smpcmn:param>
    </smpsa:orderParamList>
    <smpsa:orderItemList>
      ...
    </smpsa:orderItemList>
  </smpsa:orderValue>
</ExecuteOrderAsyncRequest>

<smpsa:ExecuteOrderAsyncResponse xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:cmn="http://java.sun.com/products/oss/xml/
Common" xmlns:smpsa="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema">
```



```

<smpsa:orderKey xmlns:smpsa="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema">
    <cmn:type xmlns:cmn="http://java.sun.com/
products/oss/xml/Common">
        com.sigma.samp.cmn.order.SampOrderKey
    </cmn:type>
    <sa:primaryKey xmlns:sa="http://java.sun.com/
products/oss/xml/ServiceActivation">
        20010
    </sa:primaryKey>
</smpsa:orderKey>
</smpsa:ExecuteOrderAsyncResponse>

```

- **replicateOrderByKey Request/Response/Exception**

The request includes the order key of an existing order. The response returns a cloned order.

```

<smpsa:replicateOrderByKeyRequest xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:cmn="http://java.sun.com/products/oss/xml/
Common" xmlns:smpsa="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <smpsa:orderKey xsi:type="smpsa:OrderKeyType">
        <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
        <sa:primaryKey>10000</sa:primaryKey>
    </smpsa:orderKey>
</smpsa:replicateOrderByKeyRequest>

<smpsa:replicateOrderByKeyResponse xmlns:smpsa="http://
www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:cmn="http://
java.sun.com/products/oss/xml/Common" xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:smpcbe="http://www.sigma-systems.com/schemas/
3.1/SmpCBECORESchema" xmlns:smpcmn="http://www.sigma-
systems.com/schemas/3.1/SmpCommonValues"
xmlns:cbesvc="http://java.sun.com/products/oss/xml/
CBE/Service">
    <smp:orderValue xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:ActionOrderValue">
        <cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
        ...
    </smp:orderValue>

```

```
</smps:replicateOrderByKeyResponse>
```

- **repairOrderByKey Request/Response/Exception**

Request includes key of order which is aborted or partially completed. Response sends back action type order for fixing aborted line item(s) in the original order.

```
<smps:repairOrderByKeyRequest xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:cmn="http://java.sun.com/products/oss/xml/
Common" xmlns:smps="http://www.sigma-systems.com/
schemas/3.1/SmpServiceActivationSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
<smps:orderKey xsi:type="smps:OrderKeyType">
```

```
<cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
```

```
<sa:primaryKey>10001</sa:primaryKey>
```

```
</smps:orderKey>
```

```
</smps:repairOrderByKeyRequest>
```

```
<smps:repairOrderByKeyResponse xmlns:smps="http://
www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:cmn="http://
java.sun.com/products/oss/xml/Common" xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:smpcbe="http://www.sigma-systems.com/schemas/
3.1/SmpCBECORESchema" xmlns:smpcmn="http://www.sigma-
systems.com/schemas/3.1/SmpCommonValues"
xmlns:cbevc="http://java.sun.com/products/oss/xml/
CBE/Service">
```

```
<smp:orderValue xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
xsi:type="smp:ActionOrderValue">
```

```
<cmn:lastUpdateVersionNumber>-1</
cmn:lastUpdateVersionNumber>
```

```
<sa:apiClientId>web</sa:apiClientId>
```

```
...
```

```
</smp:orderValue>
```

```
</smps:repairOrderByKeyResponse>
```

- **getServiceByKey Request/Response/Exception**

The `getServiceByKey` request passes keys in `SubSvcKeyType` as defined in `XmlSmpServiceActivationSchema.xsd`. The response returns values in `SubSvcType` as defined in the same schema.

```
<smps:getServiceByKeyRequest xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:"http://java.sun.com/products/oss/xml/Common"
xmlns:smps="http://www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance">
```

```
<smps:serviceKey>
```

```

        <cmn:type>SubSvcSpec:docsis_internet_access</cmn:type>
        <smpsa:primaryKey>10004</smpsa:primaryKey>
        <smpsa:externalKey>117ea373858admin_svr17</smpsa:externalKey>
    </smpsa:serviceKey>
</smpsa:getServiceByKeyRequest>
<smpsa:getServiceByKeyResponse xmlns:smpsa="http://www.sigma-systems.com/schemas/3.1/SmpServiceActivationSchema" xmlns:cmn="http://java.sun.com/products/oss/xml/Common" xmlns:sa="http://java.sun.com/products/xml/ServiceActivation" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:smpcbe="http://www.a-systems.com/schemas/3.1/SmpCBECoreSchema">
    <smpsa:serviceValue>
        <cmn:lastUpdateVersionNumber>2</cmn:lastUpdateVersionNumber>
        <sa:serviceKey xmlns:smp="http://www.sigma-systems.com/schemas/3.1/SmpServiceActivationSchema" xsie="smp:SubSvcKeyType">
            ...
        </sa:serviceKey>
    </smpsa:serviceValue>
</smpsa:getServiceByKeyResponse>

```

#### In schema XmlSMPCBECoreSchema.xsd

- **getEntityByKey Request/Response/Exception**

The request sends the entityKey, which may include the primary key, external key, or namedQuery key. The response returns the corresponding entity value. If the key is for subscriber service, the XML will be in SubSvcType as defined in XmlSmpCBEServiceSchema.xsd.

The getEntityByKeyRequest has an attribute “cascadeLoading”. If the value is true and the entity key is for subscriber, the subscriber value included in the response will include the whole subscriber (which means subscriber paramters, addresses, contacts, services, and so on). If the icascadeLoading value is false, the subscriber value only includes the subscriber parameters.

If the entityKey is for composed subscriber service, and icascadeLoading is true, the returned subscriber service will include the children services. Otherwise, Order Manager only returns the composed subscriber service with its status and service name.

Here is an example of getting subscriber service by its external key.

```

<smpcbe:getEntityByKeyRequest xmlns:smpcbe="http://www.sigma-systems.com/schemas/3.1/SmpCBECoreSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <smpcbe:entityKeyLst>

```

```

    <smpcbe:entityKey xmlns:cmn="http://java.sun.com/
products/oss/xml/Common"
xsi:type="smpcbe:EntityType">
        <cmn:type>SubSvcSpec:samp_ds_hsd_access</
cmn:type>
        <smpcbe:externalKey>117f4f0dac6admin_svr</
smpcbe:externalKey>
    </smpcbe:entityKey>
</smpcbe:entityKeyLst>
</smpcbe:getEntityByKeyRequest>
<smpcbe:getEntityByKeyResponse xmlns:smpcbe="http://
www.sigma-systems.com/schemas/3.1/SmpCBECoreSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cmn="http://java.sun.com/products/oss/xml/
Common" xmlns:cbesvc="http://java.sun.com/products/
oss/xml/CBE/Service">
    <smpcbe:entityValue xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpCBEServiceSchema"
xsi:type="smp:SubSvcType">
        <cmn:lastUpdateVersionNumber>1</
cmn:lastUpdateVersionNumber>
        <cbesvc:serviceState xmlns:smp="http://www.sigma-
systems.com/schemas/3.1/SmpServiceActivationSchema"
provisionState="active"
xsi:type="smp:SubSvcStateType">active</
cbesvc:serviceState>
        <smp:serviceKey>
            <cmn:type>SubSvcSpec:samp_ds_hsd_access</
cmn:type>
            <smp:primaryKey>10002</smp:primaryKey>
            <smp:externalKey>117f4f0dac6admin_svr</
smp:externalKey>
        </smp:serviceKey>
        <smp:paramList>
            <smpcbe:param name="rf_downstream_freq">999000</
smpcbe:param>
            <smpcbe:param name="cm_manufacturer"
xsi:nil="true"/>
        </smp:paramList>
        <smpcbe:param name="qlty_of_svc">RES</smpcbe:param>
        <smpcbe:param name="max_downstream_bw">2 MB</
smpcbe:param>
    </smp:entityValue>
</smpcbe:getEntityByKeyResponse>

```

```

        <smpcbe:associationType>service_on_address</smpcbe:associationType>
        <smp:zEndKey xmlns:smp="http://www.sigma-systems.com/schemas/3.1/SmpCBECORESchema" xsi:type="smp:SubAddressKeyType">
            <cmn:type>SubAddress</cmn:type>
            <smp:primaryKey>10000</smp:primaryKey>
        </smp:zEndKey>
        <smpcbe:changeAction>add</smpcbe:changeAction>
    </smpcbe:association>
</smp:associationList>
</smpcbe:entityValue>
</smpcbe:getEntityByKeyResponse>

```

- **MakeEntityValue Request/Response/Exception**

For performance concerns, it is recommended that you do not use this request, even though SMP supports it and it is defined in the OSSJ SA specification. XML Order client should be able to create entity xml on the client side according to the schema definition and service catalog.

The request sends the entity type. The response returns the entity prototype value. If the entity type is for service, the subscriber service XML will be returned in SubSvcType as defined in XmlSmpCBEServiceSchema.xsd.

Here is an example of making values for a subscriber address.

```

<smpcbe:MakeEntityValueRequest xmlns:smpcbe="http://www.sigma-systems.com/schemas/3.1/SmpCBECORESchema">
    <smpcbe:entityType>SubAddressSpec:-</smpcbe:entityType>
</smpcbe:MakeEntityValueRequest>

<smpcbe:MakeEntityValueResponse xmlns:smpcbe="http://www.sigma-systems.com/schemas/3.1/SmpCBECORESchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:cmn="http://java.sun.com/products/oss/xml/Common">
    <smp:entityValue xmlns:smp="http://www.sigma-systems.com/schemas/3.1/SmpCBECORESchema" addressType="" isDefault="false" xsi:type="smp:SubAddressType">
        <cmn:lastUpdateVersionNumber>-1</cmn:lastUpdateVersionNumber>
        <smp:key xsi:type="smp:SubAddressKeyType">
            <cmn:type>SubAddressSpec:-</cmn:type>
            <smp:externalKey>117f49cd4f2admin_svr</smp:externalKey>
        </smp:key>
    </smp:entityValue>
</smpcbe:MakeEntityValueResponse>

```

```

<smp:state>inactive</smp:state>
<smp:paramList>
  <smp:param name="street_nm" xsi:nil="true"/>
  <smp:param name="apt_num" xsi:nil="true"/>
  <smp:param name="street_dir" xsi:nil="true"/>
  <smp:param name="house_suffix_cd" xsi:nil="true"/
>
  <smp:param name="street_suffix" xsi:nil="true"/>
  <smp:param name="addr_seq" xsi:nil="true"/>
  <smp:param name="serviced_area" xsi:nil="true"/>
  <smp:param name="country">CAN</smp:param>
  <smp:param name="street_num" xsi:nil="true"/>
  <smp:param name="ntd_addr_seq" xsi:nil="true"/>
  <smp:param name="ntd_primary_nm" xsi:nil="true"/>
  <smp:param name="return_segment" xsi:nil="true"/>
  <smp:param name="prov" xsi:nil="true"/>
  <smp:param name="postal_cd" xsi:nil="true"/>
  <smp:param name="city" xsi:nil="true"/>
</smp:paramList>
</smp:entityValue>
</smpcbe:MakeEntityValueResponse>

```

## Notification

OrderManager supports order/lineItem state change notification and order message notification.

### State Change Notification

You can configure Order Manager to send out a notification when an order or line item has changed into a specific state, for example, the order is closed. Here is a portion of the configuration in `ordermgr_cfg.xml` taken from `smp_demo`. The setting asks Order Manager to send out a notification in XML to the JMS queue when the order is closed.

```

<notification_registries
event_type="com.sigma.samp.comn.order.SampOrderStateChange
Event">
  <registry>
    <notification_method
class="com.sigma.vframe.jnotify.impl.NotifyByQueue">
      <jndi_name>com.sigma.jms.order_notify_queue</
jndi_name>

```

```

<queue_connection_factory>com.sigma.jms.DefaultQueueConne
ctionFactory</queue_connection_factory>
  </notification_method>
  <filter
class="com.sigma.vframe.jnotify.impl.NotificationFilterImpl">
condition
    <condition expr="substr(status,0,6)=='closed'"/>
  </filter>
  <included_properties> event properties to be included in
notification
    <event_property name="orderType"/>
    <event_property name="status"/>
    <event_property name="err_reason"/>
  </included_properties>
  <message_format>xml</message_format> ask to send notification in
XML
  <join_existing_transaction>true</
join_existing_transaction>
</registry>
...
</notification_registries>

```

Order Manager sends out an order state change notification in `SmpOrderStateChangeEvent` as defined in `XmlSmpServiceActivationSchema.xsd`. The type definition is as follows:

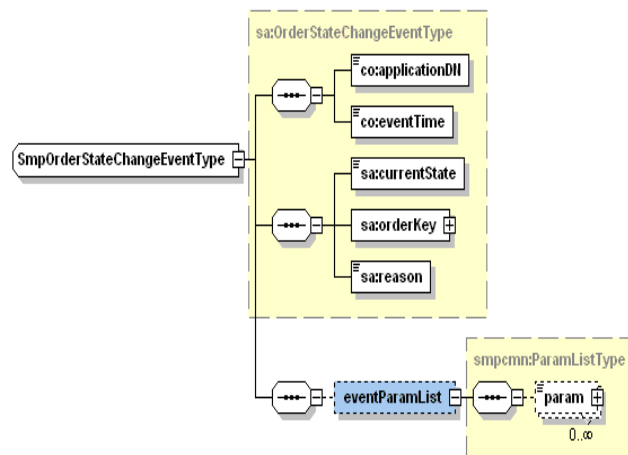


Figure 28: State Change Notification

Here is an example:

```

<sa:orderStateChangeEvent
xmlns:smpsa="http://www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema"

```

```

xmlns:sa="http://java.sun.com/products/oss/xml/
ServiceActivation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cmn="http://java.sun.com/products/oss/xml/Common"
xmlns:smpcmn="http://www.sigma-systems.com/schemas/3.1/
SmpCommonValues"
xsi:type="smpsa:SmpOrderStateChangeEvent"
    <cmn:applicationDN>-</cmn:applicationDN>
    <sa:currentState>closed.completed.all</
sa:currentState>
    <sa:orderKey xsi:type="smpsa:OrderKeyType">
        <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
        <sa:primaryKey>10903</sa:primaryKey>
    </sa:orderKey>
    <sa:reason>order processing completes</sa:reason>
    <smpsa:eventParamList>
    <smpcmn:param name="OSS_ORDER_TYPE">
        SnapshotOrderValue</smpcmn:param>
        <smpcmn:param name="orderType">SnapshotOrderValue</
smpcmn:param>
        <smpcmn:param name="err_reason"/>
    <smpcmn:param name="OSS_EVENT_TYPE">

com.sigma.samp.vframe.ordermodel.SampOrderStateChangeEventIm
pl
    </smpcmn:param>
    <smpcmn:param name="OSS_ORDER_CURRENT_STATE">
        closed.completed.all</smpcmn:param>
        <smpcmn:param name="status">closed.completed.all</
smpcmn:param>
    <smpcmn:param name="OSS_ORDER_SOURCE">super_system
    </smpcmn:param>
        <smpcmn:param name="OSS_ORDER_KEY">10903</smpcmn:param>
    </smpsa:eventParamList>
</sa:orderStateChangeEvent>

```

Event parameters, such as orderType, err\_reason, and status come from notifications. Other parameters starting with OSS\_ are parameters required by the OSSJ SA specification.

Here is the type definition for SmpOrderItemStateChangeEvent to notify a line item status change:



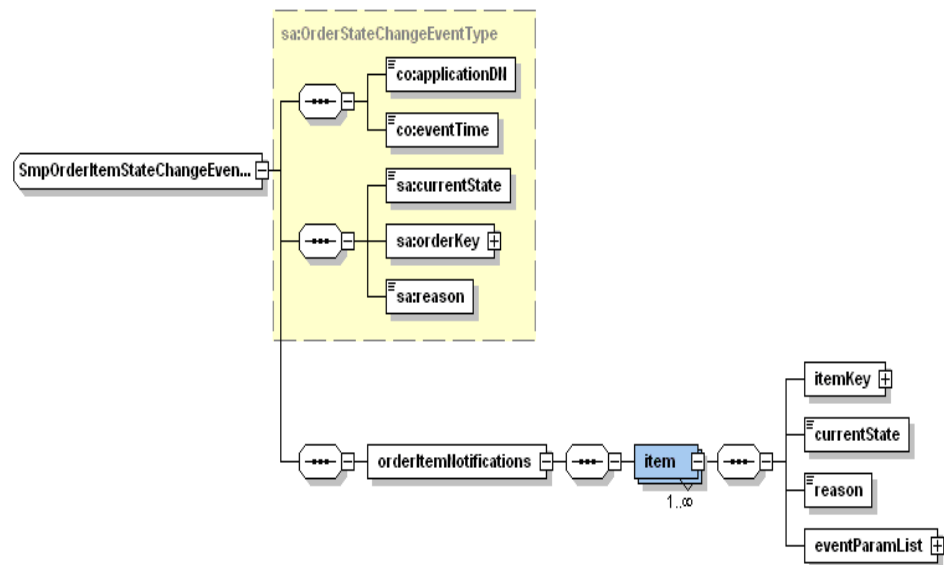


Figure 29: State Change Type Definition

For example:

```
<sa:orderStateChangeEvent
  xmlns:smpsa="http://www.sigma-systems.com/schemas/
3.1/SmpServiceActivationSchema"
  xmlns:sa="http://java.sun.com/products/oss/xml/
ServiceActivation"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:cmn="http://java.sun.com/products/oss/xml/
Common"
  xmlns:smpcmn="http://www.sigma-systems.com/schemas/
3.1/SmpCommonValues"
  xsi:type="smpsa:SmpOrderItemStateChangeEvent">
  <cmn:applicationDN>-</cmn:applicationDN>
  <sa:currentState>closed.completed.all</sa:currentState>
  <sa:orderKey xsi:type="smpsa:OrderKeyType">
    <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
    <sa:primaryKey>11603</sa:primaryKey>
  </sa:orderKey>
  <sa:reason>reply from workflow engine</sa:reason>
  <smpsa:orderItemNotifications>
    <smpsa:item>for each line item
    <smpsa:itemKey>
```

```

<cmn:type>com.sigma.samp.cmn.order.SampOrderLineItemValue
  </cmn:type>
  <sa:primaryKey>27902</sa:primaryKey>
</smpsa:itemKey>
  <smpsa:eventParamList>
    <smpcmn:param
name="status">closed.completed.enforced</smpcmn:param>
    <smpcmn:param name="notifySeqNum">1</
smpcmn:param>
    <smpcmn:param name="entityId">510150</
smpcmn:param>
  </smpsa:eventParamList>
</smpsa:item>
</smpsa:orderItemNotifications>
</sa:orderStateChangeEvent>

```

Each event can include notification for more than one LineItem. OSSJ doesn't have concept about line item, so there is not OSSJ parameter in this event.

## Message Notification

When workflow request has problem or gets stuck in workflow, it may call EA `smp_prcs_workflow_err` task to ask Order Manager sending out message type notification. The type of event is `SmpOrderMsgNotificationEventType` defined in `XmlSmpServiceActivationSchema.xsd`.

The definition is as shown below:

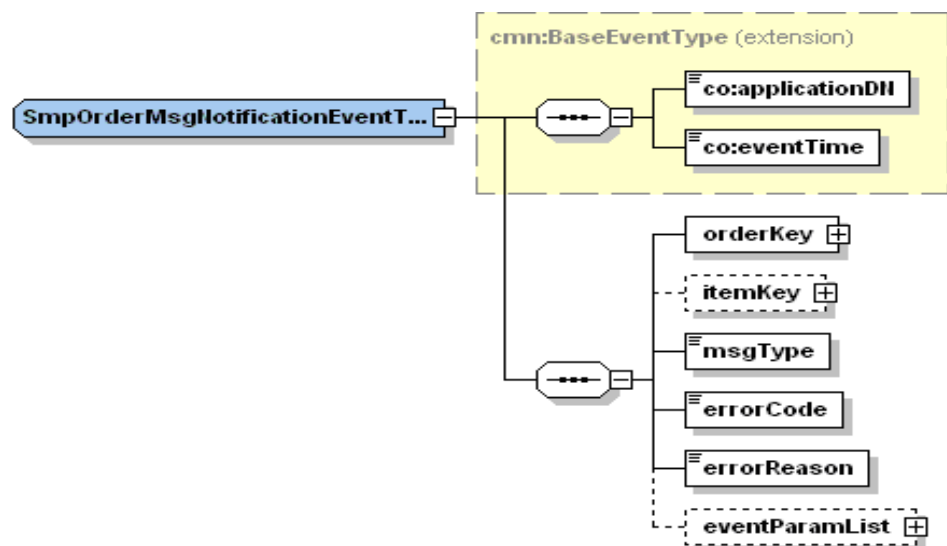


Figure 30: Mesage Notification

The following is a sample:

```

<smpsa:orderMsgNotificationEvent xmlns:smpsa="http://
www.sigma-systems.com/schemas/3.1/
SmpServiceActivationSchema" xmlns:cmn="http://
java.sun.com/products/oss/xml/Common" xmlns:sa="http://
java.sun.com/products/oss/xml/ServiceActivation"
xmlns:xsi="http://www.w3.or
g/2001/XMLSchema-instance" xmlns:smpcmn="http://
www.sigma-systems.com/schemas/3.1/SmpCommonValues">
  <smpsa:orderKey>
    <cmn:type>com.sigma.samp.cmn.order.SampOrderKey</
cmn:type>
    <sa:primaryKey>10500</sa:primaryKey>
  </smpsa:orderKey>
  <smpsa:itemKey>
    <cmn:type>com.sigma.samp.cmn.order.SampLineItemKey</
cmn:type>
    <sa:primaryKey>12334</sa:primaryKey>
  </smpsa:itemKey>
  <smpsa:msgType>ERROR</smpsa:msgType>
  <smpsa:errorCode xsi:nil="true"/>
  <smpsa:errorReason>failed</smpsa:errorReason>
  <smpsa:eventParamList>
    <smpcmn:param name="OSS_ORDER_TYPE">SvcActOrderValue</
smpcmn:param>
    <smpcmn:param name="smp_tech_platform">Nortel</
smpcmn:param>
    <smpcmn:param name="orderType">SvcActOrderValue</
smpcmn:param>
    <smpcmn:param name="smp_err_reason">failed</
smpcmn:param>
    <smpcmn:param
name="OSS_ORDER_CURRENT_STATE">open.running.workflow</
smpcmn:param>
    <smpcmn:param name="OSS_ORDER_SOURCE">web</
smpcmn:param>
    <smpcmn:param name="status">open.running.workflow</
smpcmn:param>
    <smpcmn:param name="smp_3rd_err_readon">null</
smpcmn:param>
    <smpcmn:param name="smp_sec_err_reason">null</
smpcmn:param>
    <smpcmn:param name="smp_err_code_of_platform">-2</
smpcmn:param>
    <smpcmn:param name="err_reason"/>

```

```

        <smpcmn:param
name="OSS_EVENT_TYPE">com.sigma.samp.vframe.ordermodel.Sa
mpOrderMsgNotificationEventImpl</smpcmn:param>
        <smpcmn:param name="OSS_ORDER_KEY">10500</
smpcmn:param>
    </smpsa:eventParamList>
</smpsa:orderMsgNotificationEvent>

```

Event paramaters comes from:

- OSSJ
- EA task smp\_prcs\_workflow\_err
- Event properties of notification registry in ordermgr\_cfg.xml

## Web Services samples in Order Manager SDK

The Order Manager SDK provides a Web Servcies sample under the directory sdk/src/com/sigma/samp/ordermgr/sdk/simpleclient.

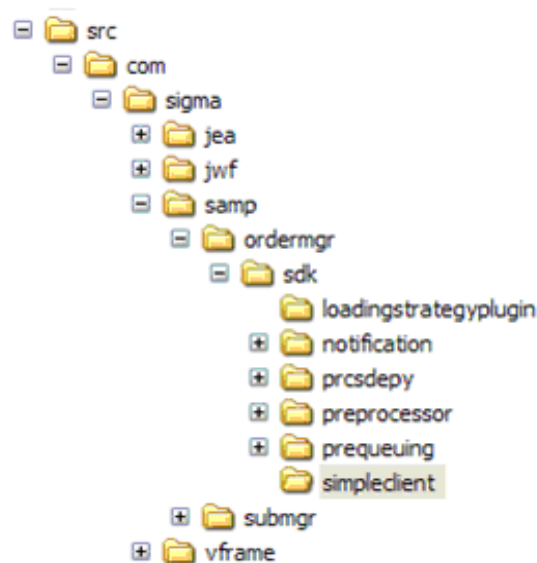


Figure 31: Directory Structure

The sample includes the following steps:

Steps	Request XML Name	Description
1. Create sub	create_sub.xml	executeOrderRequest with snapshot type order
2. add hsd service	add_hsd.xml	executeOrderRequest with action type order for adding hsd servcie

3. Get order by key	get_order_by_key.xml	getOrderByKeyRequest with order id key
4. Get sub by key	get_sub_by_key.xml	getEntityByKeyRequest with namedQuery(by account number) key for subscriber and cascadeLoading="true"
5. Get sub entity by key	get_entity_by_key.xml	getEntityByKeyRequest with namedQuery(by account number) key for subscriber.
6. Get service by key	get_service_by_key.xml	getServiceByKeyRequest with namedQuery(by mac address) key.
7. execute order async	update_svc.xml	ExecuteOrderAsyncRequest with svc_provider_nm change on samp_ds_hsd_access service identified by external key
8. Create order by value	update_svc_using_name_query_key.xml	createOrderByValueRequest with iis_primaryi parameter change on samp_ds_hsd_access service identified by namedQuery key.
9. set order by value	update_svc_using_name_query_key_with_future_date.xml	setOrderByValueRequest with order parameter "start_datetime" change Order is now become a future date order.
10. Start order by key	start_order.xml	startOrderByKeyRequest with order id key. Order will be pre-queued, since it is a future date order.
11. Abort order by key	abort_order.xml	abortOrderByKeyRequest with order id key. Order is closed and aborted.
12. Sub namedQuery	query_entity.xml	queryManagedEntitiesRequest with sub search by account id

To run the test, see the "TestWebServicesClient" section in package.html or readme.txt for details.

WebServicesClient.java from the SDK defines the method for each SMP supported request. It can be reused in your Web Services client if it is written in Java.



# Appendix A - Order Parameters

Parameter Name	Type	Description
ordre_type	String ReadOnly	Can be used in order notification configuration. Values are: -SnapshotOrderValue - ReprovOrderValue - SvcActOrderValue - ReprovisionSubOrderValue - com.sigma.samp.cmn.order.BatchOrderValue - javax.oss.order.CreateOrderValue
order_prov_type	String ReadOnly	Can only be used in order notification configuration. Values are: -provision for snapshot,action,ossj order with sub. -peprovision.stm for order generated by STM project -reprovision.sub for sub reprocessing request -event For actuations order without sub
has_groups	“yes” or “no” (default) yes: line item level commit/rollback no: order level commit/rollback	
impact_generation_policy	"skip_tmplt"	To be used when you don't want the impact for this order.
order_request	String ReadOnly for client	Can only be used in getOrderByKey request or order notification configuration.
queue_request_flag	“yes” or “no” Default is “yes”	If don't want order to be queued, need to add this parameter with “no”. Order manager will throw exception instead of queuing the order if order has to be queued (or pre-queued)
batch_ordr_id	Long	Point to batch order. Only needed for order belonged to batch.

external_batch_id	String	External id of batch order
hold_ordr_flag	'y' or 'n' y: if wants OrderManager to hold the order n: to release the holding	See pre-queuing session for detail
start_datetime	Datetime Format like: YYMMDDHHM ISS Example: 080201203938 For Feb. 01 2008 20:39:38	Specify order execution time. See pre-queuing session for detail.
prereq_ordr_id	Long	Specify the prerequisite order. See pre-queuing session for detail.
original_ordr_id	Long ReadOnly for client	Order manager automatically assigns original order id to this parameter in generated repairing order.
resolved_indicator	'y' or 'n' ReadOnly for client	When an order (for example A) is aborted or partially completed, order manager automatically assign it with resolved_indicator='n'.  When order manager receives the repairing order, will automatically modify the resolved_indicator for order A to be 'y'.
ordr_owner	String	Owner of order, input by order client and used by SMP UI. The value must be a valid SMP AM user name.



# Appendix B - Xbeans Library Examples

You can construct the XML request using the Xbeans library. An XML Bean compiles an XML Schema and generates a jar file “SmpXmlOrderApi.jar”. You can use this library to construct the XML request if Client is written in Java.

The following jar files are required to be shown in the classpath. They are available in the \$DOMAIN\_HOME/extjars directory or extjars under the following SDK package:

- SmpXmlOrderApi.jar
- xbean.jar
- jsr173\_1.0\_api.jar

## Constructing XML Objects

---

You can use this generated library to construct XML objects and eventually get the XML request.

### *Entity XmlObject*

Each type defined in an XML XSD has a corresponding Java interface derived from `org.apache.xmlbeans.XmlObject`.

The important `XmlObject(s)` for entity XML construction are as follows:

- Subscriber `XmlObject`
- SubKey `XmlObject(s)`
- Servcie `XmlObject(s)`
- subEntity `XmlObject(s)`

### Packages

Import the following packages in the Java code:

- `com.sigma.samp.schemas.xmlorderapi.cbesvc`
- `com.sigma.samp.schemas.xmlorderapi.cbecore`
- `com.sigma.samp.schemas.xmlorderapi.cmn`

### `makeSubSvc()`

The Java type for CBE service is `com.sigma.samp.schemas.xmlorderapi.cbesvc.SubSvcType`.

An example to construct a CBE service is as follows:

```
SubSvcType respSvc = SubSvcType.Factory.newInstance();
    //populate subSvcKey
    SubSvcKeyType key = makeSubSvcKey();
    respSvc.setServiceKey(key);
    //populate parent subSvcKey
    SubSvcKeyType parentSubSvcKey== makeSubSvcKey ();
    respSvc.setParentServiceKey(parentSubSvcKey);
    //populate subSvc status
    //you only need to do it if you want to change subSvc
    status
    SubSvcStateType state =
    SubSvcStateType.Factory.newInstance();
    String status=îactiveî;
    state.set (status);
    respSvc.xsetServiceState(state);
    //for updating subSvc's parm, only need to populate the
    parm you want to change
    //for creating new subSvc, only need to provide parm which
    value is different from the
    //default value
    EntityParamListType parmLst = makeParmList ();
    if (parmLst.sizeOfParamArray() > 0) {
        respSvc.setParamList(parmLst);
    }
    //this step is required for new subSvc
    //for updating subSvc's association, please skip this step
    and create line item for association
    AssocListType assocLst = MakeAssocList();
    if (assocLst.sizeOfAssociationArray() > 0) {
        respSvc.setAssociationList(assocLst);
    }
```

## MakeSubSvcKey()

The Java type for CBE service is

com.sigma.samp.schemas.xmlorderapi.cbescvc.SubSvcKeyType.

There are three kinds of keys:

- ID type key
- External type key
- Name query key

**Example for ID key:**

```
SubSvcKeyType xkey
    =SubSvcKeyType.Factory.newInstance();
int id=1213;
xkey.setPrimaryKey(id);
String subSvcType=îSvc:primary_emailî;
xkey.setType(subSvcType);
```

**Example for external key:**

```
SubSvcKeyType xkey
    =SubSvcKeyType.Factory.newInstance();
String externalKey=îmy_serviceî;
xkey.setExternalKey(externalKey);
String subSvcType=îSvc:primary_emailî;
xkey.setType(subSvcType);
```

**Example for name query key:**

```
SubSvcKeyType xkey =SubSvcKeyType.Factory.newInstance();
NamedQuery query=xkey.addNewNameQuery();
String queryName=îyou query nameî;
query.setQueryName(queryName);
ParamListType list=query.addNewInputParamList();
for(every query parm) {
    ParamType parm=list.addNewParam();
    String parmName=îquery parm nameî;
    parm.setName(parmName);
    String parmValue="query parm value";
    parm.setStringValue(parmValue);
}
String subSvcType="Svc:primary_email";
xkey.setType(subSvcType);
```

**makeParmList ()**

The Java type for parameter list is

`com.sigma.samp.schemas.xmlorderapi.cmn.makeParmList()` can be used to make subSvc or make entity such as address or contact.

**Example for making a parameter list:**

```
EntityParamListType paramList =
EntityParamListType.Factory.newInstance();
for (every parm ) {
    ParamType xparm = paramList.addNewParam();
```

```

        String parmNm="my parm name";
        xparm.setName(parmNm);
        /String parmValue = "my parm value";
        xparm.setStringValue(parmVal);
    }

```

## MakeAssocList()

Association was a new concept introduced in SMP3.1. The service address, service preReq and parm alias now belong to the association category.

Example for creating an association list is as follows:

```

AssocListType assocLst =
AssocListType.Factory.newInstance();
ArrayList assocs = new ArrayList();
for (every association) {
    assocs.add(makeAssoc());
}
if (assocs.isEmpty() == false) {
    assocLst.setAssociationArray( (AssocType[])
assocs.toArray(new AssocType[
    0]));
}

```

## MakeAssoc()

```

AssocType xAssoc=null;
if (is a parm alias) {
    xAssoc = ParmAssocType.Factory.newInstance();
    ( (ParmAssocType) xAssoc).setAParamName(iyour entity
    parm name);
    ( (ParmAssocType) xAssoc).setZParamName(ipreReq parm
    name);
    assocs.add(xAssoc);
}
else {
    xAssoc= AssocType.Factory.newInstance();
}
EntityKeyType zEntityKey=makeEntityKey();
xAssoc.setZEndKey(zEntityKey);

```

## MakeEntity()

A subscriber entity includes address, contact, user, etc. The Java type for entity is `com.sigma.samp.schemas.xmlorderapi.cbcore.EntityType`.

**Example code for making entity:**

```

EntityType xEntity = EntityType.Factory.newInstance();
xEntity.setKey(makeEntityKey());
//only need if you want to change status, otherwise,
can skip this step
String entityState="inactive";
xEntity.setState(entityState);
//for updating entity's parm, only need to populate the
parm you want to change
//for creating new entity, only need to provide parm
which value is different from the
//default value
EntityParamListType parmLst = makeParmList();
if (parmLst.sizeOfParamArray() > 0) {
    xEntity.setParamList(parmLst);
}
//this step is required for new entity
//for updating entity's association, please skip this
step and create line item
//for association
AssocListType assocLst = makeAssocList();
if (assocLst.sizeOfAssociationArray() > 0) {
    xEntity.setAssociationList(assocLst);
}

```

**makeEntityKey()**

The java type for EntityKey is

com.sigma.samp.schemas.xmlorderapi.cbcore.EntityKeyType.

There are three kinds of keys:

- ID type key
- External type key
- Name query key

Example for ID key:

```

EntityKeyType
xkey=EntityKeyType.Factory.newInstance();
//populate the entity type
//for example it is address
xkey.setType(iSubAddress:-i);
xkey.setPrimarykey(123123);

```

Example for external key:

```

EntityKeyType
xkey=EntityKeyType.Factory.newInstance();
String externalKey=îmy_serviceî;
xkey.setExternalKey(externalKey);
xkey.setType("SubAddress:-");

```

Example for name query key:

```

EntityKeyType
xkey=EntityKeyType.Factory.newInstance();
NamedQuery query=xkey.addNewNameQuery();
String queryName="you query name";
query.setQueryName(queryName);
ParamListType list=query.addNewInputParamList();
for(every query parm) {
    ParamType parm=list.addNewParam();
    String parmName="query parm name";
    parm.setName(parmName);
    String parmValue="query parm value";
    parm.setStringValue(parmValue);
}
xkey.setType("SubAddress:-");

```

## MakeSubscriber()

The Java class for subscriber is

```
com.sigma.samp.schemas.xmlorderapi.cbcore.SubType.
```

The subscriber is derived from the sub entity. The code to make subscriber is as follows:

```

EntityType xEntity = SubType.Factory.newInstance()
    xEntity.setKey(makeEntityKey());
    xEntity.setServiceProvider("service provider
name");
    xEntity.setSubscriberType("subscriber type, for
example residential");
    xEntity.setLocale("locale, for example en_US";
//only needed when create new subscriber with
services and subEntities
    Collection entityLst=new ArrayList();
    if (for every subSvc or subEntity) {
        if (it is a subSvc) {
            entityLst.add(makeSubSvc());
        }
    }

```

```

        else
            entityLst.add(makeEntity());
        }
        if (entityLst.isEmpty() == false) {
            EntityListType xEntityList =
            xsub.addNewEntityList();
            xEntityList.setEntityValueArray( (EntityValue[])
            entityLst.toArray(new
                EntityValue[0]));
            xsub.setEntityList(xEntityList);
        }
        //only needed if want to change status, otherwise, can
        skip this step
        String entityState="active";
        xEntity.setState(entityState);
        //for updating entity's parm, only need to populate the
        parm you want to change
        //for creating new entity, only need to provide parm which
        value is different from the
        //default value
        EntityParamListType parmLst = makeParmList();
        if (parmLst.sizeOfParamArray() > 0) {
            xEntity.setParamList(parmLst);
        }
    }

```

## MakeSubKey()

SubKey is derived from the entity key. makeSubKey(). The logic is same as makeEntityKey(). The only difference is type. subKey is an instance of the SubKeyType.

```

SubKeyType xkey=SubKeyType.Factory.newInstance()
(same as mekEntityKey())

```

## order xml

The SmpXmlOrderApi.jar also includes XML Object for the following types of orders:

- snapshot order
- entity action order
- ossj order
- reprovisioning order

## package

Order XmlObjects are defined in package  
com.sigma.samp.schemas.xmlorderapi.sa

## Snapshot order XmlObject

The following is a code example to construct the XML Object of the snapshot order.

```
SnapshotOrderValue
xorder=SnapshotOrderValue.Factory.newInstance();
    xorder.setApiClientId(iapi client id, for example
    smpwebi);
    xorder.setDescription(iorder descriptioni);
    //populate other order parm
    ParamListType xparamLst = makeParamList();
xOrder.setOrderParamList(xparamLst);
SubType xsub=makeSubscriber();
xOrder.setSubscriber(xsub);
```

## Entity action order XmlObject

```
SnapshotOrderValue
xorder=ActionOrderValue.Factory.newInstance()
    xorder.setApiClientId("api client id , for example
    smpweb");
    xorder.setDescription(iorder descriptioni);
    //populate other order parm
    ParamListType xparamLst = makeParamList();
    xOrder.setOrderParamList(xparamLst);
    SubKeyType subKey=makeSubKey();
    xOrder.setSubKey(subKey);
    ActionOrderValue.OrderItemList xitemLst =
    xorder.addNewOrderItemList();
    for (every line item) {
    OrderItemType xitem= xitemLst.addNewOrderItem();
    xitem.setAction(ifor example add/delete/updatei);
    //populate line item parm
    ParamListType xparams = xitem.addNewItemParamList();
    for (every line item parm) {
        ParamType xparm = xparams.addNewParam();
        xparm.setParmNm(iparm namei);
        xparm.setStringValue(iparm valuei);
```



```

    }
    //only need to populate entity key
    // if don't want to provide entity value
    // for example delete an entity
    xitem.setEntityKey(makeEntityKey());
    switch(TheEntityType) {
        case sub_svc:
            xitem.setEntityValue(makesubSvc());
            break;
        case sub_entity:
            xitem.setEntityValue(makeEntity());
            break;
        case sub:
            xitem.setEntityValue(makesub());
            break;
        case association:
            xitem.setEntityVlaue(makeAssoc());
            break;
    }
}

```

### Order XML

Now, we have order XmlObject. To get the XML, you need:

```

SampOrderDocument orderDoc =
SampOrderDocument.Factory.newInstance();
OrderValue orderXmlObject = snapshot order value or action
order value;
orderDoc.addNewSampOrder().setOrderValue(orderXmlObject);
String xmlStr = orderXmlObject.xmlText();
You can also convert an XML string to order XML object:
    SampOrderDocument orderDoc =
    SampOrderDocument.Factory.parse(xml);
    OrderValue orderXmlObject =
    orderDoc.getSampOrder().getOrderValue();

```

### Order Request

OrderManager only accepts XML request, not the order XML.

Some sample code for creating ExecuteOrder XML request and sending it to orderManager are as follows:

```
ExecuteOrderRequestDocument
```

```
req=ExecuteOrderRequestDocument.Factory.newInstance();  
req.addNewExecuteOrderRequest().setOrderValue(xorder);  
    String xmlReq=req.xmlText();  
    String  
xmlResponse=getJvtActBean().executeXmlRequest(xml);
```

**In the same way, you can create other XML order request, such as  
CreateOrderByValue or ExecuteOrderAsync.**