HP Service Provisioner

User's and System Integrator's Guide

Edition: V70-1A

for Microsoft Windows® Server 2012 R2, HP-UX 11i v3, and Red Hat Enterprise Linux 6.6 operating systems

Manufacturing Part Number: None

January 21, 2015

Legal Notices

Warranty.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices.

Trademark Notices.

Java™ is a registered trademark of Oracle and/or its affiliates.

Linux is a U.S. registered trademark of Linus Torvalds

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

Red Hat® Enterprise Linux® is a registered trademark of Red Hat, Inc.

JBoss® is a registered trademark of Red Hat, Inc. in the United States and other countries

EnterpriseDB® is a registered trademark of EnterpriseDB.

Postgres Plus® Advanced Server is a registered trademark of EnterpriseDB.

Oracle® is a registered trademark of Oracle and/or its affiliates.

UNIX® is a registered trademark of the Open Group.

Windows® and MS Windows® are U.S. registered trademarks of Microsoft Corporation.

Document id: HPSA p158-pd026302

Install Location Descriptors

The following names are used to define install locations throughout this guide.

Descriptor	What the Descriptor Represents
$ACTIVATOR_OPT	The base install location of HP Service Provisioner and HP Service Activator. The UNIX® location is `/opt/OV/ServiceActivator` The Windows® location is `<install drive>:\HP\OpenView\ServiceActivator`
$ACTIVATOR_ETC	The install location of specific HP Service Provisioner and HP Service Activator files. The UNIX location is `/etc/opt/OV/ServiceActivator` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\etc`
$ACTIVATOR_VAR	The install location of specific HP Service Activator files. The UNIX location is `/var/opt/OV/ServiceActivator` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\var`
$ACTIVATOR_BIN	The install location of specific HP Service Activator files. The UNIX location is `/opt/OV/ServiceActivator/bin` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\bin`
$JBOSS_HOME	The install location for JBoss. The UNIX location is `/opt/HP/jboss` The Windows location is `<install drive>:\HP\jboss`
$JBOSS_DEPLOY	The install location of the HP Service Activator JEE components. The UNIX location is `/opt/HP/jboss/standalone/deployments` The Windows location is `<install drive>:\HP\jboss\standalone\deployments`
$JBOSS_EAR_LIB	The location for libraries (Java `*.jar` files) to be executed by the HP Service Activator engine (Workflow Manager and Resource Manager). The UNIX location is `/opt/HP/jboss/standalone/deployments/hpsa.ear/lib` The Windows location is `<install drive>:\HP\jboss\standalone\deployments\hpsa.ear\lib`

Conventions

The following typographical conventions are used in this guide.

Font	What the Font Represents	Example
Italic	Book or manual titles, and manpage names	Refer to the document *HP Service Activator—Workflows and the Workflow Manager* and the *Javadocs* for more information
	Provides emphasis	You *must* follow these steps.
`Computer`	Text and items on the computer screen	The system replies: `Press Enter`
	Command names	Use the `InventoryBuilder` command
	Method names	The `get_all_replies()` method does the following…
	File and directory names	Edit the file `$ACTIVATOR_ETC/config/mwfm.xml`
`Computer Bold`	Text that you must type	At the prompt, type: **`ls –l`**

In This Guide

This guide provides a general description of HP's Service Provisioner product as well as the detailed information needed

- to define products and services in the technical service catalog and maintain the definitions

and

- to monitor and interact with the HP Service Provisioner runtime engine as service orders are executed, in particular when order processing requires manual action.

The guide has the following chapters, which address different groups within the entire audience:

- **Introduction to HP Service Provisioner:** This chapter contains a complete general description of the HP Service Provisioner product from a function perspective. Read it to gain a general understanding of the product, for example to assess it for a potential application, or as an introduction before reading one or more of the following specific chapters in order to learn to use the product.

- **Fulfillment Processes:** This contains detailed information about the fulfillment processes supported by HP Service Provisioner including information about CFS/RFS processing order and rollback.

- **Implementation Architecture:** This chapter describes how HP Service Provisioner is implemented on the combined platform of HP Service Activator and HP Subscription Repository, interworking also with Trueview Inventory.

- **Installation:** This chapter contains information about how to install and configure HP Service Provisioner. A description of how to localize HP Service Provisioner is also included.

- **Client Integration:** This chapter is aimed at system integrators and describes the northbound API of the HP Service Provisioner, facing the client system, i.e. CRM or Order Management System which will inject service order requests into HP Service Provisioner.

- **Editing the Service Catalog:** This chapter, aimed at system integrators and technical product managers, describes how to edit the contents of the catalog from the HP Service Activator user interface. It is relevant for the system integrator who will implement an initial catalog for a solution, and for the technical product manager who will maintain it.

- **Monitoring and Interacting with Running Orders:** This chapter is the specific guide for (runtime) operators of HP Service Provisioner solutions with screenshots.

- **Processes for Resource Facing Services:** This chapter, aimed at system integrators, explains how to implement process functionality for Resource Facing Service in the form of HP Service Activator workflows.

- **Delta Operations:** This chapter describes the new and powerful delta operations supported in HP Service Provisioner 7.0. With delta operations, it is possible to modify existing services by specifying the desired end result; Service Provisioner will then calculate how transform the existing service into the new service.

- **Dependencies:** This chapter, aimed at system integrators, explains how to use dependencies. With dependencies it is possible for product instances the act as resources that can be consumed by other product instances, CFSs, and RFSs.

- **Scheduled Requests:** This chapter describes how to make use of scheduled requests in HP Service Provisioner. With scheduled requests it is possible to schedule one or more requests to be “fired” in the future. One-off as well as repeating scheduled requests are supported.

- **Workflow Manager Modules and Node Library:** This chapter describes how to configure and use all workflow manager modules and workflow nodes that are included in the HP Service Provisioner software.

Audience

The audience for this guide includes all groups who need information about the HP Service Provisioner product:

- CSP solution architects who will evaluate the product

- Systems integrators who will plan and deliver service operations factories, including architects as well as developers of catalog contents and fulfillment processes

- CSP technical product managers who will maintain product specifications in the service catalog

- CSP operators who will work with the service operations factory at runtime: monitor it and interact with running service orders

Document References

The following documentation will also be relevant, depending on the role of the reader:

- *HP Service Activator, System Integrator’s Overview*

- *HP Service Activator, User’s and Administrator’s Guide*

- *HP Service Activator, Workflows and the Workflow Manager*

- *HP Subscription Repository v7.0, User Guide*

- *HP Subscription Repository v7.0, Developer Guide*

- *HP Subscription Repository v7.0, Installation and Administration Reference*

- *HP Trueview 2.1, Install Guide*

- *HP Trueview 2.1, Admin Guide*

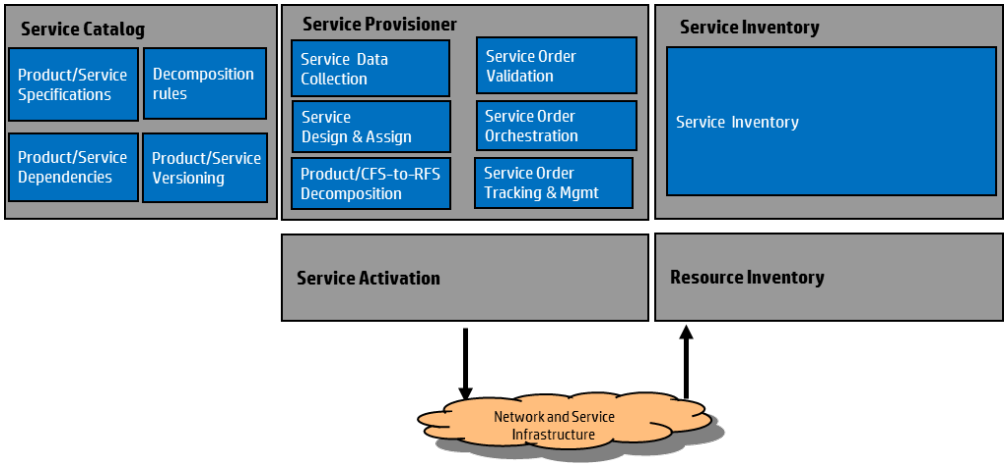
- *HP Trueview 2.1, Inventory User Guide*

1 Introduction to HP Service Provisioner

HP OSS Fulfillment, see Figure 1, is HP's implementation of the operational part of a telecommunications factory for customer on-boarding. It supports the approach of a Service Operations Factory concept to avoid technology silos. It includes:

- Catalog-driven Service Order Management to manage orders within the factory. It stores product instances created by orders in a service inventory.
- Service Provisioner to map ordered products to Customer Facing Services and Resource Facing Service, and further onto the network capabilities. Complex planning and provisioning processes are managed through assign and design.
- Service Activation to orchestrate execution of commands towards network and IT infrastructure.
- Trueview Inventory to provide additional functions like planning, inside plant, discovery, and data accuracy.

Figure 1 **HP OSS Fulfillment**

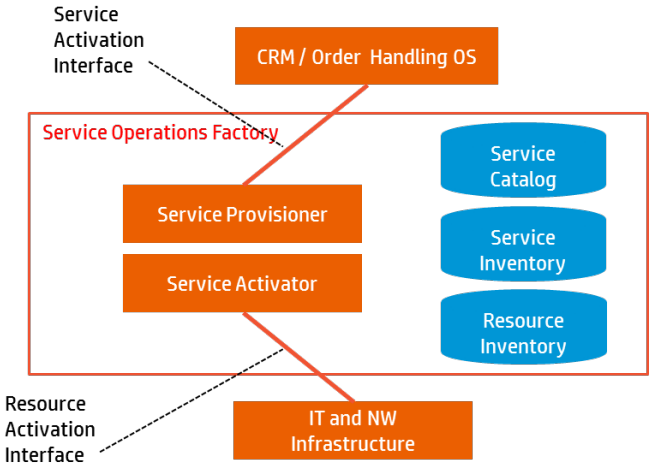


HP Service Provisioner is a framework that supports both development and deployment of a broad range of technical order management solutions. It applies to the space of converged services delivered through combinations of IP network and IT processing infrastructures. It supports unified service operations factories spanning multiple technologies and thus avoids technology silos. Such factories can be set up for all situations where SID product structuring applies, i.e.

whenever all orderable products are specified by combining customer facing services which again comprise resource facing services delivered by the converged infrastructure.

The positioning of Service Provisioner as part of HP OSS Fulfillment is shown in Figure 2. As the figure shows, the factory contains two architecturally separate processing components: HP Service Provisioner and HP Service Activator, and three data stores: the service catalog, the service inventory, and the resource inventory. HP Service Provisioner interfaces to all three data stores and is closely integrated with HP Service Activator.

Figure 2 **HP Service Provisioner Positioning in HP OSS Fulfillment**



Being a framework, HP Service Provisioner does not contain the complete solution for any specific products and services. It contains the tools to develop and maintain such solutions, and it contains the runtime engine for the solutions.

In terms of its technical implementation HP Service Provisioner is built on top of HP Service Activator software which in a similar fashion contains tools and a runtime engine. The HP Service Provisioner runtime engine is an extension of the HP Service Activator runtime engine.

The process to build and maintain specific solutions is one main topic of this document. The other main topic is how operators interact with such solutions.

The northbound interface of HP Service Provisioner (the Service Activation Interface in Figure 2) is a web service interface implemented as SOAP and REST over HTTP. Over this interface HP Service Provisioner receives requests to process service orders. All requests are quickly responded to when they have been checked for syntactic correctness. This synchronous response does not convey the result of the request. That will be sent asynchronously over a message service that will typically be JMS. For more details about interactions on the interface including the contents of the request messages refer to Chapter 5.

Standards

HP Service Provisioner complies with applicable TMF Frameworx standards, the most important ones being:

- SID, which defines the concepts of products and services, instances as well as their specifications, with a hierarchical containment structure, along with the concept of (product/service) characteristics, whose values can be invariant over all instances of a given product/service, i.e. defined by the specifications, or may be specific to each instance (variant).

- MTOSI, with its business agreement documented in TMF 518, which specifies:

    1. the northbound interface of a HP Service Provisioner system (the Service Activation Interface in Figure 2) that allows a client system (CRM/commercial order management system in the BSS space) to request the creation of service orders, i.e. effectively product instances, and state transitions to be performed on them, and
    2. the data model used in exchanges across that interface, in particular the state model for service orders to be processed by a HP Service Provisioner system

- TAM, the Telecom Applications Map. HP Service Provisioner and solutions built on it cover Service Catalog, Service Inventory and most of Service Order Management as defined by TAM: Service Data Collection, Service Design/Assign, Service Order Validation and Service Order Orchestration.

- eTOM, the map of CSP processes. HP Service Provisioner covers a large part of the L2 Service Configuration and Activation.

Catalog Driven

HP Service Provisioner is catalog driven. That is where the SID concepts come in. In SID terms the catalog contains the specifications of the products and services that will be supported by a solution. SID structuring is hierarchical. Taking a top-down view, products contain and can be decomposed into services. In a bottom-up view, resource facing services can be combined into customer facing services which are again combined into products.

The HP Service Provisioner catalog is technical, not commercial. It is not concerned with commercial aspects of product offerings such as availability of products to particular categories of customers or with their prices, such as will be needed to support the customer-facing selling process, marketing campaigns, etc. Those aspects belong to commercial product catalogs, which are BSS systems, out of scope for HP Service Provisioner. The HP Service Provisioner catalog contains detailed technical information on how the services which constitute the substance of converged telco/IT products are fulfilled and deployed.

Catalog content may be created as part of solution development and may be delivered by a system integrator as part of solution delivery and support. The user interface to define and maintain the catalog content is uncomplicated to use, once the concepts of the catalog are mastered. Hence it is expected that new services will be added to the catalog and existing ones maintained with new versions by the CSP’s technical product managers.

The highest technical complexity lies in the area of process implementation. Processes for assigning resources, provisioning, and activation are implemented as workflows using the Workflow Designer tool from HP Service Activator. Workflow development requires programming skills. Once they have been developed, workflows can easily be tied to specifications of resource facing services in the catalog.

Service Specifications and Service Instances

The catalog of HP Service Provisioner contains specifications of products and services. Every technically distinct product to be offered and all the services needed as components of the products will have their technical specifications in the catalog.

The concepts underpinning the catalog structure are standardized in SID. In particular the concepts of products and services, with services falling into the categories of customer facing (CFS) and resource facing (RFS) all specified in terms of their characteristics, come from SID. So does the hierarchical structure: a product can be specified in terms of simpler products (in a commercial product catalog), recursively, and eventually of customer facing services. Customer facing services may again be specified in terms of simpler services: simpler customer facing services, also

recursively, implying a multi-level hierarchy, and resource facing services. Resource facing services are those that use resources in the provider's shared infrastructure.

SID clearly distinguishes between the specifications of products and services and instances of them. We will maintain the distinction by using the abbreviations PS for product specification, and CFSS/RFSS for specifications of CFS and RFS, respectively. PSs, CFSSs and RFSSs all belong in the catalog. Data describing instances of products and services (CFS as well as RFS) are managed in the service inventory, which is architecturally clearly separate from the catalog. The catalog is maintained by technical product managers as part of the product offering life-cycle. The service inventory is maintained by the running HP Service Provisioner solution as part of the service instance life-cycle, i.e. as part of the processing of service orders.

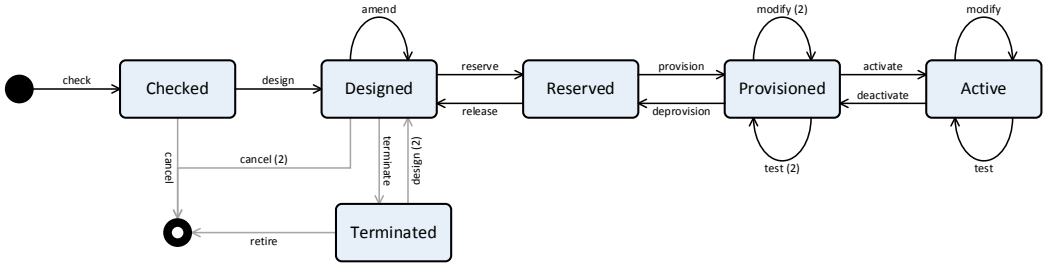
NOTE — HP Service Provisioner 7.0 has introduced a new object called an Engineering Work Item (EWI). Functionally, EWIs are similar to RFSs, but their intended use is to serve as a placeholder for a special (manual) piece of work that needs to be carried out between two RFSs or CFSs.

Service Orders and Product Instances

All the runtime processing capability of HP Service Provisioner is related to service orders and product instances. The way requests are made on the northbound interface is compliant with TMF 518. The NBI allows a client system to create and process service orders; once a service order has resulted in the creation of a product instance, further requests to change its state can be made by reference to the product instance by its service id.

The basis for service order processing requests is a state model. The default life-cycle[1] of a service order and the resulting product instance goes through the states shown in Figure 3, starting at the top left (filled circle) and ending when the service order has been cancelled or the resulting product instance has been retired at the lower left (circle). The arrows between the states indicate the possible *state transitions*, not to be confused with the possible *operations*. The grey arrows indicate state transitions for which it is *not* possible to associate action workflows (see the Section "Action Workflows" on page 21).

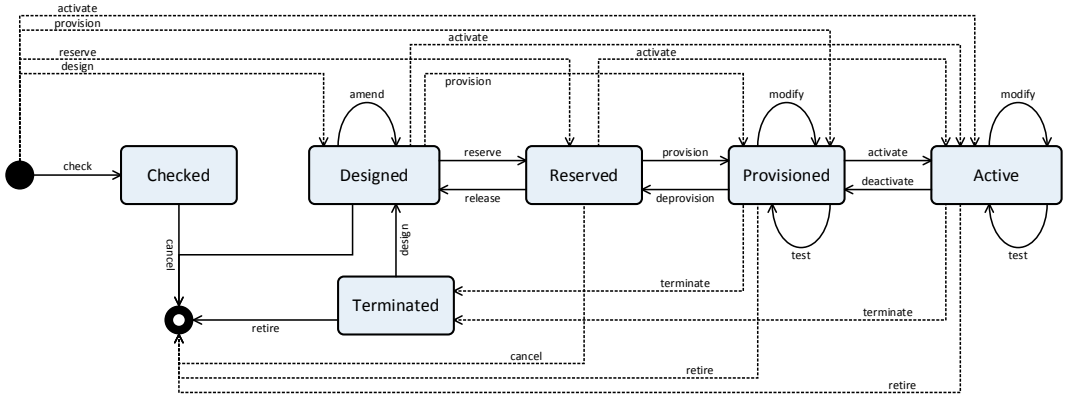
Figure 3 **Life-Cycle States (default) for Product Instances and the Possible State Transitions**



The default life-cycle states and the supported *operations* are shown in Figure 4. All operations are comprised of one or more state transitions (except for the special *noop* operation which does not contain any state transitions). Operations that contain a single state transition are illustrated with solid arrows whereas operations comprised of multiple state transitions are illustrated with dashed arrows.

[1] Other life-cycle profiles are support as of HP Service Provisioner version 7.0; see the Section "Life-Cycle Profiles" on page 29 for more details.

Figure 4 **Life-Cycle States (default) for Product Instances and the Supported Operations**



Requests supported on the NBI will contain a so-called *request type* which is mapped to an operation that, in turn, performs one or more state transitions on service orders/product instances. All the processes that are managed and executed by HP Service Provisioner implement such state transitions. More details about state transition processes are given in the next section.

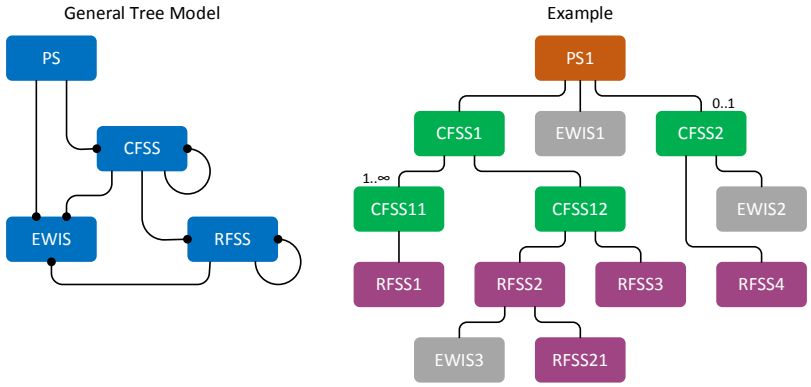
A request to create a new service order can work in two different modes depending on whether or not the northbound system is CFS-aware:

1. The request may reference a product specification which is then looked up in the service catalog by HP Service Provisioner.
2. The request may reference a product specification and *all* its constituent CFSs (including nested CFSs). HP Service Provisioner will look up the PS in the service catalog, verify that the CFSs match the CFSSs in the catalog, and add RFSs and EWIs to the correct positions in the tree. Please read the Chapter “Delta Operations” on page 71 for more details.

A product instance (PI) will be created based on the PS, and the PI will be the root of a tree whose branches are CFSs, RFSs, and EWIs, and whose leaves are RFSs and EWIs.

The tree structure of a PS in the catalog is illustrated in Figure 5, which shows the general model of the specification tree and an example. For CFSSs it is possible in the catalog to specify the minimum and maximum number of occurrences, including “unlimited” (see the Section “Variable Cardinality CFSs” on page 71). To make use of variable cardinality CFSs the northbound system needs to be CFS-aware. Upon receiving a service request, HP Service Provisioner will create a product instance as a tree structure mimicking the specification structure in the catalog.

Figure 5 **Catalog Tree Structure**



LEGEND	The bullet at the end of a line means that multiple occurrences of that object can be connected to one object at the other end. This indicates the branching structure of the tree. In the example CFSS11 and CFSS12 are both children of CFSS1 and CFSS11 may occur between 1 and an infinite number of times. Also, in this example CFSS2 is optional. RFSSs and EWISs always have the minimum and maximum occurrences set to 1 (i.e. they are always mandatory)

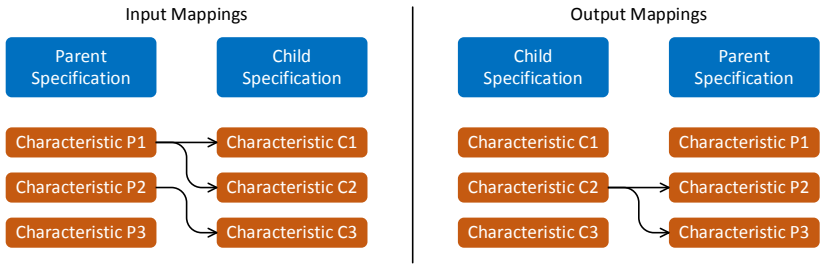
When the service order is received, a product instance, which is a tree with a structure similar to that of the specification, consisting of service instances, will be created by and persisted inside Service Provisioner (as a tree with a product instance root and a number of service instance objects as branches and leaves). In addition, Service Provisioner will generate a service id which can be used to reference the product instance.

Subsequent requests for state transitions will then reference the product instance at the root of the tree by its service id. For each product specification specified in the catalog, generally a large number of instances will be created over the time during which the product is available.

Every CFSS defines a CFS (or multiple CFSs in the case of a variable number of occurrences) as consisting of (decomposable into) a number of child services, which can be CFSs or RFSs, by reference to their specifications. Also each product or service specification (PS, CFSS, or RFSS) defines a set of characteristics for the service. Characteristic is the term used by SID. Synonyms like “attributes” or “parameters” may be more familiar. The characteristics and their values are the meat of a product or service instance. A PS, CFSS, or RFSS will also define a mapping between the characteristics of the specification itself and those of the children. The mapping goes two ways: There is an input mapping assigning values to the characteristics of the children from the values of parent characteristics, and an output mapping going the opposite way.

An example of input and output mappings is shown in Figure 6. The parent has 3 characteristics, P1, P2, and P3. One child is shown; it also has 3 characteristics, C1, C2, and C3. In this example, the input mapping assigns values to all three child characteristics, of which C1 and C2 takes their values from the same parent characteristic, P1. For the output mappings, the child characteristic C2 is mapped to the two parent characteristics, P2 and P3.

Figure 6 **Characteristic Input and Output Mappings**



Characteristics may be *invariant*, i.e. with values defined once for all instances in the catalog. A collection of characteristics (including types, description, restrictions, and default values) may be defined as a *profile*. Profile characteristics are used to define a set or subset of properties of a service offering that may be shared among multiple service offerings. A service offering may refer to a number of such profiles.

In addition, a collection of characteristic *values* can be defined as a *template*. For example, bandwidth and quality of service parameter definitions for different classes of service that could be given user-friendly names such as ‘platinum’, ‘standard’ or ‘gamer’ may be gathered and hidden behind the user-friendly name, in a template for a subscriber broadband access service.

Other, variant, characteristics may differ from instance to instance, typically information which needs to be specified per customer such as the customer’s address.

Note that even when a value is specified for a characteristic in the catalog it may be overruled by a parameter in a service request, possibly through an input mapping from a parent service. In such a case the defined value is not invariant, but serves only as a default that may be overridden.

A product characteristic can be defined to be either “constant”, “set-once”, or “modifiable”:

- **constant:** The characteristic will always have its default value and it may not be overwritten by a characteristic value from the service request.
- **set-once:** The characteristic can be set once upon creation of the product instance; either from its defined default or from a characteristic value in a service request. Once set, the characteristic may not be overwritten.
- **modifiable:** The characteristic can be set and updated at any time; typically this happens in the event of a *modify* operation.

2 Fulfillment Processes

This chapter contains a walkthrough of the fulfillment processes supported by HP Service Provisioner, and concepts are introduced that are important to understand in order to use HP Service Provisioner as intended.

States

The possible states of service orders and product instances were introduced in the section "Service Orders and Product Instances" on page 14. The intended use of each of the states in the default life-cycle profile is as follows:

Checked — It has been checked that it is technically feasible to build the services comprising the product given the requested characteristics such as site address, etc. The *Checked* state is a so-called transient state which means that this cannot be the permanent state of a product instance. If this is the final state of an operation the product instance will be deleted when the *Checked* state is reached.

Designed — The ordered product has been decomposed according to the tree structure defined in the service catalog and the resulting service tree persisted as an instance in service inventory. When a product is in this state it is possible to manually interact with the product if the "user interaction" flag has been set to "true".

Reserved — The shared infrastructure resources needed for the product (actually, for its RFSs) have been reserved in resource inventory and associated with the instance in service inventory.

Provisioned — All network elements and other infrastructure resources which are affected according to the design of the service have been configured to perform the service. This implies all needed logical resources – such as bandwidth, numbers/identifiers, registry entries, storage, etc. – have been allocated in the infrastructure. The service has not been turned active.

Active — The service is active in the infrastructure and usable subject to correct functioning of the infrastructure as monitored by assurance systems.

Terminated — All resources reserved for the product (its RFSs) have been released and may be reused for new services. The product remains persisted in Service Provisioner's service inventory.

NOTE — Additional – and simpler – life-cycle profiles will be introduced in the Section "Life-Cycle Profiles" on page 29.

Note in particular that the processes commonly referred to as "design and assign" are implied by the states *Designed* and *Reserved*. With HP Service Provisioner, the "design" needed for a product is the decomposition into its constituent services as controlled by the catalog, and "assign" means selecting and associating specific resources in the shared infrastructure with RFSs as done by the action workflows which are associated with its RFSSs in the catalog (workflows may also be associated with PSs, CFSs, and EWIs). Typically, with HP Service Provisioner, the design and

assign processes will be fully automated. See the Section “Manual Design and Assign” below for a description of how to overrule the automation and allow manual interaction.

Operations

Service orders are managed by requesting operations which are mapped to a number of state transitions. For example it is possible to make a single service request to create a new product instance and make it active; this will cause a total of five state transitions: to *Checked*, to *Designed*, to *Reserved*, to *Provisioned*, to *Active*. In such cases each transition in the sequence is completely processed, by traversal of the product tree structure, before the next one is undertaken.

Service requests may initiate the following *operations* for the default life-cycle profile (see Figure 4 on page 15):

check	From non-existing to Checked; this request will not persist the product instance in service inventory, the service order is only cached for a period of time; hence the operation may be repeated. The purpose is only to retrieve the information whether the transition is feasible.
design	From non-existing to Designed, from Checked to Designed, or from Terminated to Designed.
reserve	From non-existing to Reserved or from Designed to Reserved.
cancel	From Checked to non-existing, from Designed to non-existing, or from Reserved to non-existing.
provision	From non-existing to Provisioned, from Designed to Provisioned, or from Reserved to Provisioned
activate	From non-existing to Active, from Designed to Active, from Reserved to Active of from Provisioned to Active.
deactivate	From Active to Provisioned
deprovision	From Provisioned to Reserved
release	From Reserved to Designed
terminate	From Provisioned to Terminated or from Active to Terminated.
retire	From Terminated to non-existing, from Provisioned to non-existing, or from Active to non-existing.
amend	From Designed to Designed.
modify	From Provisioned to Provisioned or from Active to Active.
test	From Provisioned to Provisioned or from Active to Active.
noop	“No operation”; does not result in any state transitions.

Every state transition on a product (order) involves a process to be executed under control of the HP Service Provisioner process engine. The process traverses the complete tree of the product instance in a depth-first fashion.

NOTE The depth-first tree traversal can be in either “forward” or “backward” direction depending on the current state transition. For more details read the Section “Processing Direction” on page 22.

Action Workflows

In the catalog it is possible to associate PSs, CFSs, RFSs, and EWIs with HP Service Activator workflows that carry out specific actions for each transition between two states; such workflows will be referred to as *action workflows*.

There are two ways of specifying action workflows for catalog items; either the same action workflow can be specified to *all* possible state transitions (in this case the action workflow must include all the required actions for *all* state transitions) *or* different action workflows may be specified for each possible state transition (including no action workflow). Which of the two options are preferable is entirely up to the system integrator. For all non-leaf items in the catalog it is possible to define pre-workflows as well as post-workflows, and for leaf items only a single action workflow can be defined.

RFS state transition action workflows play an important role because they will typically implement the logic needed to interact with the network equipment and/or the resource inventory system. Each time an RFS action workflow is run as part of the process for a particular state transition, that transition is identified by parameters which must be taken into account by the workflow.

An example showing some typical actions to include in RFS action workflows is shown below (for selected state transitions):

non-existing → Checked	Check availability of access device and cabling to support connection to the customer site.
Designed → Reserved	Based on relevant characteristics of the RFS, such as customer site address, bandwidth required, etc., select appropriate shared resources and reserve them for use by the RFS.
Reserved → Provisioned	Through interaction with network elements, possibly working through element managers, configure all needed network elements appropriately to perform the service.
Provisioned → Active	Configure network elements to allow traffic to flow on connections that have been provisioned.
Active → Provisioned	Configure network elements to disallow traffic to flow on connections that have been provisioned.
Provisioned → Reserved	Undo configuration of network elements (inverse of configuring them for the service).
Reserved → Designed	Remove recorded relationships between service inventory and resource inventory and release resources in resource inventory

State Transitions

The product instance tree consist of leafs as well as non-leaf branches. The non-leaf branches of the tree are the root (product instance), the CFS instance objects, and (optionally) some of the RFS instance objects. The leaf branches are (typically, most of) the RFSs and the EWIs.

Each non-leaf branch has a set of children at the next level of decomposition. The process for any non-leaf branch has the following steps for each state transition:

1. Execute an action workflow (if a *pre-workflow* has been defined in the catalog).
2. For each child branch, do the following (the children are processed in the order defined for the branch; for backward transitions – e.g. from *Provisioned* to *Terminated* or from right to left in Figure 3 – the children are processed in inverse order):
    a. Perform the input mappings defined for the child branch, characteristics for which no mapping is defined are not affected.

b. In the context of the child branch, continue from step 1 (i.e. all branches are traversed in a depth-first fashion in a recursive fashion)

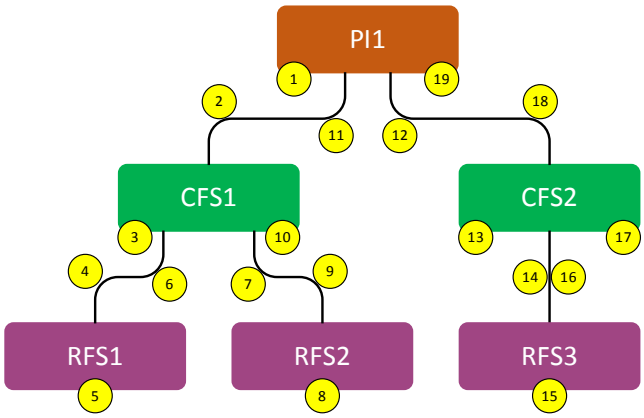
c. Perform the output mappings defined for the child branch, characteristics for which no mapping is defined are not affected.

3. Execute an action workflow (if a *post-workflow* has been defined in the catalog). For leafs, this step is skipped.

For further description of input/output characteristics mappings, see the Section “Product Specification” on page 51.

An example may help to understand the process. Figure 7 shows a product instance tree with only a single layer of CFSs, a single layer of RFSs, and no EWIs. The actions performed during one state transition process are shown as tings/circles with numbers indicating their sequence. Actions number 1, 3, and 13 are executions of pre-workflows; actions number 2, 4, 7, 12 and 14 are input mappings; actions number 6, 9, 11, 16, and 18 are output mappings, actions number 5, 8, and 15 are executions of RFS state transition action workflows, finally actions number 10, 17, and 19 are executions of post-workflows.

Figure 7 **Tree Traversal Process for a State Transition**



Processing Direction

When performing a specific state transition, the direction in which the CFSs and RFSs in the product tree structure are processed depends on the start and end states; the possible directions are “forward” and “backward”.

The tree traversal shown in Figure 7 is an example of a “forward” processing direction; this could, for instance, be a state transition from Provisioned to Active. The processing order in this example is:

```
PI1 →
     CFS1 →
            RFS1 →
            RFS2 →
     CFS2 →
            RFS3 →
```

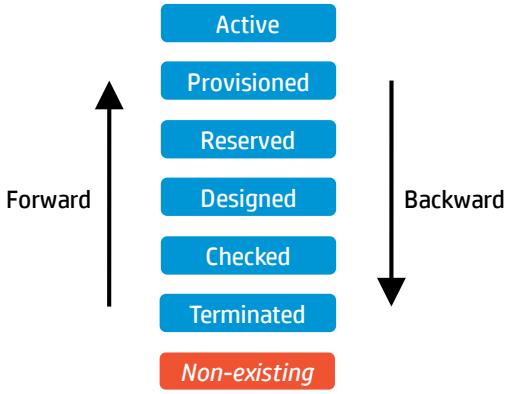
In the case of “backward” processing direction the processing order of the example tree shown in Figure 6 will be as follows:

```
PI1 →
      CFS2 →
            RFS3 →
      CFS1 →
            RFS2 →
            RFS1 →
```

In order to define an unambiguous processing direction for all possible state transition, the possible states (as well as a “non-existing” state, i.e. the state of a product before it has been checked or after it has been retired) are ordered by their so-called “maturity” levels. State transitions that move towards a higher maturity level will enable “forward” processing direction whereas state transitions that move towards a lower maturity will enable “backward” processing direction.

Figure 8 shows all states ordered by their maturity levels, with the highest maturity level listed at the top and the lowest maturity level at the bottom. The two arrows indicate the processing direction when traversing from one state to another.

Figure 8 **Maturity Levels and Processing Direction**



Error Handling

An attempted state transition may fail in the processing of an action workflow. When such a failure occurs, the behavior of the HP Service Provisioner engine will depend on the selected error-handling strategy which can be one of the following:

- **Atomic:** Service Provisioner will attempt to perform a full rollback, i.e. to the state the product instance was in before executing the latest service request. This is the default error handling strategy.
- **Best-effort:** Service Provisioner will set a “failed” flag for the instance that failed and then continue in a best-effort manner.
- **Partial:** Service Provisioner will attempt to roll back to the previous “stable” state; e.g. if an error happens in the state transition between *Reserved* and *Provisioned*, an attempt will be made to roll back to the *Reserved* state (this was the error-handling behavior in HP Service Provisioner 1.0).

Rollback

If the error-handling strategy is set to either “Atomic” or “Partial” a *consistent* error in an action workflow will trigger a rollback operation. During rollback, the actions that have been successfully completed until the point of failure will be processed once again in “reverse” mode; this means that if, for instance, a “reserve” action workflow for an RFS has been executed with success before the point of failure, then the “release” action workflow (for the same RFS) will be called during rollback.

IMPORTANT

A *consistent* error (result code = 1) in an action workflow means that the workflow was able to clean up its own changes and bring the affected service back to the state is was in before the action workflow was invoked. An *inconsistent* error (result code = 2) means that a failing action workflow was *not* able to bring the affected service back to the state it was in before the action workflow was invoked. In case of inconsistent errors, Service Provisioner will halt processing and *neither* attempt to do a roll back *nor* to roll forward. Therefore, solutions should be carefully designed in such a way that inconsistent errors are a rare event.

Generally speaking, for all state transitions that are allowed to do changes outside of HP Service Provisioner (i.e. in the resource inventory or in the network) *and* that can bring a product instance from one state to another there is also an “inverse” state transition (see Figure 3 for the “default” life-cycle profile). Table 1 shows the mappings between state transitions and inverse state transitions for the default life-cycle profile.

Table 1 **Mapping between State Transitions and Inverse State Transition**

State Transition Name	Inverse State Transition Name
Reserve	Release
Provision	Deprovision
Activate	Deactivate
Release	Reserve
Deprovision	Provision
Deactivate	Activate

As an example, consider the instance tree illustrated in Figure 7. If an error occurs during an “activate” state transition (i.e. from “Provisioned” to “Active”) while processing the action workflow for “RFS3”, the process will be as follows:

```
PI1 (pre-WF, activate) →
      CFS1 (pre-WF, activate) →
            RFS1 (WF, activate) →
            RFS2 (WF, activate) →
      CFS1 (post-WF, activate) →
      CFS2 (pre-WF, activate) →
            RFS3 (WF, activate) → ERROR (result = 1)
      CFS2 (post-WF, deactivate) ⮌
      CFS1 (pre-WF, deactivate) ⮌
            RFS2 (WF, deactivate) ⮌
            RFS1 (WF, deactivate) ⮌
      CFS1 (post-WF, deactivate) ⮌
PI1 (post-WF, deactivate) ⮌
```

LEGEND

The → symbol indicates a “roll forward” operation and the ⮌ symbol indicates a “rollback” operation.

IMPORTANT	The failing RFS action workflow ("RFS3" in the example shown above) will not be called again during rollback. Hence, in case of errors in the action workflow it must attempt to revert all its actions before terminating.

There may be situations where rollback fails. If this happens the product instance will be left in an inconsistent state; i.e. the states of the CFSs, RFSs, and EWIs in the product instance will not be identical. An example of a rollback failure is illustrated below:

```
PI1 (pre-WF, activate) →
      CFS1 (pre-WF, activate) →
            RFS1 (WF, activate) →
            RFS2 (WF, activate) →
      CFS1 (post-WF, activate) →
      CFS2 (pre-WF, activate) →
            RFS3 (WF, activate) → ERROR (result = 1)
      CFS2 (post-WF, deactivate) ↩
      CFS1 (pre-WF, deactivate) ↩
            RFS2 (WF, deactivate) ↩ ERROR (result = 1)
```

In this example the "RFS2" workflow fails when the "deactivate" transition is executed and this brings the rollback operation to a halt. This means that manual action will be required to clean up the changes caused by the actions workflows for "RFS2" and "RFS 1" as well as the changes caused by the action pre-workflows for "CFS1" and "PI1".

Finally, there can be cases where rollback is even not attempted. This happens if an RFS workflow fails with a "result" code larger than 1, which indicates that the RFS workflow was not able to clean up its own changes. An example of this is shown here:

```
PI1 (pre-WF, activate) →
      CFS1 (pre-WF, activate) →
            RFS1 (WF, activate) →
            RFS2 (WF, activate) →
      CFS1 (post-WF, activate) →
      CFS2 (pre-WF, activate) →
            RFS3 (WF, activate) ↯ ERROR (result = 2)
```

Also in this case, manual actions will be required bring the state of the product instance back into a consistent state.

Modify Operations

An existing product can be modified as a result of an incoming service request. Two types of modifications are supported:

- **Modifications of characteristic values:** One or more characteristic values may be modified by sending a service request to HP Service Provisioner containing new characteristic values. The new characteristic values will be applied to the product instance (the "root") before any state transitions are commenced and may (or may not) ripple down from the product instance to child CFSs, RFSs, and EWIs through characteristic input mappings. An example of a scenario where characteristic modification can be used could be the modification of the upstream and downstream bandwidths of an existing DSL service.

- **Structural modifications:** CFSs may be added to and/or deleted from an existing product instance through a service request containing “modify-add” and/or “modify-delete” actions. If a service request contains multiple “modify-add”/”modify-delete” actions, they will be added/deleted using the order in which they are specified in the service request. Structural modifications are only possible for operations that explicitly support modifications. For the default life-cycle profile (see Figure 3 and Figure 4) these operations are “modify” (*Active→Active* or *Provisioned→Provisioned*) and “amend” (*Designed→Designed*). An example of a “modify-add” scenario could be the addition of a VoIP service to an existing DSL service.

Structural Modify

As described above, there are two types of structural modifications that may either add or delete CFSs. The position in which to add a CFS or from which to delete a CFS is identified using a so-called “path”. The “path” is in essence a space-separated list of index values that uniquely identifies a position within a product instance tree; the “path” of the root element in a product instance tree is an empty string. An example of a product instance tree with all paths (in brackets) is shown below:

```
PI1 []
     CFS1 [0]
            RFS1 [0 0]
            RFS2 [0 1]
     CFS2 [1]
            RFS3 [1 0]
     CFS3 [2]
            RFS4 [2 0]
            RFS5 [2 1]
```

To add or delete a CFS the service request must provide the following information:

- **Add CFS:** The service request must contain the name and version of the CFSS to add as well the path. In addition, the following optional information may be provided: a CFS label, a template name/version from which to get characteristics values, a list of characteristic values to assign to the CFS’s characteristics, and lists of input/output mappings for this CFS. The CFS instance to be added is created from the CFS specification; i.e. the CFS instance will include child CFSs, child RFSs, and/or child EWIs. Before the new CFS instance is added to the product instance, the CFS itself and all its children will traverse all states from *Initial* until reaching the same state as the other instances. Hence, if a CFS is added to a product instance in state *Provisioned*, it will traverse the states *Initial→Checked→Designed→Reserved→Provisioned* (action workflows will be invoked, if any have been defined for these transitions).

- **Delete CFS:** The service request must contain a “path” that uniquely identifies the position of the CFS to be deleted. Before a CFS instance is deleted from the product instance the CFS itself and all its children will traverse all states from its current state until reaching the *Retired* state. Hence, if a CFS is deleted from a product instance in state *Provisioned*, it will traverse the states *Provisioned→Reserved→Designed→Terminated→Retired* (action workflows will be invoked if any have been defined for these transitions).

NOTE

Using “path” to identify the position of a CFS to be added or deleted may be inconvenient in some cases because it may be considered to be too technical. A much more powerful way to add or delete CFSs is using *Delta Operations* which is described in Chapter 9 on page 71.

After the completion of a “modify-add” or “modify-delete” operation (or a combination of multiple “modify-add” and “modify-delete” operations), the entire product instance will traverse the “modify” or “amend” state transition (depending on the current state of the product instance).

An example of a “modify-add” operation is shown below. Consider the following product instance:

```
PI1 []
     CFS1 [0]
            RFS1 [0 0]
            RFS2 [0 1]
     CFS2 [1]
            RFS1 [1 0]
```

Now, someone wishes to add a new CFS (“CFS3”) to position “0 1”. The new CFS has two children, “RFS4” and “RFS5”. In this case, the resulting product instance becomes:

```
PI1 []
     CFS1 [0]
            RFS1 [0 0]
            CFS3 [0 1] - NEW
                   RFS3 [0 1 0]
                   RFS4 [0 1 1]
            RFS2 [0 2]
     CFS2 [1]
            RFS1 [1 0]
```

Notice that the “path” of “RFS2” has changed as a result of the insertion of “CFS3”.

Shadow Characteristics

In some cases it is convenient from with an action workflow to be able to access *previous* values of certain characteristics. For instance, if an RFS’s action workflow for a “modify” state transition needs to change an IP address, then both the new as and the old IP address may be needed; the old IP address might be needed so that the address can be freed from a pool of resources, and the new IP address will be needed for making the necessary changes in the network.

Shadow characteristics provide a convenient way of getting access to the previous values for all characteristics. Before a new value is assigned to a characteristic HP Service Provisioner will copy the current value of the characteristic to a so-called *shadow characteristic*; this will be done for all characteristics in a product instance, including all its children.

Service Provisioner comes with two workflow nodes that can be used in action workflows to access shadow characteristic values. One workflow node is a process node that can assign current and/or shadow characteristic values to case-packet variables. The other workflow node is a rule node that can be used to make a logic decision based on whether or not a one or more characteristic values have changes (i.e. whether or not the *current* values match their *shadow* characteristic values).

NOTE — The two workflow nodes are documented in the Section “Workflow Nodes for Accessing Characteristics” on page 98.

In certain cases it may be desirable to *not* run an action workflow for a modify operation. As an example, consider user who has subscribed to a triple play service modeled by product instance containing a “Data” service, a “Video” service, and a “Voice” service. Now, if the user wishes to modify the bandwidth of the “Data” service but not touch the two other services, then it would be convenient if it was possible to completely skip the action workflows for the “Video” and “Voice” services.

With HP Service Provisioner it is, in fact, possible to set a flag on action workflows (this has to be done in the catalog) so that they will be skipped in the event of a modify operation, *provided* that none of the characteristic values have been changed. If the flag is set, Service Provisioner will – before reaching the point where the action workflow is to be called – compare all the service’s characteristic values to their shadows, and if none of the characteristic values have changed the action workflow will not be invoked.

Manual Design and Assign

Design of a product means designing its components, i.e. the tree of CFSs, RFSs, and EWIs. In general the design is determined once and for all in the catalog, so that the state transitions to the desired end state can run automatically.

However, it is possible to allow a manual design action. It can be specified in a PS or service specification that a manual design action shall take place for all instances of the product (if set on the PS or any of its member CFSSs, RFSSs, or EWISs), or it can be specified by a parameter in the service request to design an instance of a product. Then, after the product instance has been created and persisted by the automated process that follows the tree structure defined in the catalog, a manual action to consider and optionally alter the design must take place before the product instance is considered to have fully reached the “Designed” state.

How to interact with HP Service Provisioner to perform such a manual action is described in the Section “Performing Manual Design” on page 57.

Manual action makes it possible to remove or to add services or complete sub-trees from/to the product instance, using all services specified in the catalog. Also, changing their characteristics is possible.

This is a very complete capability to alter a design, because everything that is created, provisioned and activated for a product is expressed as services.

A further possibility for manual interaction, applying to all state transitions, is to include user interaction in the state transition action workflows. This is a general mechanism to include a process that the user controls. It could be manual work, or it could be something the user does by invoking another system. Hence it is a mechanism to integrate HP Service Provisioner with a peer system in a way that was not specifically planned and implemented as a system integration, by asking the user to work as intermediary.

For example, a manual design and assign process for a circuit included into a product as a single RFS can involve Trueview Inventory by including in the action workflow algorithm for the “reserve” state transition (from *Designed* to *Reserved*) a user interaction that works as follows:

1. The action workflow interacts with a user, asking him/her to perform a manual design using Trueview, i.e. its user interface. The endpoints of the circuit are known as RFS characteristics and are displayed to the user as part of the specification of the circuit. The user is asked to obtain and enter the Trueview identifier of the designed circuit in response to the interaction.
2. The user will take the information provided as input and transfer it manually to the proper Trueview UI, ask Trueview to design the circuit, forcing Trueview to assign and keep track of the necessary intermediate resources. When Trueview completes this design and assign task, the final result will be available as an identifier assigned to the circuit. Note that there is some leeway for the user to control the circuit design as long as the RFS characteristics are respected. This output is then transferred manually back to the interaction with the RFS state transition workflow, hence to HP Service Provisioner.

Of course, integration with Trueview Inventory could also be automated, if it is foreseen when an automated solution is developed. The description above covers unforeseen integration points, or points that are not supported by the API that is exposed by Trueview Inventory but only by its user interface.

Force Operations

In addition to the processes described earlier in this chapter, the HP Service Provisioner process engine supports “force” operations with the following capabilities:

- Set the state of a product instance (and its children) to any of the possible service order states. No action workflows will be executed; *only* the state is set.
- Delete a product instance from HP Service Provisioner (and from Subscription Repository). Again, no action workflows will be executed.

These operations can be useful in cases where a product instance is left in an inconsistent state; for instance, due to a rollback error.

Force operations are invoked by sending a service request to HP Service Provisioner WS interface containing the service id to identify the product instance as well as the `<Force>` element containing the desired state of the product. A list of supported `<Force>` values (for the default life-cycle profile) and their meanings is shown in Table 2. Please note that force operations *completely* ignore the current state of the product instance.

Table 2 List of Supported Force Operations for the Default Life-Cycle Profile

Force Value	Effect
Designed	Set the state of the product instance to “Designed”
Reserved	Set the state of the product instance to “Reserved”
Provisioned	Set the state of the product instance to “Provisioned”
Active	Set the state of the product instance to “Active”
Terminated	Set the state of the product instance to “Terminated”
Checked	Delete the product instance
Initial	Delete the product instance
Retired	Delete the product instance

Life-Cycle Profiles

The descriptions in all previous Chapters and Sections have been done with the assumption that the life-cycle of the fulfillment processes supported by HP Service Provisioner use the states (and thereof following state transitions and operations) described by MTOSI/TMF 518.

As of version 7.0 two additional so-called *life-cycle profiles* have been added to HP Service Provisioner; so a total of three life-cycle profiles are now supported. The names of these life-cycle profiles are:

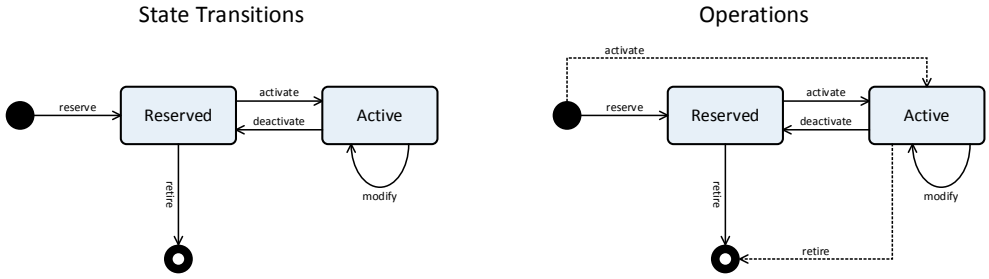
- **Default:** This life-cycle profile is used if no other life-cycle profile has been explicitly selected. It implements the states, state transitions, and operations described by MTOSI/TMF 518.
- **Reserved:** This life-cycle profile is a much simplified version of the *Default* life-cycle profile. In addition to the *non-existing* state it only contains the states *Active* and *Reserved* (thereof the name of this life-cycle profile).

- **Simple:** This life-cycle profile is the simplest imaginable life-cycle profile (hence, its name) containing only the state *Active* in addition to the *non-existing* state.

The supported state transitions as well as the supported operations for the *Default* life-cycle profile were already introduced in Chapter 1 (see Figure 3 and Figure 4).

Figure 9 shows the supported states, state transitions, and operations for the *Reserve* life-cycle profile. All state transition names coincide with operations of identical names; only the operations *active* and *retire* can span across two state transitions (illustrated using dashed arrows).

Figure 9 **State Transitions and Operations for the *Reserved* Life-Cycle Profile**



The supported states, state transitions, and operations for the *Simple* life-cycle profile are illustrated in Figure 10. For the *Simple* life-cycle profile there is a full match between all operation names and state transition names as well as their start and end states.

Figure 10 **State Transitions and Operations for the *Simple* Life-Cycle Profile**

NOTE Although not enforced by HP Service Provisioner, any given solution should stick to a single life-cycle profile in order to keep the semantics of states, state transitions, and operations consistent across all supported services.

Conditional Child Removal

In HP Service Provisioner it is possible for child specifications (i.e. CFSSs, RFSSs, and EWISs) to specify conditions that can cause instances created from them to be removed (or kept). An example where this functionality can be used could be a triple-play service (data, voice, and video) where it is possible for the subscriber to opt out of the voice or the video service (or both).

In the catalog it is possible for CFS specifications, RFS specifications, and EWI specifications to specify the following two parameters:

- **Included parameter:** The value of this parameter must contain the name of one of the characteristics that belong to this specification.

- **Included condition:** This parameter is used to define the condition for which this specification (or more precisely, the *instance* based on this specification) shall be included in the product instance tree. The supported conditions are identical to the *restrictions* that are supported for all characteristics; please read the Section “Restrictions” on page 47.

The evaluation of the *included* parameter (i.e. the decision as to whether or not to remove this child) is done when the very first state transition has been completed; i.e. for the *Default* life-cycle profile this means just before the *Checked* state is reached.

An example of a product specification containing child CFSSs that can be conditionally stripped off is shown below (RFSs are not included in this example):

```
3Play (PS) --|
             |-- Data (CFS)
             |
             |-- Voice (CFS)
             |     - Included parameter: include_voice
             |     - Included condition: YES,TRUE (Enumeration)
             |
             |-- Video (CFS)
                   - Included parameter: include_video
                   - Included condition: YES,TRUE (Enumeration)
```

It is assumed that the characteristics `include_voice` and `include_video` are defined in the `3Play` product specification and mapped to its CFSSs children to characteristics of identical names.

Now, when creating an instance based on this specification, the fulfillment processes will copy the values of the characteristics `include_voice` and `include_video` to the child CFS instances. If the service request sets the value of these two characteristics to `YES` or `TRUE`, then the `Video` and `Voice` CFSs will be included in the product instance tree. Otherwise, if the characteristics are set to any other values then Service Provisioner will strip off the CFSs (`Voice` or `Video` or both) just before the *Checked* state is reached.

If a CFS has been deleted from the product instance tree it may be re-added later on using a “modify-add” operation. For more details on this please read the Section “Structural Modify” on page 26.

3 Implementation Architecture

This chapter describes HP Service Provisioner from a software implementation perspective. This information does not bear on the functionality of HP Service Provisioner software, but it is important for operation of the software and to understand how state transition action workflows access the data stores of service catalog, service inventory, and resource inventory.

The order processing engine of HP Service Provisioner is itself implemented as workflows (most importantly, `SOMController`, `SOMAction`, and `SOMRollback`) that execute in the workflow engine of HP Service Activator. The roles of the three workflows are as follows

- **`SOMController`** is responsible for orchestrating the Service Order Management state machine and for all external communication (via JMS and WS/SOAP/REST); actions that are executed are pushed onto a so-called *action stack*. In addition, this workflow is responsible for handling scheduled requests, delta calculation, conflict detection, force operations, and cancelation of ongoing processes.

- **`SOMAction`** is responsible for performing a depth-first traversal of all CFSs, RFSs, and EWIs in a product instance and for invoking the processes associated with the PI, the CFSs, the RFSs, and the EWIs (these processes are also implemented as workflows).

- **`SOMRollback`** is responsible for performing a full or a partial rollback in case of a consistent error in one of the action workflows. It does so by popping actions off the action stack one by one and reversing each action.

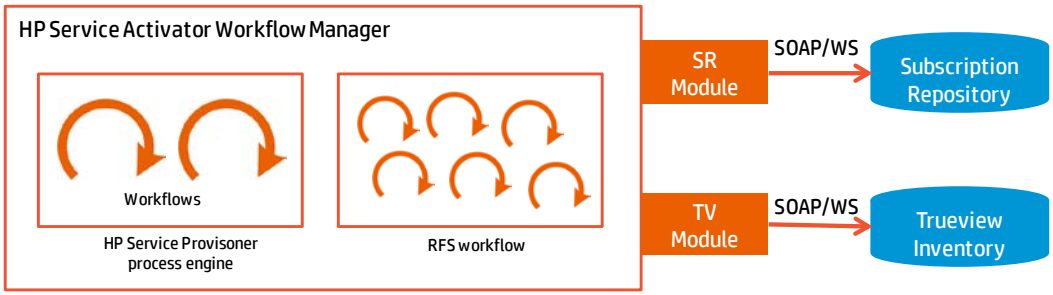
The interaction between the HP Service Provisioner process engine and action workflows happens simply by running the latter within the same workflow engine directly under the control of the process engine workflow as the parent job.

Of the three data stores which will be part of a solution based on HP Service Provisioner, the service catalog and service inventory, although they are conceptually distinct, are both implemented with HP Subscription Repository software.

The third data store, resource inventory, is less tightly integrated into HP Service Provisioner, as it is not accessed from the HP Service Provisioner process engine. Access to resource inventory will be needed from state transition action workflows. The resource inventory offering for HP's integrated service operations factory is Trueview Inventory. Therefore, HP Service Provisioner comes with features to allow access to Trueview Inventory, but it will be possible on a project basis to use a different inventory.

In relation to the HP Service Activator workflow engine both of the inventory platforms are cooperating systems which expose web service interfaces for integration. As shown in Figure 11, HP Service Provisioner includes workflow manager modules for both integrations. There is also a library of workflow nodes for access to the service catalog and the service inventory. For more a full description of the workflow manager module and the workflow nodes, read Chapter 12 on page 95.

Figure 11 **HP Service Provisioner Implementation Architecture**



For operation and administration of Subscription Repository and Trueview Inventory please refer to the documentation for those products.

The primary user interface for HP Service Provisioner includes two UIs, one for editing the catalog, described in Chapter 6, and one for monitoring and interacting with running orders, described in Chapter 7. In addition, there is a user interface that allows the user to enter orders in a generic manner.

All the HP Service Provisioner user interfaces are integrated in the (zero client-side footprint) web service user interface for HP Service Activator, which is generally described in *HP Service Activator, User's and Administrator's Guide*.

4 Installation

This chapter guides you through the steps required to install HP Service Provisioner. At a glance, installation of HP Service Provisioner consists of the following steps:

- Install HP Service Activator
- Install HP Subscription Repository
- Deploy the HP Service Provisioner solution using the HP Service Activator Deployment Manager
- Configure the required modules: `SRModule`, `TrueviewModule`, `JMSSenderModule` (or `SocketSenderModule`) , and `DBAuditModule`

For information about installing HP Service Activator and HP Subscription Repository, please refer to their respective documentation.

NOTE — This document does not describe how to set up a JMS server. You may choose to use the JMS software that comes with JBoss AS 7.1 which is bundled with the HP Service Activator product. In that case, please consult the JBoss documentation for detailed instructions.

Deploying HP Service Provisioner

The HP Service Provisioner framework software is packaged as a zipped HP Service Activator solution named `SOM.zip` which is installed along with the HP Service Activator core product (in the directory *`$ACTIVATOR_OPT`*`/SolutionPacks`) and can be deployed with the HP Service Activator Deployment Manager (which is introduced briefly in *HP Service Activator, System Integrator's Overview* and thoroughly document in the dedicated manual *HP Service Activator, Solution Separation and the Deployment Manager*).

To deploy HP Service Provisioner follows these steps:

- Launch the Deployment Manager and configure the system database parameters (typically, only the system database username and password need to be entered).
- Under "Local Deployment", click the "Import Solution" menu item, select the ZIP file *`$ACTIVATOR_OPT`*`/SolutionPacks/SOM.zip`, and click the `[Import]` button.
- Click the "Deploy Local Solution" menu item, select the solution named "SOM", and click the `[Deploy solution]` button.

Now the HP Service Provisioner software (workflows, UI components, workflow nodes and modules) have been deployed.

Configuring HP Service Provisioner

Before the HP Service Provisioner software can be used, you need to enable generation of service ids and then configure four workflow manager modules as described in this section.

Generate Service Id

To use Service Provisioner, the Workflow Manager must be configured to automatically generate service ids. This is done by setting the value of the XML element `<Generate-Service-ID>` to "true" in the Workflow Manager's configuration file, *$ACTIVATOR_ETC*`/config/mwfm.xml`.

SRModule and TrueviewModule Configuration

NOTE — Configuration of the `SRModule` (named "ServiceOrderManagement") is mandatory for running HP Service Provisioner. If HP Trueview is used as the inventory system, you must also configure the `TrueviewModule`.

To use the `SRModule` and `TrueviewModule` they need to be added to the Workflow Manager's configuration file, *$ACTIVATOR_ETC*`/config/mwfm.xml`. It is recommended to copy the examples in the file *$ACTIVATOR_OPT*`/solutions/SOM/etc/newconfig/mwfm_SOM.xml` and then edit the configuration parameter to suit the environment. For a full documentation of the parameters supported by the modules, please read the documentation in Chapter 12 (page 95).

If you do not wish to have clear text passwords in your configuration files, you can enable support for encrypted passwords. In that case, you need to encrypt the password using HP Service Activator's crypt utility and paste the encrypted password into the configuration file.

The following example shows how to encrypt the password `verySecret`:

```
$ACTIVATOR_OPT/bin/crypt -encrypt verySecret
Text is verySecret and encrypted text is t4q0rlkf294JsRdTXn7SJA==
```

Hence, the encrypted version of `verySecret` is `t4q0rlkf294JsRdTXn7SJA==`.

Sender Module Configuration

The workflows implementing the service order management processes require that HP Service Activator has a SenderModule enabled; typically, the `JMSSenderModule` or the `SocketSenderModule`. Refer to HP Service Activator, Workflows and the Workflow Manager for details of how to enable and configure these modules.

IMPORTANT — The name of the **`JMSSenderModule`** *must* be **`som_sender_queue`**.

Conflict Module Configuration

HP Service Provisioner requires that a Conflict Module is present in order to detect conflicts and enable cancelation of ongoing processes. To enable the Conflict Module open the Workflow Manager's configuration file and uncomment the module named `conflict_module`. The default parameter values of the conflict module will suffice.

Audit Module Configuration

HP Service Provisioner requires that the Audit Module is configured unless the `store_audit` parameter in the `SRModule` is explicitly set to "false"; see the description of the `SRModule`'s `store_audit` parameter in Chapter 12 (page 95). To enable the Audit Module, uncomment the module named `auditor` in the Workflow Manager's configuration file,

$ACTIVATOR_ETC/config/mwfm.xml. Also, remember to set the value of the store_audit in the Audit Module to "true".

HP Service Provisioner stores audit records HP Service Activator's database table named AUDIT_RECORD. Since the use of the fields of audit records is entirely the responsibility of the solution developer (i.e. outside the control of HP Service Activator), no database indexes have been defined for the AUDIT_RECORD database table. This is a deliberate choice because excessive use of database indexes has the potential of dramatically reducing the write performance for a database table (because indexes will also need so be updated in case of DELETE, INSERT, and UPDATE statements).

If a HP Service Provisioner solution needs to be able to query audit records using a specific database column (or specific database columns), then please consider adding indexes to those columns.

NOTE

Please create database indexes with care. For instance, if a column can only contain a very small set of values (for instance true and false) then the use of an index for such a column may not have the desired effect on performance; in fact, it may do more harm than good.

Installing HP Service Provisioner License

HP Service Provisioner comes with an instant-on license which is valid 30 days after installing the HP Service Activator product. To install, inspect for verification, or remove a license for HP Service Provisioner, you can use the utilities checkLicense and updateLicense belonging to HP Service Activator (found in *$ACTIVATOR*/bin).

IMPORTANT

The **checkLicense** and **updateLicense** utilities must be invoked with the **-som** option.

Localization

Localizing HP Service Provisioner is similar to localizing HP Service Activator. Please read the document *HP Service Activator System Administrator's Overview* for instructions on how to localize HP Service Activator.

Localizing HP Service Provisioner Engine Components

The resource property bundles (in English) are for the HP Service Provisioner engine components can be found in the directory *$ACTIVATOR_OPT*/solutions/SOM/etc/nls. The localization process begins with translating the resource bundle files (ending with _en.properties). You must make a copy of each resource bundle file, where you replace _en in the file name with the appropriate abbreviation for the locale, such as _jp or _dk.

Then you must translate the contents of each file to the language of the locale. The files must be saved encoded in the ISO 8859-1 character set with appropriate escape sequences to represent characters that do not have 8-bit codes; the Java utility native2ascii may be helpful to convert from a UTF character set to ISO 8859-1.

Once the resource files have been translated, you must create a Java archive named somnls.jar containing all resource bundle files and copy it to *$JBOSS_EAR_LIB*.

Localizing HP Service Provisioner UI

The HP Service Provisioner UI is implemented with Java Server Faces. The resource property bundles for these parts are found in the directory *$JBOSS_ACTIVATOR*/WEB-INF/classes/jsf-resources. When you add support for a new locale, you must also add that locale in the file *$ACTIVATOR_WAR*/WEB-INF/classes/faces-config/locales.xml.

Deploy SOM Demo Solution

NOTE	Deployment of the SOM Demo solution is optional; it is not a part of the HP Service Provisioner product.

In addition to the solution zip file containing the HP Service Provisioner framework software, a demo solution called “SOM Demo” is also bundled with the installation kit.

The name of the “SOM Demo” solution zip file is SOM_Demo.zip and it is located in the directory *$ACTIVATOR_OPT*/examples/som_demo). It can be deployed with the HP Service Activator Deployment Manager using similar steps to the ones described in the Section “Deploying HP Service Provisioner” on page 35.

The SOM Demo solution consists of sample workflows as well as a sample product catalog. The sample workflows are:

- Five RFS action workflows; the workflows do not perform any actions, as such. Their role is mainly to demonstrate the “contract” between the HP Service Provisioner workflows and the action workflows.

- Two test workflows; one for sending dummy messages and one for listening to messages. In order to use the listener workflow, you need to configure a ListenerModule (e.g. the JMSListenerModule) and set the value of the parameter workflow to SOMTestJMSListener. For more information about configuring the JMSListenerModule, consult the HP Service Activator documentation.

- Three sample workflows that communicate with the Trueview Inventory system. There is one workflow for creating an object in Trueview, one for retrieving an object, and one for deleting an object.

If you wish to use the sample product catalog (which can be found in the directory *$ACTIVATOR_OPT*/solutions/SOM_Demo/etc/data), you can import it into Subscription Repository using the SOMData utility with the –import option. You need to have HP Service Provisioner running before you can use the SOMData utility. Please read the Section “Import and Export of Catalog Content” on page 54.

5 Client Integration

It is expected for CSP deployments of HP Service Provisioner, service order requests will always be received by a CRM or Order Management client operation system. The client facing API of HP Service Provisioner is the service activation interface as depicted in Figure 2. This interface with its interactions and parameter information is described in the first section below.

For stand-alone testing of HP Service Provisioner a manual interface is also supported. It is shown and briefly described in the last section of this chapter.

Northbound API

Two northbound APIs of HP Service Provisioner are implemented as web service interfaces: SOAP and REST. All requests are quickly responded to when they have been checked for syntactic correctness. This synchronous response does not convey the result of the processing of the request, which may take some time depending on the complexity and whether manual effort is involved.

Subsequent response information is returned as asynchronous messages (typically, via JMS). This is because the time to wait for the additional message is variable, and by sending the messages over a mechanism that has the capability to store and deliver asynchronously, HP Service Provisioner and its client become less interdependent operationally: the requesting client is relieved of being able to receive and process the response messages at all times.

First it is checked that references in the request to catalog and service inventory are correct, i.e. all referenced items exist, and an update message is sent. If the request is not accepted due to an invalid reference, that message terminates the interaction.

Then follows the processing of the requested operation which involves actions, such as state transitions, scheduling, delta operations, etc. When everything has been processed successfully, or when processing of the service request ends with a failure, the final response message is sent asynchronously.

All requests to create a new product must refer to a PS in the service catalog whereas all requests to operate on an existing product must refer to the service id of the product instance. Requests received on the northbound interface have the following contents:

Request type	The operation that is requested for a given service order (for instance, “activate”); together with current state of the service order/product instance it identifies the sequence of state transitions to perform.
Request id	Client’s identifier of the request, returned in all responses to allow client to understand what is being responded to.

Order id	Optional, client's identifier of the customer order from which the request is derived; many requests may come from a single customer, the order id of a service order/product instance is displayed and is filterable in several places on the HP Service Provisioner user interface to allow operators to locate activity from a customer order.
Display label	Optional, a label to display on the user interface for the product instance.
Product name and version	Identifies the product specification in the catalog; this is the starting point for retrieving the complete tree of specifications. Mandatory, if creating a new product instance.
Customer	Name of customer, for display on the user interface.
Service id	Not present when the request refers to a non-existing service order; once the service order (and the product instance) has been created, the service id is returned in a response and allows subsequent requests to refer to it.
User Interaction	Optional boolean parameter; indicates, when present, whether manual intervention shall be allowed or not in the design of the product instance, overruling the catalog attribute of the PS (see next section).
Send asynchronous response	Optional boolean parameter (default is "true"); indicates, when present whether Service Provisioner shall send asynchronous responses for this product instance (typically, via JMS).
Priority	Optional, indicates the priority of this service request; higher values means higher priority. Please note that prioritization only becomes noticeable if many processes are contending for the same processing resources.
Solution	Name of the solution to which this product instance belongs.
Queue	Name of the queue to place this request into.
Force	Optional parameter; when set the processing engine enters "force" mode which means that product instances can be forcefully deleted or forced into any state. Action workflows will not be called when operating in force mode. For more information, please read the Section "Force Operations" on page 29.
Characteristics	List of names and corresponding values of product characteristics; these values are the parameters of the service order.
Error handling strategy	Optional parameter, specifies how Service Provisioner shall behave in case of errors ("ATOMIC", "BEST_EFFORT", or "PARTIAL").
Template name and version	Optional, identifies a template from which to take values to assign to the characteristics of a product instance.
CFS trees	Optional, a list of CFS trees that are the immediate children of the product instance. Mainly relevant when used in the context of delta operations; see Chapter 9 on page 71 for details.
Modify actions	Optional, a list of "modify-add" and/or "modify-delete" actions. See the Section "Modify Operations" on page 25 for details.
Schedules	Optional, a list of scheduled requests defined for this product instance; see Chapter 11 on page 89.

When a request has been accepted for processing and the first asynchronous response has been sent indicating the request is valid, the final response sent after all state transitions have been completed will contain:

- Overall result code: 0 for complete and correct execution, 1 for error during a state transition, but rollback successful, 2 for error during a state transition and unsuccessful rollback *or* for an inconsistent error where rollback has not been attempted.
- Explanatory result text (a single string).
- An XML structure containing a list of activities for each state transition, invoked action workflow, added/deleted CFS, etc.

Requests to HP Service Provisioner must be sent to the following URL:

- `http://<SA host>:<SA port>/ServiceOrderManagement/ServiceOrderManagement`

To extract the WSDL document for the synchronous part of the NBI access from a browser the same URL with `?wsdl` appended.

As mentioned in Chapter 4, you must configure for the HP Service Activator workflow manager a sender module with the name `som_sender_queue` to send the asynchronous messages. This module must be configured with the URL of the port where your client will be listening for the messages.

Template files for the asynchronous response messages are found in the folder *`$ACTIVATOR_OPT`*`/solutions/SOM/etc/template_files`.

Conflicts

Whenever HP Service Provisioner runs a process for a product instance, Service Provisioner registers the product instance's service id in HP Service Activator's Conflict Module. If another service request attempts to operate on the same service id the Conflict Module will detect that there is a conflict which will be dealt with by Service Provisioner. In most cases a conflict will be handled simply by rejecting the conflicting service request and sending an asynchronous response with result code 1 (error) and an appropriate error message.

Cancelation

As described, HP Service Provisioner is capable of detecting service requests that are contending for the same product instance. If the conflicting service request contains a "cancel" operation, the operation will be injected into the running process (the one that holds the lock in the conflict module). Next time the running process completes a state transition it will check for incoming "cancel" requests, and if there are any "cancel" requests the running process will be canceled if possible. It is only possible to cancel running processes in the states *Checked*, *Designed*, and *Reserved* (for the default life-cycle profile); if any other state has been reached, the cancel operation will be ignored.

Order Entry UI

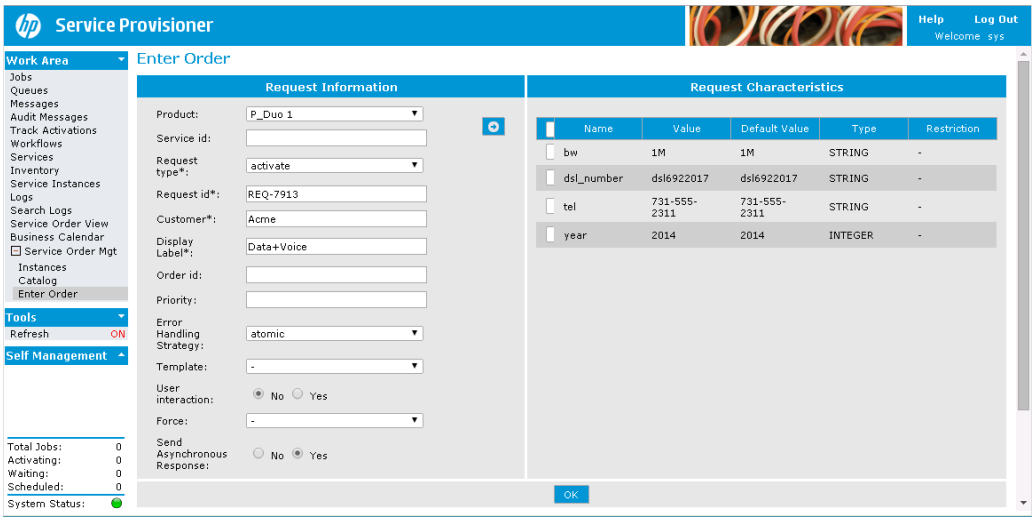
In addition to sending service requests via the NBI, HP Service Provisioner comes with a graphical user interface that can be used for entering orders manually. Please note that even for orders entered using the UI, asynchronous responses will be sent unless if the "send asynchronous response" field is explicitly set to "no". To track progress of manually entered orders, use the "Instance Management" user interface as described in Chapter 7.

The screen window for entering an order is included in the user interface of HP Service Activator as a menu item that can be selected in the Work Area menu. This is shown in Figure 12. Select

first then “Service Order Mgt” menu item, and then “Enter Order”; the order entry window will then appear in the work area.

The data entry fields in the window are all described in the preceding section as contents of the request message to create a service order.

Figure 12 **Manual Entry of Service Request**



6 Editing the Service Catalog

This chapter is primarily of interest for the system integrator or technical product manager who will define and maintain products and services in the service catalog.

An introduction to this topic was given in Chapter 1 (the sections “Catalog Driven” and “Service Orders and Product Instances”). The contents of those sections are assumed to be present in the reader’s mind. Additional detail is given here.

There are six kinds of items to define in the service catalog: Templates, Profiles, EWISs, RFSSs, CFSSs, and PSs. Because of dependencies between them, they will normally be created in the listed order, i.e. bottom up.

All items in the catalog have three common attributes: name, version, and description. The combination of the name and the version identifies the item and is used to reference it. This combination must be defined when the item is created and cannot subsequently be changed. The description serves as documentation and may be edited at any time.

An important aspect of any product or product specification is the characteristics it defines for the products or services to be derived from it. Conceptually a distinction is made between variant and invariant characteristics, as discussed in Chapter 1. This distinction is not specified in the catalog, but only by the way characteristics are used in service order processes. In the catalog a value must be specified, even for characteristics that are intended to be variant. It will serve as default, in cases where no values is provided for that characteristic.

Characteristics may be defined as part of a profile, or they may be defined directly on an EWIS, RFSS, CFSS, or PS. Editing of characteristics is similar in all those cases and explained (only) in the “Profiles” section.

Instead of assigning values to individual characteristics when creating a new service instance, it is possible to create templates which contain groups of characteristic values which can be assigned to instances by referring to the name and version of the template.

The screen window for editing the catalog is included in the user interface of HP Service Activator as an item that can be selected in the Work Area menu. This is shown in Figure 10 and the following figures. Select first “Service Order Mgt” and then “Catalog” to make the “Catalog Management” window appear in the work area.

Administrative and Operational States

All types of catalog items support a so-called *administrative state*; the value of this state can be set through the user interface. The administrative state can have either of the following four values:

- **editable:** When catalog items are in this state they may be edited through the user interface; they are not available for use by services. Newly created catalog items will have this administrative state (unless another administrative state has been explicitly assigned to the catalog item).

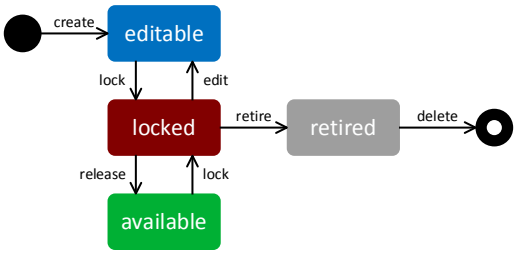
- **locked:** Catalog items in this state can neither be edited nor are they available for use by services.

- **available:** When catalog items are in this state they are available for use in services. Catalog items in this state cannot be edited.

- **retired:** Catalog items in this state can neither be edited nor are they available for use by services. Catalog items in this state behave identically to catalog items in “locked” state. The intended use of the “retired” state is to indicate that a catalog item is never to be turned “active” again.

HP Service Provisioner defines no restrictions for how to switch between the four administrative states, it is entirely up to the user to ensure that the administrative values are set to appropriate values. However, in typical use cases the life-cycle of the administrative state for a catalog item is expected to be as shown in Figure 13. In the “Catalog Management” UI the administrative state of a catalog item is marked by its color; the colors used to indicate the four different administrative states are identical to the colors used in Figure 13.

Figure 13 **Administrative States for Catalog Items**



There are two ways to change the administrative state of a catalog items. Either the administrative state can be changed from the “Create/Update” UI (this UI is only available if the current administrative state of the catalog item is “editable”) or it can be changed by clicking on the small arrow in the catalog tree to the right left of the catalog item’s name/version and then select the “Administrative State” menu item.

In addition to the administrative state, HP Service Provisioner also supports an *operational state* for PS, CFSS, RFSS, and EWIS catalog items. The operational state cannot be explicitly set; instead it is derived from the administrative state of the catalog item itself and all of its constituent components. The operational state can only have one of the two values shown below:

- **available:** This will be the operational state of a catalog item if the catalog item itself *and* all of its constituent parts (child items, profiles, and templates) are in the administrative state “available”. It is only possible to create product instances (and CFSs) from specifications that are operationally available.

- **not available:** This will be the operational state of a catalog item if the catalog item itself *or* any of its constituent parts (child items, profiles, and templates) are in an administrative state different from “available”. It is not possible to create product instances (and CFSs) from specifications that are operationally “not available”.

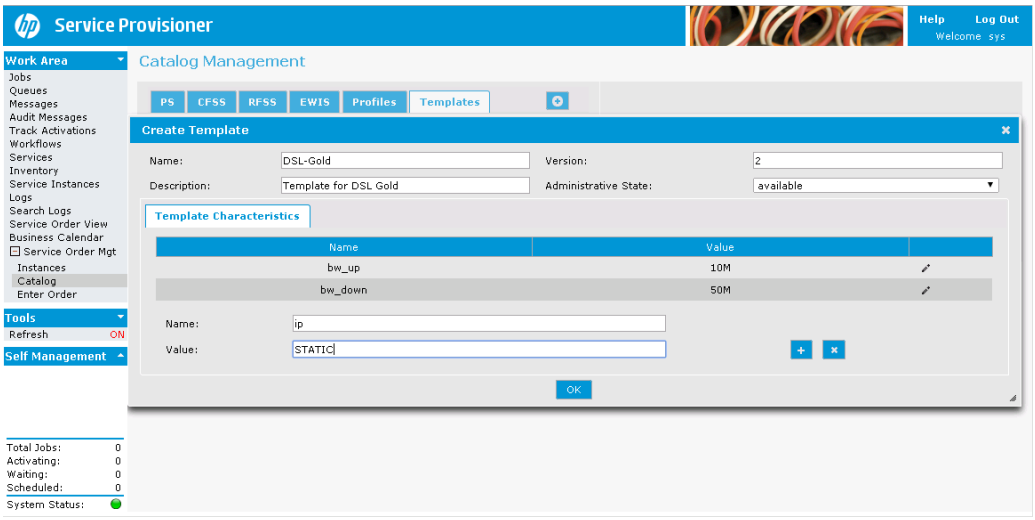
The operational states of catalog items are shown in the catalog tree to the left of the names/versions of the catalog items. All catalog items that are operationally “available” are marked with green tick marks.

Templates

A template is a group of characteristic values. Once defined it can be used to assign multiple values to characteristics in a product (or CFS) just by referring to the name and version of the template.

To create a new template, go to the “Catalog Management” UI, click on the “Templates” tab, and then click on the [+] button located to the right of the tabs; Figure 14 shows a screenshot of the “Create Template” UI.

Figure 14 **Catalog Management, Templates**



Like all other catalog items, templates must have a name and version through which they can be uniquely identified. Adding characteristic values to a template is simple. Enter a name and value and click in the [+] button. To delete a characteristic, click on the pencil icon in the table row to make the characteristic editable and then click on the blue [×] button (the one next to the [+] button). Once all characteristic values have been added and/or delete click on the [OK] button.

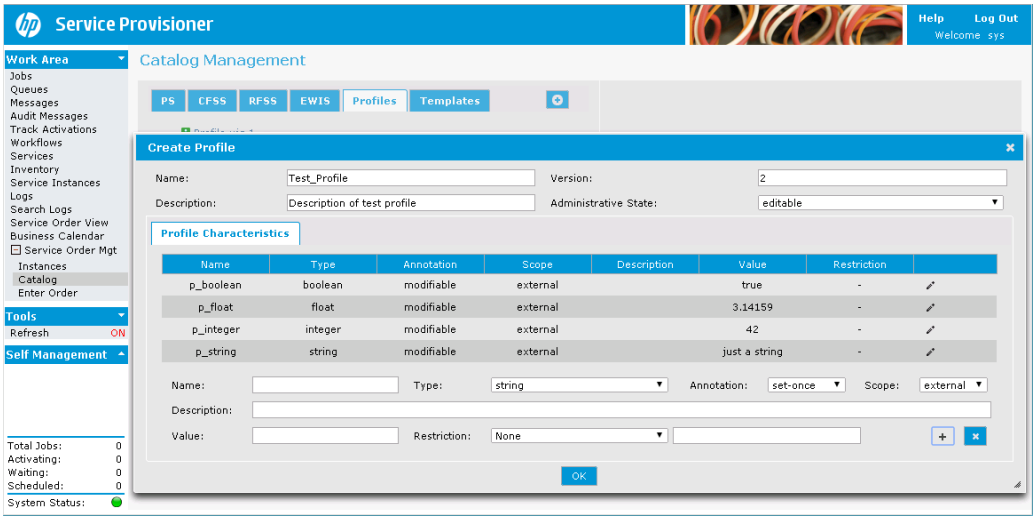
Profiles

A profile is simply a group of characteristics with defined invariant values. It can be defined once and used in several specifications as a convenient shorthand to include all those characteristics. Every product or product specification can include multiple profiles. Even when a characteristic is defined with a value in a profile, it can still be redefined with the same name, with or without an invariant value, in an instance or specification, to overrule the value from the profile.

Figure 15 shows the Catalog Management window with focus on profiles. Select the Profile tab, and a list of defined profiles will appear on the left (under the tabs).

To create a new profile definition, click the [+] button shown right to the bar with the tabs, and a Create window will pop up (the Create window and the Update window are very similar). As for all other catalog items the Name and Version fields are editable and mandatory to fill. The list of characteristics will initially be empty. A profile must contain at least one characteristic.

Figure 15 **Catalog Management, Profile**



To delete a profile, click on the arrow icon to the right of the item in the catalog tree and select Delete in the pop-up menu which appears.

Once a profile has been defined in the catalog, it can be referenced in EWISs, RFSSs, CFSSs or PSs.

To inspect a profile, select it in the list. Its attributes will be shown and its characteristics listed on the right.

To edit a profile, click on the arrow icon to the right of the item and select Update in the pop-up menu which appears. An Update window pops up as shown in Figure 15. At the top of the window are fields showing the name, version, and description attributes of the profile; in addition, the administrative state is shown. Only the description and the administrative state can be changed in the Update window. Next, the window contains a list of the currently defined characteristics, showing for each one, its name, type, annotation, scope, description, value, and restriction and an edit button (a pencil icon).

At the bottom of the window, under the list, is an edit area where one characteristic can be edited at a time. Click the edit button (the pencil) in the row for an existing characteristic to edit it, or just fill the fields to define a new characteristic. To commit a new characteristic or changes to an existing one, click the `[+]` button. To delete an existing characteristic, click the `[×]` button.

If the name of an existing characteristic is changed during editing, committing it will add a new characteristic with the new name.

Name, type and value are mandatory to define a characteristic, the other fields are optional. The description of a characteristic serves documentation purposes only; it has no semantics.

A value must always be specified when a characteristic is defined. Even if the intention is that the characteristic shall be variant, with a value to be specified from a request parameter, from manual editing, or obtained from a state transition workflow, possibly through a mapping, the specified value will serve as a default value. For characteristics of type string, empty values are allowed.

The following types are supported: string, boolean, integer, date (format: dd/MM/yyyy), and float (decimal number).

Restrictions, annotations, and scope are optional and selected from drop-down lists. Detailed descriptions of restrictions, annotations, and scope are given in the following subsections.

To commit the complete profile definition to the catalog, click the `[OK]` button.

Restrictions

A specified restriction will be applied to characteristics values when they are received, as service order parameters on the northbound interface or from manual entry. Available restrictions (types) are:

- **String:** Supported restriction types are “string length”, “regular expression”, and “string enumeration”
- **Integer:** Supported restriction types are “integer maximum”, “integer minimum”, and “integer range”
- **Float:** Supported restriction types are “float maximum”, “float minimum”, and “float range”
- **Date:** Supported restriction type is “date pattern”

As described above, it is possible to define restrictions for characteristics. Restrictions are divided into two parts; a restriction type and a restriction value. When a restriction type other than “NONE” has been selected, the value(s) specifying the details, such as regular expression or value interval must be entered in the field to the right. The syntax for the restriction values depends in the restriction type. Valid restriction values are (listed by restriction type):

- **String length:** An integer value
- **Regular expression:** A valid regular expression
- **String enumeration:** A comma-separated list of string values; e.g. “PLATINUM, GOLD, SILVER, BRONZE”
- **Date pattern:** A valid date pattern; e.g. “yyyy-MM-dd”
- **Float maximum / Float minimum:** A float value
- **Float range:** Two float values separated by a slash; e.g. “1.75/10.25”
- **Integer maximum / Integer minimum:** An integer value
- **Integer range:** Two integer values separated by a slash; e.g. “10/200”

Characteristic Annotations

In some cases it will be convenient for solution developers to be able to control if and when it shall be possible to set or modify values of characteristics. For instance, there may be characteristic values that must be identical for all product instances belonging to a specification (invariant), characteristics that can only be set when the instance is created, and characteristics that can be modified at any time.

HP Service Provisioner has so-called characteristic annotations to support these cases. The three possible characteristic annotation values are:

- **Constant:** The value of the characteristic can only be set in the catalog; i.e. in the specification, not in the instance. This means that once the instance is created the characteristic cannot change its value as a result of a service request. In case of “manual design” the characteristic *may* be modified (although this will normally be discouraged).

- **Set-once (default):** The value of the characteristic can be set when creating the product instance. However, once the product instance has been created the characteristic will be treated as if it was “Constant”.

- **Modifiable:** The value of the characteristic can be modified at any time.

Scope

When designing a service the number of characteristics can sometimes become unmanageable because there may be a lot of characteristics that are only used inside the service’s action workflows; hence, they have no significance for other services using this service.

The visibility of characteristics (on the UI) can be controlled be setting the “scope”; the following scopes are possible:

- **internal:** This scope is used to mark characteristics that are only intended to be of interest to the service’s own action workflow and/or for input/output mappings from/to this service to/from child services, if any.

- **visible:** This scope is used to mark characteristics that are only intended to be modified by the service’s own action workflow and/or as a result of output mappings to this service from child services, if any. As opposed to internal characteristics, characteristics with this scope will be visible on the UI.

- **external:** This scope is used to mark characteristics that are of “external” interest; i.e. they may be modified by the service’s own action workflow and/or as a result of output mappings to this service from child services, if any. But they may also be of interest to parent services; for instance, because the parent service will pass values to these characteristics through parameters mappings.

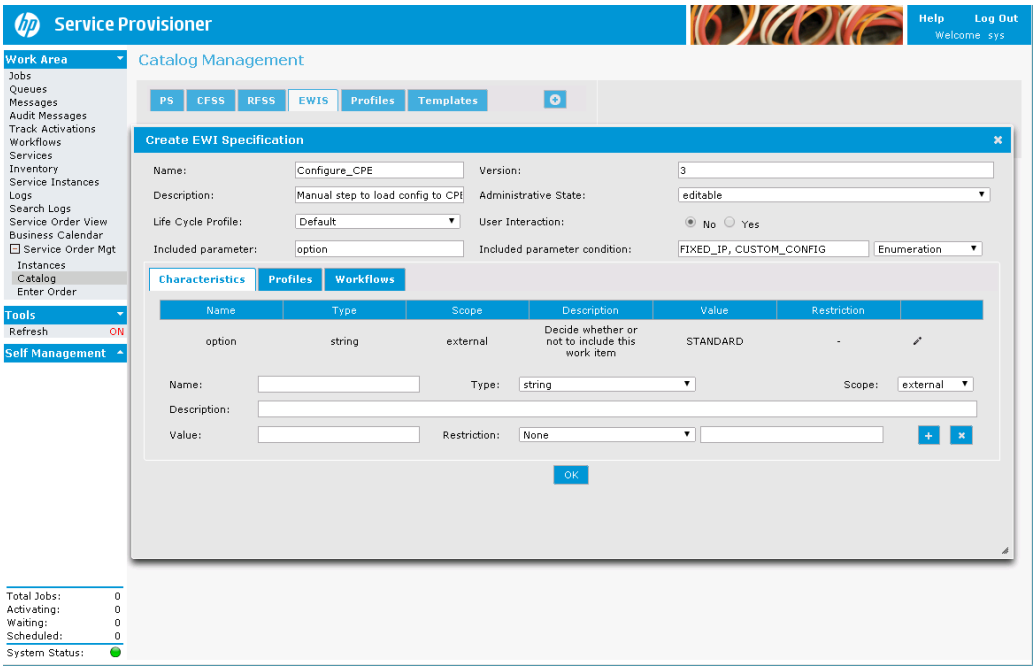
EWI Specifications

Figure 16 shows the Catalog Management window with focus on EWISs. Select the EWIS tab, and a list of defined EWISs will appear on the left. To inspect an EWIS, select it in the list. Its attributes will be shown and its characteristics listed on the right. The attributes of an EWIS include the names of its associated state transition action workflow and an optional list of profiles; note that the EWIS shown in Figure 16 does not have any profile associated. There are no mappings of characteristics, as an EWI cannot be a parent branch in the tree structure of a product instance; it can only appear as a leaf.

In the same way as with profiles, you can edit or delete existing EWISs and create new ones as described in the section “Profiles” above. The editing steps are very similar, the only difference is that an RFSS has the three attributes Profile, Workflow and User Interaction that are not found on profiles.

The boolean attribute User Interaction, if set, will force the state transition process to reach the state *Designed* to finish with a manual action, for any product that includes the specified EWI. It has a similar effect when it appears on an RFSS, a CFSS, or on a PS.

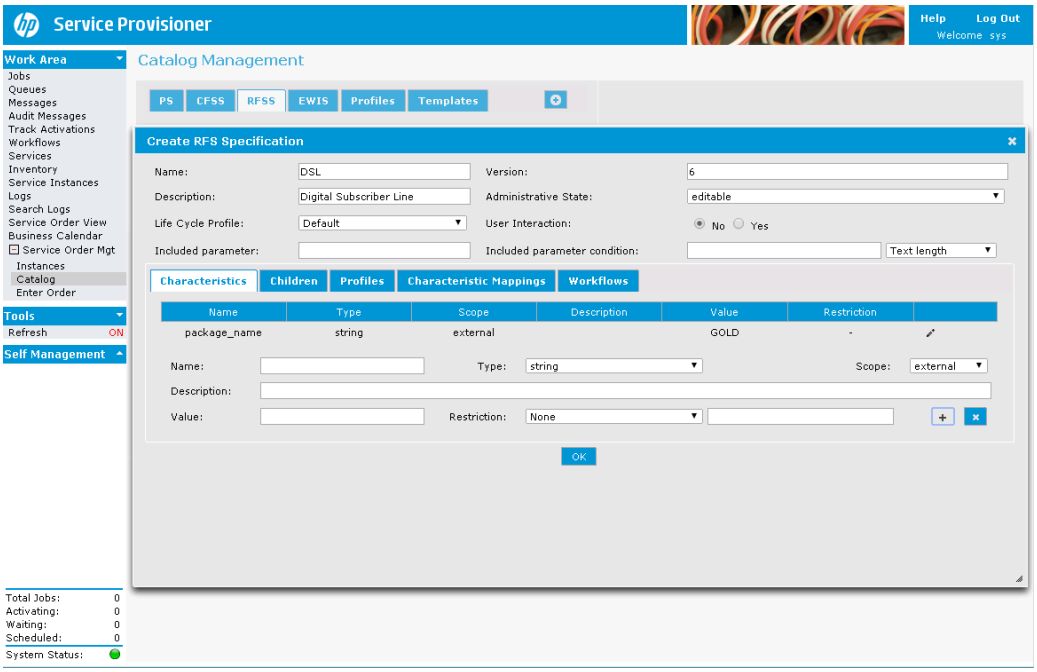
Figure 16 **Catalog Management, EWI Specification**



RFS Specifications

Figure 17 shows the Catalog Management window with focus on RFSSs. Select the RFSS tab, and a list of defined RFSSs will appear on the left. To inspect an RFSS, select it in the list. Its attributes will be shown and its characteristics listed on the right. The attributes of an RFSS include the names of its associated state transition workflow and an optional list of profiles. And RFSS may have other RFSSs or EWIs as children; mappings to these can be defined by selecting the tab labeled “Characteristics Mappings”.

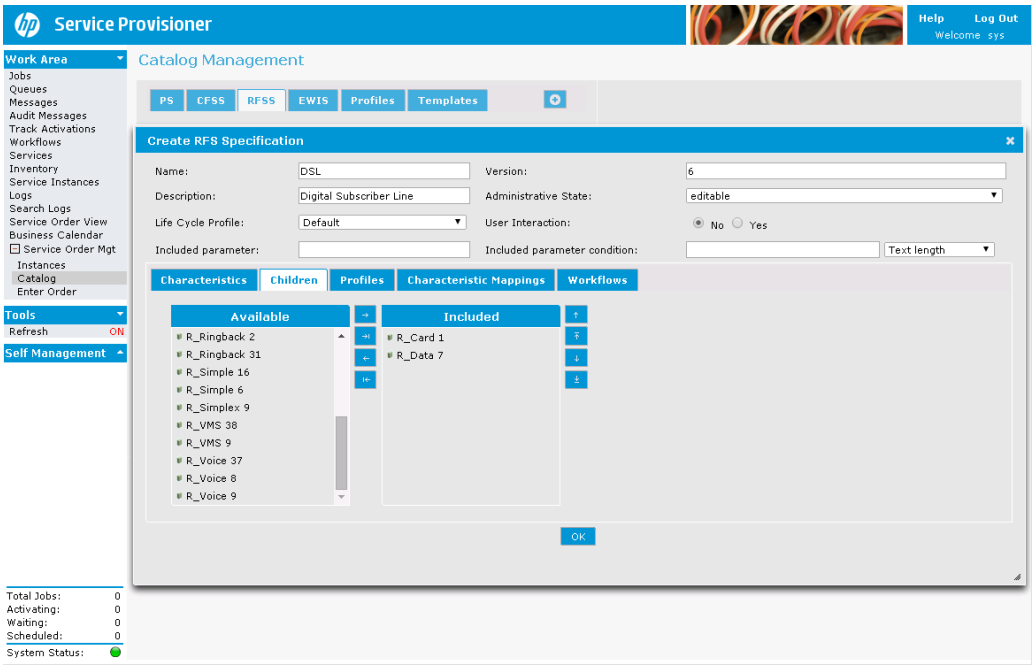
Figure 17 **Catalog Management, RFS Specification**



Existing RFSSs can be edited or deleted and new ones can be created as described in the sections “Profiles” and “EWI Specifications” earlier in this chapter. The editing steps are very similar, the only difference compared to EWISs is that RFSSs may have child items which is the case in the example shown in Figure 18.

The boolean attribute User Interaction, if set, will force the state transition process to reach the designed state to finish with a manual action, for any product that includes the specified RFS.

Figure 18 **Catalog Management, RFS Specification with Child Items**



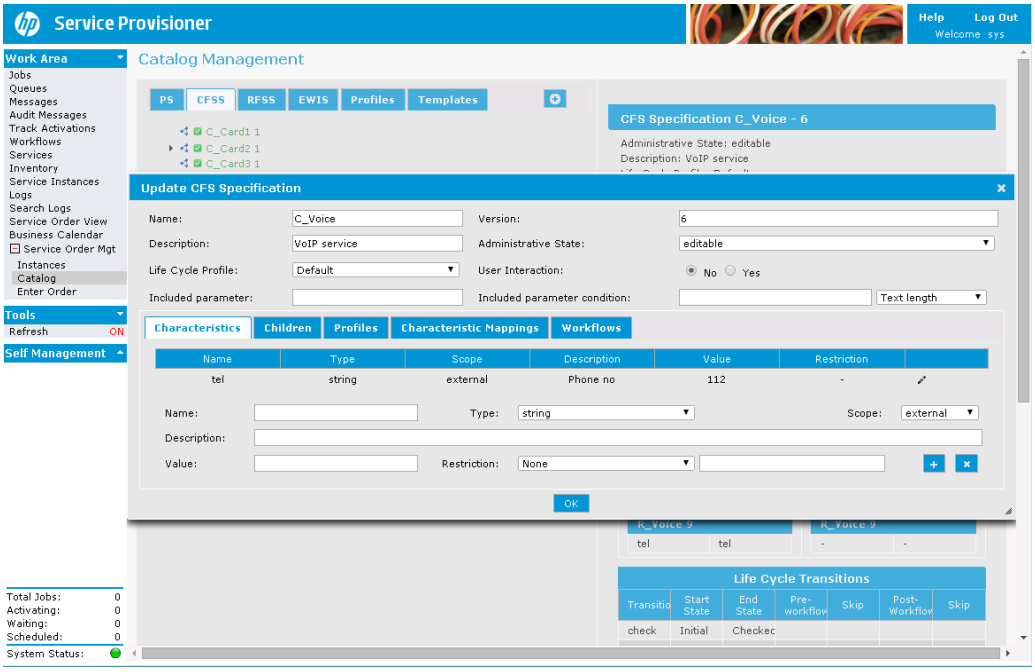
CFS Specifications

CFSSs are quite similar to RFSSs. They can also have contained (child) service specifications, which can be either CFSSs, RFSSs, or EWISs, and for each child specification input and output mappings of characteristics can exist. Like for RFSSs may they also have associated action workflows.

Figure 19 shows the Catalog Management window with focus on CFSSs. Select the CFSS tab, and a list of defined CFSSs will appear on the left. To inspect a CFSS, select it in the list. Its attributes will be shown and its profile, additional characteristics and mappings listed on the right.

In the same way as with profiles and EWISs, you can edit or delete existing CFSSs and create new ones as described in the sections “Profiles” and “EWI Specifications” above. In addition, you can edit the list of child service specifications and the associated mappings; please refer to the section “Product Specification” below, where those steps are described for the very similar case.

Figure 19 **Catalog Management, CFS Specification**



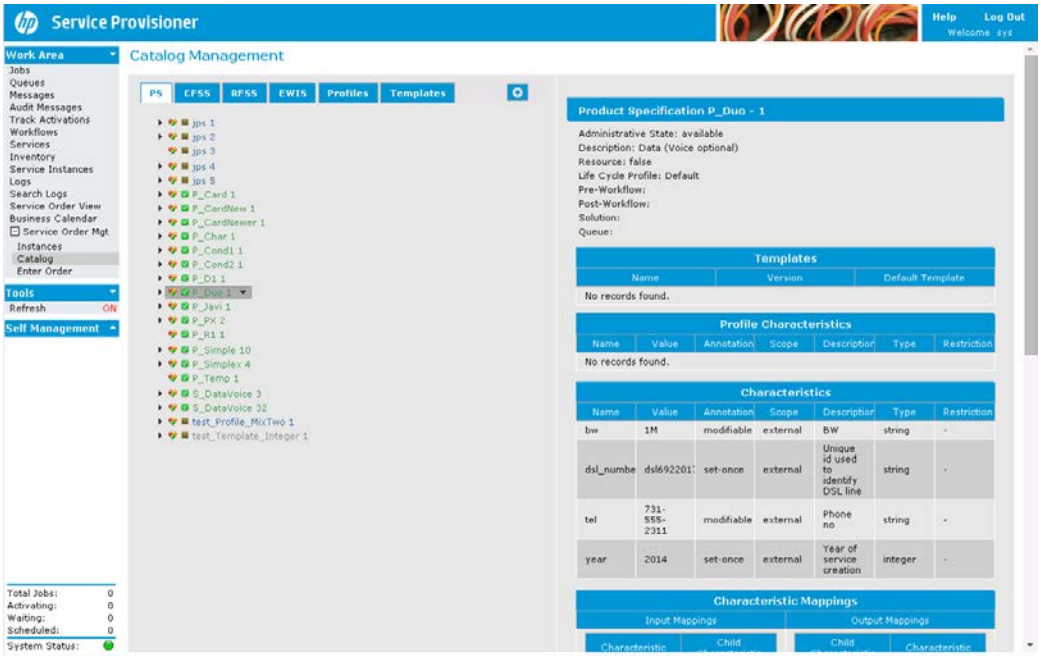
Product Specification

PSs are very similar to CFSSs. Like CFSSs, they can have contained (child) service specifications and mappings of characteristics. The children must be CFSSs or EWISs.

Figure 20 shows the Service Catalog Management window with focus on PSs. Select the PS tab, and a list of defined PSs will appear on the left. To inspect a PS, select it in the list. Its attributes will be shown and its profiles, additional characteristics and mappings listed on the right.

In the same way as with the previously describes catalog items, you can edit or delete existing PSs and create new ones as described in the sections “Profiles” and “EWI Specifications” above. The editing steps for the attributes and characteristics are very similar.

Figure 20 **Catalog Management, View Product Specification**



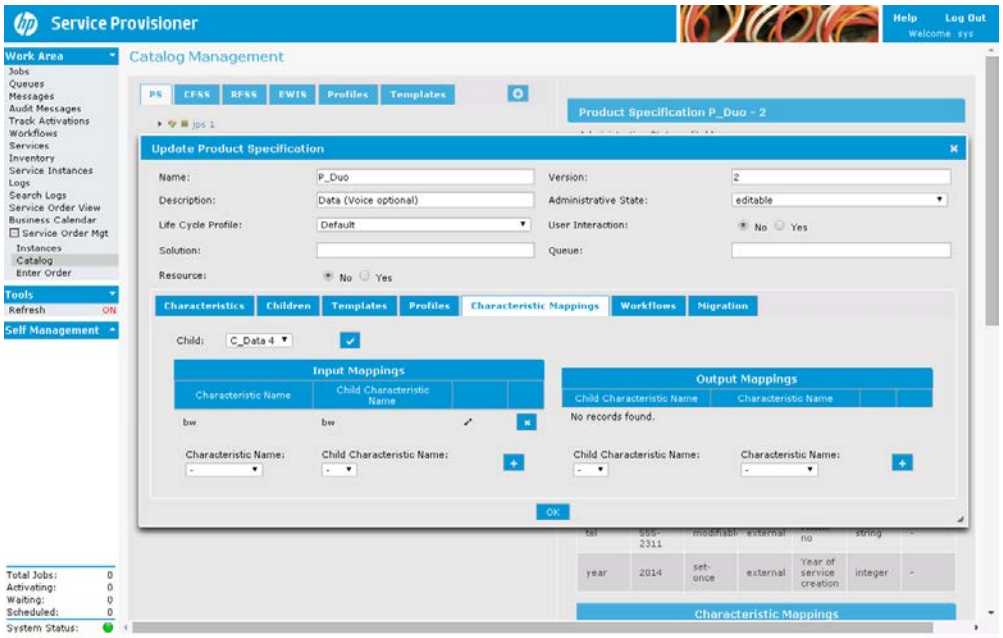
With PSs (and with CFSSs and RFSSs) you can also edit the list of child service specifications. For a PS all children must be CFSSs or EWIs, for a CFSS there can also be RFSSs. Figure 21 shows the Update Product Specification popup window which appears when you select Update in the menu that pops up when you click the arrow icon. The process is similar when you create a new PS. To build the list of child service specifications, work in the area under the “Children” tab. An “Available” column will be shown for all the specifications that are already defined in the catalog and are available for inclusion in the list. The “Included” column shows the service specifications have been included in the child list. Click on an available/included service specification and use the simple arrow buttons between the columns to include or exclude it. You can include all available specifications by clicking on the →| button or empty the list of included by clicking on the |← button.

It is important to notice that the order of the children is significant. It determines the order in which they are processed during state transitions. Select a child and use the up/down buttons on the right to move it up or down in the sequence.

When you have selected and ordered the child specifications, work in the “Characteristic Mappings” tab to create the characteristic mappings. The PS (or CFSS, RFSS) you are working on is the parent. Select one child service at a time from the “Child” drop-down list. You work on input and output mappings, under the respective headings at left and right. Remember, a mapping is a pairing of a parent characteristic and a child characteristic. To add a mapping, select the parent and the child characteristics in the two drop-down fields, and then click the [+] button. To remove a mapping, click the [×] button in the right side of the row. To change a mapping, select it in the list, click the edit button, and the mapping will be shown in the two drop down fields. Change one or the other, as appropriate, and click the [+] button.

When you are satisfied with the complete PS (or CFSS) definition, click the [OK] button to commit it.

Figure 21 **Catalog Management, Update Product Specification**



Resource

The Catalog Management UI for creating/updating product specifications has a radio button called “Resource”. By changing this radio button to “Yes”, product instances based on this specification will be usable as consumable resources (with capacities) from other instances. It is the responsibility of a product instance’s own action workflows to set and modify its capacity. For more details, please read Chapter 10.

Migration

In the UI to create/update product specifications there is a tab named “Migration”. In this tab it is possible to add names/versions of other product that may be migrated into this product. Chapter 9 describes this topic in more detail.

Solution and Queue

The fields named “Solution” and “Queue” in the create/update product specification UI can be to control which queue to be used when spawning workflow jobs to handle service requests that refer to products of this type. The document *HP Service Activator System Integrator’s Overview* provides a detailed description of this topic.

Import and Export of Catalog Content

NOTE	HP Service Provisioner must be running in order to use the **SOMData** utility.

HP Service Provisioner includes a utility, called `SOMData`, to export catalog content (on a development/test system) to a flat XML-formatted files and import it again from that file on the target system, typically as part of solution installation and deployment on a target system. The utilities are found in *$ACTIVATOR_OPT*`/bin`.

The utilities interwork with a running HP Service Activator Workflow Manager, which may run on a different machine from the `SOMData` utility, and need credentials to establish a session with the Workflow Manager. `SOMData` must be called with a number of options, from the following:

`-export`	to export the catalog contents
`-import`	to import the catalog contents
`-filePath`	full path name of the file
`-user`	mandatory, user name for session with workflow manager
`-password`	mandatory, password for session with workflow manager
`-host`	HP Service Activator host (omit if local)
`-port`	HP Service Activator port number (omit if the default port is used)
`-verbose`	generates verbose output
`-help`	outputs usage information

Exactly one of the `export` and `import` parameters must be specified.

There is also a utility, `SOMDataCleaner`, which can be used to delete catalog and service inventory contents. It has the same options as `SOMData`, except instead of choosing between `export` and `import`, you must choose one of:

`-instances`	delete product instances, but leave catalog contents
`-all`	delete products instances and also profiles, product/service specifications, and templates from the catalog

7 Monitoring and Interacting With Running Orders

This chapter is primarily of interest for the runtime operator who will monitor running orders and possibly interact with them.

You can search for service orders which are being processed according to active requests, or for product instances in the service inventory. The same search criteria are used for both, and both are shown in the same way, as tree structured service orders/product instances. When viewing active search orders you can inspect their progress as reflected in the state of each service in the tree.

When processing of a service order calls for manual action, an operator must locate the order, launch the window for the manual action and then complete it. This must be done when the service order enters the *Designed* state (if the instance requires manual user interaction), to make entry to the state complete.

Finally, depending on how a specific solution is implemented, state transition action workflows may require user interaction.

These three topics are covered by the sections in this chapter. For all of them you will work through the HP Service Activator user interface. Normally you start by selecting “Service Order Management” in the “Work Area” menu. This chapter also describes other ways to audit/monitor running orders through the use of “activity lists”, audit records, and annocations.

There is also, primarily for testing purposes, a user interface for entering service order requests. It is described in Chapter 5.

Inspecting Service Orders and Product Instances

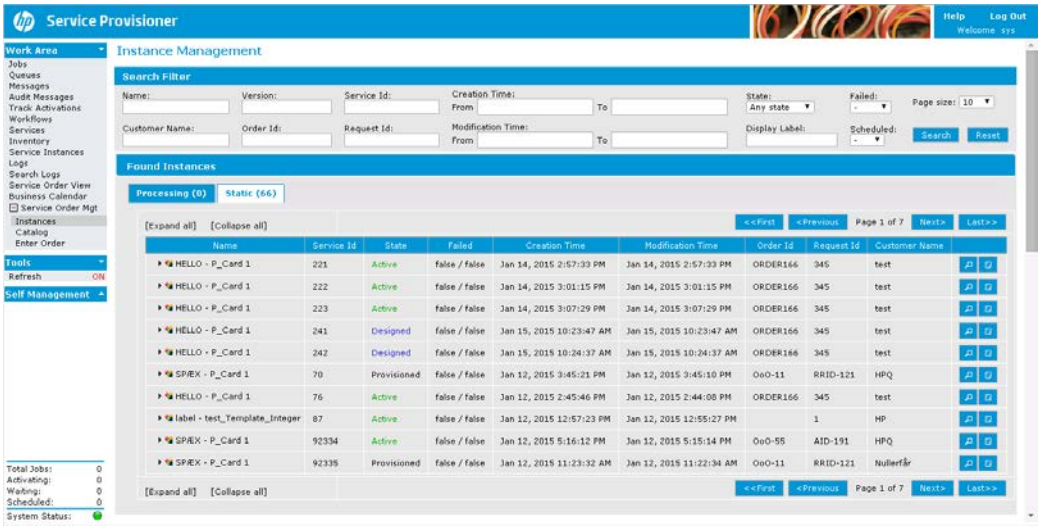
Figure 22 shows the “Instance Management” window which appears when the “Instances” menu item has been selected in the submenu under “Service Order Mgt”. Initially the window shows the search filter and the first page of instances; if there are no processing instances the UI will automatically display the “Static” tab.

You can search by filtering on values attributes of product instances:

- Product name and version, order id, request id, customer name, display label – these are all parameters of the request that created the service order / product instance
- Service id – initially this attribute is not known; it will be returned to the client once it has been assigned by the request which creates a service order / product instance
- Time interval – either when the service order / product instance was created, or when the most recent request to change its state occurred.
- State – a drop-down with the possible states that product instance can be in

- Failed – if the product instance has been flagged as “failed” or not; i.e. if an action workflow has failed
- Scheduled – if the product instance has one or more scheduled requests attached to itself or not

Figure 22 **Instance Management, Search Filter**

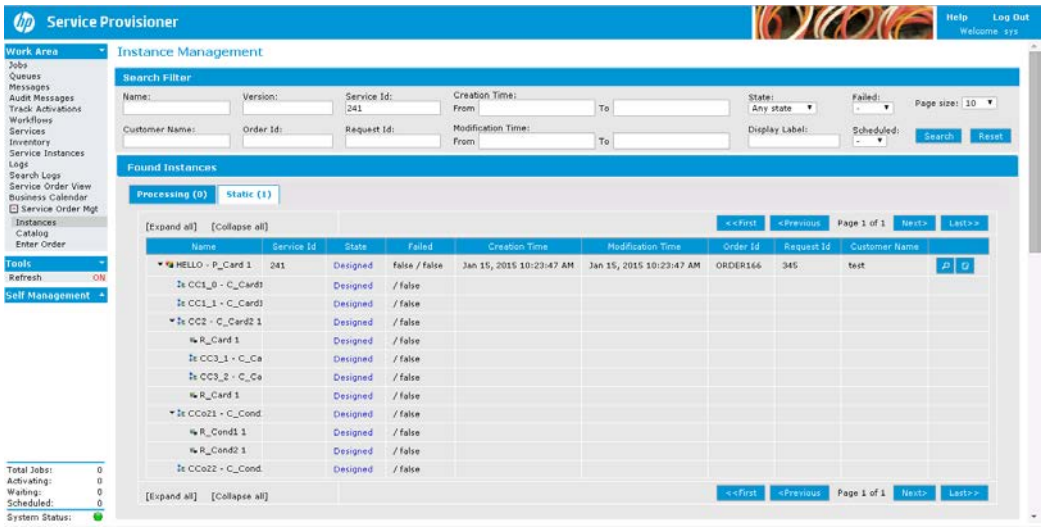


Once the results of a search are retrieved, you can choose by the tabs to view them from a processing perspective or from a static perspective. The static perspective can be used even when there is no order processing activity, typically for active services; you will see the data retrieved from the service inventory about the products that passed your filter. The static view cannot be expected to be up-to-date for service orders/product instances which are subject to ongoing order processing, as the service inventory is only updated at the end of request processing.

The processing perspective is normally used to locate a particular service order. A narrow search filter should be used, for example the service id or request id, to retrieve a single service order. This view is limited to a single page. The static view is intended to show a number of product instances and can be multiple pages long; see Figure 22 for an example.

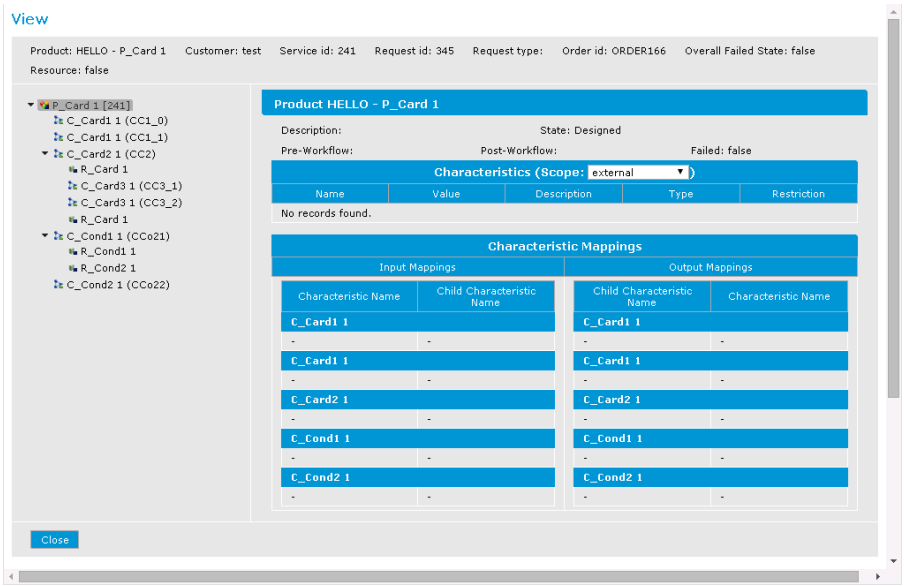
The view will be very similar regardless of the selected perspective. It shows – see Figure 22 for an example – all the product instances that match the search filter (in this example the filter is empty), with their breakdown into CFSs, RFSs, and EWIs on subsequent indented lines. The view summarizes the main attributes and the state of each product or service on a single line. The breakdown can be collapsed or expanded. Figure 23 shows an example of a search by service id; the resulting product instance tree has been expanded revealing its constituent CFSs and RFSs (no EWIs in this example).

Figure 23 **Instance Management, Search for Service Id, Expanded Product Instance Tree**



By clicking the magnifier icon you can launch a popup window, as shown in Figure 24, where details of individual services are shown. This window is similar to the one that appears in the catalog editor, as described in Chapter 6.

Figure 24 **Instance Management, View Product Instance**



Performing Manual Design

When the user interaction flag is active on a service order, all state transitions into the *Designed* state require a manual action that can change the design, i.e. the tree structure composition and even the characteristics and corresponding mappings of each product/service in the tree. For this purpose you will use the “Design” UI.

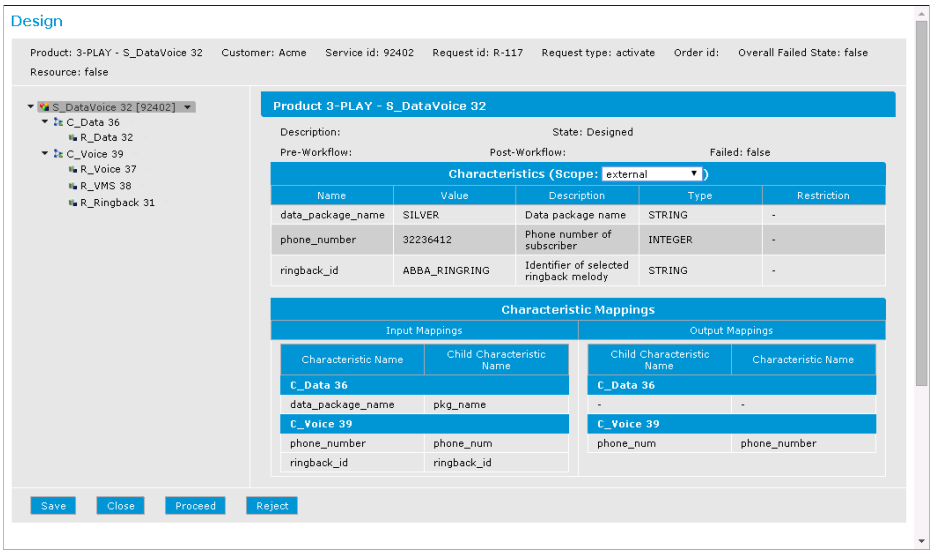
The “Design” UI can be launched by clicking the magnifier icon shown to the right of the product instances shown in the “Instance Management” UI. Alternatively, the “Design” UI can be launched from the “SOM” queue; this queue is found by clicking the “Queues” menu item and

then on the “SOM” tab. If there are no processes waiting for manual user interaction, there may not be any “SOM” tab. To bring up the “Design” UI, right-click on the job in the “SOM” queue and select the “Interact with job” menu item. An example of the manual “Design” window is shown in Figure 25.

In this window you can edit the entire tree structure of the product instance, in the same way as you can edit its specification in the catalog (see Chapter 6). You can add or remove CFSs, RFSs, and EWIs from the tree. CFSs, RFSs, or EWIs that you add must be picked from the catalog. You can even add characteristics, with values and mappings, to services as needed. The main idea is to allow services to be added to a product dynamically.

NOTE — If the user does not have administrative privileges (or if the product instance is not in state *Designed*) the UI will be opened in read only mode.

Figure 25 **Service Order Details, Manual Design**



When the manual design is complete, press the `[Proceed]` button to allow the overall automated service order process to continue. If more work is needed before letting the process continue, press the `[Save]` or `[Close]` button and return to the order at a later time.

Pressing the `[Reject]` button will cause the transition to fail and the order will be canceled.

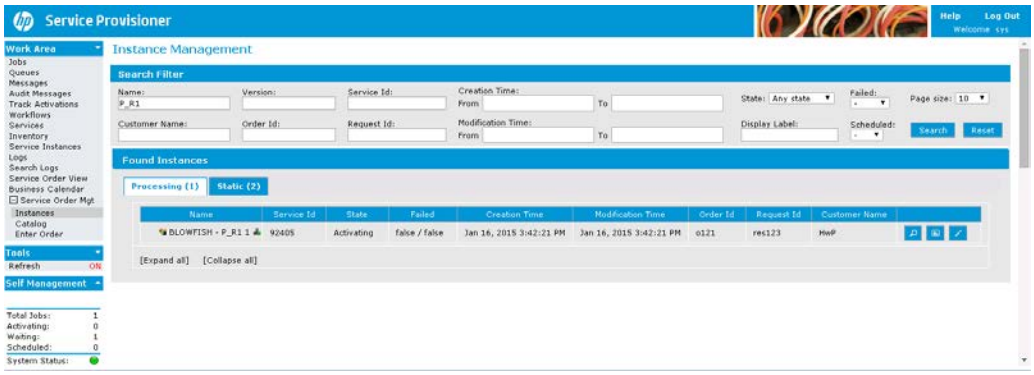
Interacting with State Transition Action Workflow Jobs

State Transition action workflow jobs are written for specific solutions. They are not part of the HP Service Provisioner framework product, so their behavior cannot be described here. But they may include user interactions, to be launched via the HP Service Activator Jobs as described immediately above. This could be to ask the user to make a choice, complete manual work, or enter necessary data. An example of how this could be used for integration with Trueview Inventory with the user acting as intermediary was given in Chapter 2 in the Section “Manual Design and Assign”.

If a processing order requires user interaction due to an `AskFor` workflow node in one of the action workflows, the “Instance Management” UI will display a small icon (resembling a human operator) in the “Name” column; an example of this is shown in Figure 26. In this way, the operator can easily see whether a process is stuck because it is waiting for a user to interact.

More information and specific instructions must be provided per solution.

Figure 26 **Instance Management, Waiting for User Interaction**



Activities

Whenever an activity takes place for a product instance, HP Service Provisioner adds an “activity” data record to an “activity list”. The activity list will be sent as part of the asynchronous responses that can be emitted from Service Provisioner. Only the 200 newest activities will be stored; older activities are silently deleted.

The following events result in activities being added to the activity list:

Conditional removal of child instance	This activity indicates that Service Provisioner has stripped a child instance from the product instance tree as a result of the specified inclusion/exclusion conditions; please see the Section “Conditional Child Removal” on page 31 for details.
Force operation	Indicates that the state of a product instance has been forcefully set or that a product instance has been forcefully deleted. Force operations are described in the Section “Force Operations” on page 29.
Modify add or modify delete	Such an activity is written when a child instance is added to or delete from an existing product instance as a result of a “modify-add” or “modify-delete” operation (or as a result of a delta operation).
Schedule	This activity is written when a fulfillment process results in a service request being stored in the scheduler for this product instance.
State transition	Indicates that the fulfillment process has completed a transition between two states.
Workflow action	This activity is stored for every action workflow that is executed. The result code as well as the result text are stored as part of this activity.

An example of an activity list for a product instance that was first activated and then modified is shown below:

```
<Activities>
  <Activity id="0" type="STATE_TRANSITION" timestamp="2015-01-21 10:26:48.231"
            operation="activate" stateTransitionName="check" startState="Initial"
            endState="Checked" >
  </Activity>
  <Activity id="1" type="STATE_TRANSITION" timestamp="2015-01-21 10:26:48.468"
            operation="activate" stateTransitionName="design" startState="Checked"
            endState="Designed" >
  </Activity>
  <Activity id="2" type="STATE_TRANSITION" timestamp="2015-01-21 10:26:48.745"
            operation="activate" stateTransitionName="reserve"
            startState="Designed" endState="Reserved" >
```

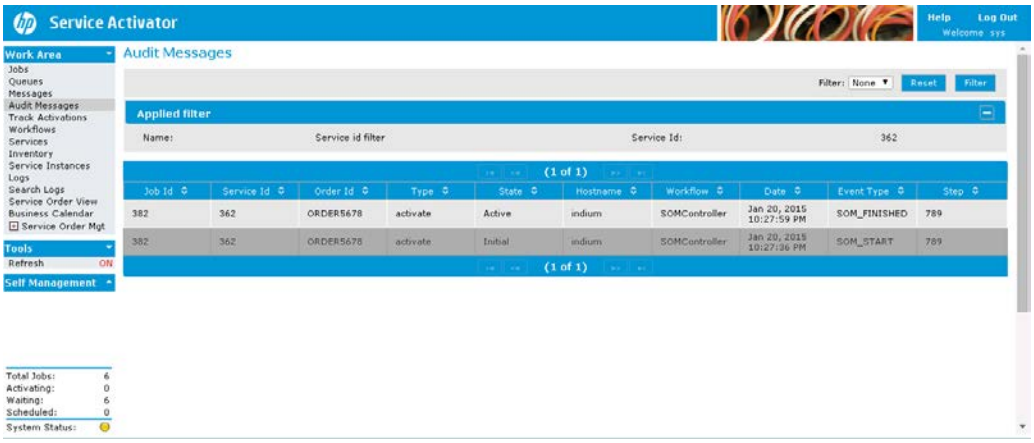
```
  </Activity>
  <Activity id="3" type="STATE_TRANSITION" timestamp="2015-01-21 10:26:49.071"
            operation="activate" stateTransitionName="provision"
            startState="Reserved" endState="Provisioned" >
  </Activity>
  <Activity id="4" type="STATE_TRANSITION" timestamp="2015-01-21 10:26:49.671"
            operation="activate" stateTransitionName="activate"
            startState="Provisioned" endState="Active" >
    <Workflow itemType="CFS" itemName="C_DSL" itemVersion="3" path="0"
              postWF="false" rollback="false" wfName="Delay" result="0"
              resultText="OK" timestamp="2015-01-21 10:27:13.057" />
  </Activity>
  <Activity id="5" type="STATE_TRANSITION" timestamp="2015-01-21 10:27:49.091"
            operation="modify" stateTransitionName="deactivate"
            startState="Active" endState="Provisioned" >
  </Activity>
  <Activity id="6" type="STATE_TRANSITION" timestamp="2015-01-21 10:27:49.435"
            operation="modify" stateTransitionName="deprovision"
            startState="Provisioned" endState="Reserved" >
  </Activity>
  <Activity id="7" type="STATE_TRANSITION" timestamp="2015-01-21 10:27:49.844"
            operation="modify" stateTransitionName="release" startState="Reserved"
            endState="Designed" >
  </Activity>
  <Activity id="8" type="STATE_TRANSITION" timestamp="2015-01-21 10:27:50.211"
            operation="modify" stateTransitionName="terminate"
            startState="Designed" endState="Terminated" >
  </Activity>
  <Activity id="9" type="STATE_TRANSITION" timestamp="2015-01-21 10:27:50.522"
            operation="modify" stateTransitionName="retire"
            startState="Terminated" endState="Retired" >
  </Activity>
  <Activity id="10" type="MODIFY_DELETE" timestamp="2015-01-21 10:27:50.703" >
    <DeleteItem itemType="CFS" itemName="C_CPE" itemVersion="2"
                itemLabel="Modem/Router" path="1" />
  </Activity>
  <Activity id="11" type="STATE_TRANSITION" timestamp="2015-01-21 10:27:50.709"
            operation="modify" stateTransitionName="modify" startState="Active"
            endState="Active" >
    <Workflow itemType="CFS" itemName="C_DSL" itemVersion="3" path="0"
              postWF="false" rollback="false" wfName="Delay" result="0"
              resultText="OK" timestamp="2015-01-21 10:28:06.691" />
  </Activity>
</Activities>
```

In this example, the two state transitions *activate* and *terminate* result in a single action workflows being executed. In addition, the activity list shows that the *modify* operation has resulted in a single CFS being retired and then deleted.

Audit

HP Service Provisioner is capable of storing audit records that can be used to examine historical orders; this functionality piggybacks on HP Service Activator's Audit workflow node and "Audit Message" UI. Figure 27 shows the "Audit Messages" UI where the messages have been filtered by a service id (362).

Figure 27 **Audit Message UI**

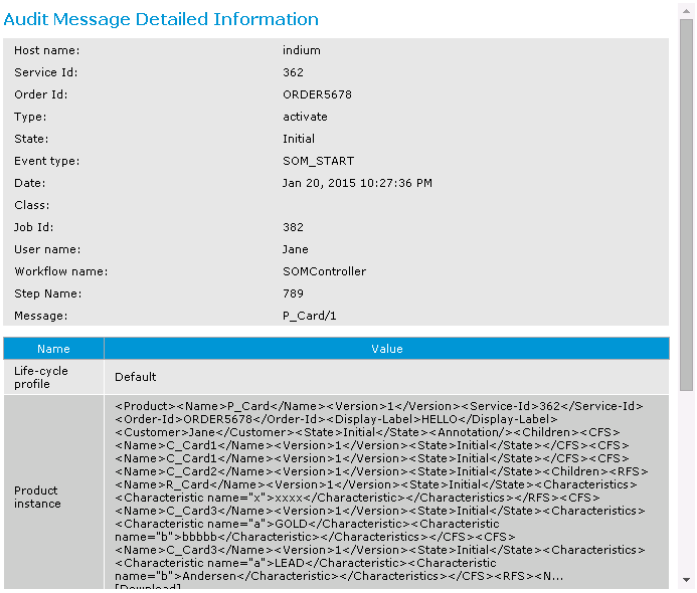


The following audit events are supported:

SOM_FORCE	Written when a "force" operation takes place; i.e. the state of a product instance has been forcefully set to a new value.
SOM_START	Written when a new fulfillment process is about to commence for a product instance. The service request has been successfully validated when this audit record is written.
SOM_FINISHED	Written when a fulfillment process is finished.
SOM_SCHEDULE	Written when a service request has been scheduled to take place in the future for a product instance. When the schedule is triggered, another *SOM_START* event will be written (because a service request emitted by the scheduler is treat like any other service request).
SOM_ERROR	Written in case an error occurs for an ongoing fulfillment process.
SOM_RETIRE	Written when product instance is retired and the product instance deleted.

Figure 28 shows the "Detailed Information" UI for audit message; in this example, an audit record with event type *SOM_STATE* is inspected for service id 362.

Figure 28 Audit Message Detailed Information UI



The meaning of each of the fields in an audit record is listed below:

Host name	Contains the name of the host on which the fulfillment process was running when this audit record was written.
Service Id	Contains the service id of the product instance for which a fulfillment process has been running.
Order Id	The order id of the service request.
Type	The operation that is requested (also known as the “request type”).
State	The current fulfillment state of the product instance.
Event type	The type of this audit event (see the description of the supported event types earlier in this section).
Date	The date/time when this record was written.
Job Id	The job id of the workflow job that was executing the fulfillment operation for this product instance.
User name	The customer name, defined in the service request.
Workflow name	The name of the workflow from which the audit record was written. In Service Provisioner all audit records are written from the `SOMController` workflow.
Step name	This field contains the request id of the service request.
Message	Contains the name and version of the product instance using a slash as the separator.
Life-cycle profile	The name of the fulfillment life-cycle profile name used for this product instance.
Product instance	An XML representation of the product instance, including all its child instances (RFSs, CFSs, and EWIs).
Display label	The value of the display label which is assigned to the product instance from the service request.

Activities	An XML representation of the activities that have taken place for this product instance. The syntax of this XML structure is the same as the "activity list" XML structure that is part of the asynchronous messages emitted from Service Provisioner.
Service request	An XML representation of the latest service request received via the northbound interface (or from the scheduler). This field is only written if the audit type in *SOM_START*.

IMPORTANT	Audit records are intended to be used to do "post mortem" auditing of fulfillment process; i.e. to see what has happened in the past and when it happened. It is *not* designed to be used to track the progress of on-going processes.

Annotations

In HP Service Provisioner it is possible for an operator to add annotations (small text messages) to a running fulfillment process that can be read/edited by other operators. Annotations are purely for informative purposes; they have no semantics. There are two ways to add/read/update annotations in Service Provisioner:

- **From the Instance Management UI** (see Figure 26)**:** In the row on the running process (in the "Processing" tab) click the button with the pencil icon. This will bring up a small window in which it is possible to read and modify the current annotations, see Figure 29. To save modifications done to the annotations, click the `[OK]` button.

- **From the HP Service Activator's Jobs or Queues UI:** In the row representing the running fulfillment process, right-click to bring up a context menu. Then click on the menu item named `Annotate`. This will bring up the window in which the annotations are displayed and can be modified. Figure 30 shows how to access the annotations UI from HP Service Activator's "Jobs" UI.

Figure 29 UI for Reading and Writing Annotations

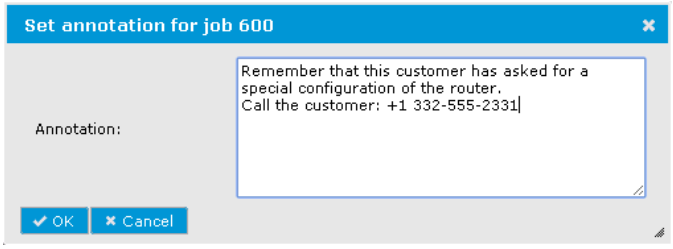
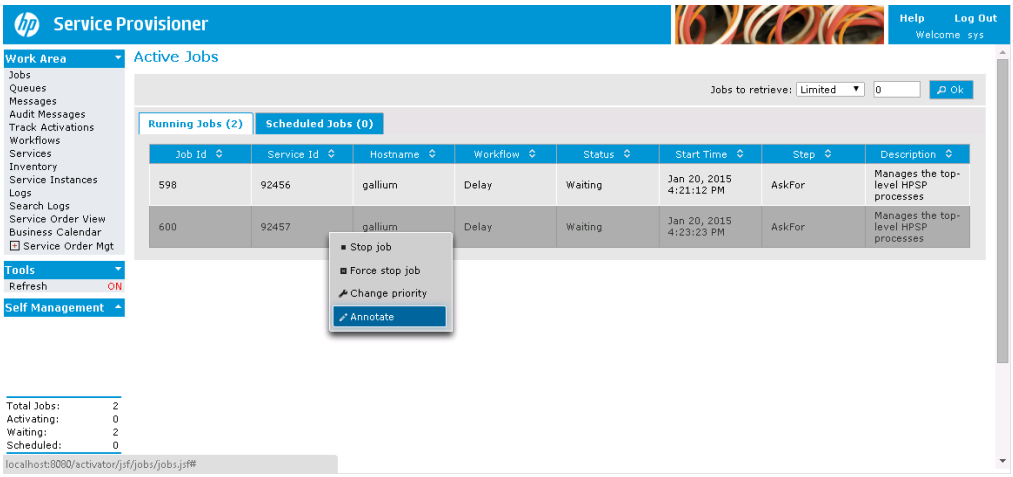


Figure 30 Accessing the Annotations UI from Jobs View



8 Processes for Resource Facing Services

This chapter is primarily of interest for the system integrator who will implement the technical processes.

For each catalog item an action workflow (or two action workflows, pre- and post-workflows, if the catalog item has children) is required to carry out the solution specific work. In order to keep a clear separation between what is customer facing and what is resource facing, only action workflows associated with RFSs should interact with external systems (resource inventory, network equipment, etc.). Any action workflow could, in principle, interact with external system, but such a design choice is discouraged.

Different algorithms must be implemented for all the state transitions which require some action. If the same workflow is configured for all state transitions the workflow must begin with branching logic based on the state transition name. Otherwise, if different action workflows are configured for different state transitions, it is possible to keep a more clear separation of the roles of the different action workflows. Which of the two options make most sense will depend on the solution.

All the inputs to the action workflow are specified below as part of the workflow contract. In addition to the start state and end state (and state transition name) the main inputs are the characteristics for the instance (PI, CFS, RFS, or EWI), with the values they have after input mapping has been performed.

In addition to the inputs the action workflow may look up information in service or resource inventory, it may interact with operators to ask for more information, and it general it will be able to interact with external systems. For the latter kind of interaction the means that are generally available to HP Service Activator workflows can be used. Consult *HP Service Activator, System Integrator's Overview* for more information.

Information can be passed on to subsequent parts of the process from a state transition algorithm in an action workflow by updating values of the case-packet variables holding the characteristics.

For a description of the typical work in state transition algorithms, please read the Sections "States" and "State Transitions" in Chapter 2.

Action Workflow Contract

All action workflows must have contracts defined; otherwise they cannot be used by the HP Service Provisioner workflows. The action workflow contract's input parameters are described in Table 3.

Table 3 Action Workflow Contract, Input Parameters

Input Parameter	Type	Description
parentJobId	Integer	Used to hold the job id for the workflow that spawned this action workflow. Since the action workflow is, technically speaking, running as a "macro job" it is currently not used.
syncToParent	Boolean	Used to indicate to the action workflow whether it needs to synchronize back to the caller workflow upon completion. In the current version of HP Service Provisioner, this parameter is always set to "false".
rollback	Boolean	Indicates whether or not this action workflow is called as part of a rollback operation. If set to "true" this is a rollback operation; otherwise, it is a "roll forward" operation.
name	String	Contains the name of the instance that is currently being processed.
version	String	Contains the version of the instance that is currently being processed.
parameters	Object	Contains a map of all characteristics that are visible to this instance. The characteristics can be accessed using the standard HP Service Activator syntax for retrieving values from maps. Example: `parameters{"bandwidth"}`
startState	String	Contains the name of the start state for the state transition that is currently taking place.
endState	String	Contains the name of the end state for the state transition that is currently taking place.
requestType	String	Contains the name of the request type (a.k.a. operation) that triggered the current fulfillment process (e.g. *activate*, *design*, *reserve*, etc.).
requestId	String	Request identifier sent as part of service request.
rollbackStrategy	String	The selected error-handling strategy: ATOMIC, BEST_EFFORT, or PARTIAL
customer	String	Contains the name of the customer.
post	Boolean	If set to "true" this workflow was called as a "post-workflow"; otherwise it was called as a "pre-workflow".
stateTransition	String	Name of the current state transition; in the action workflow the use of this parameter may replace the use of the `startState` and `endState` parameters which can lead to a simpler workflow logic because only a single string needs to be tested instead of two (the start state and the end state).

The output parameters of the action workflow contract are described in Table 4.

Table 4 Action Workflow Contract, Output Parameters

Output Parameter	Type	Description
`Result`	Integer	Used to pass a result code back to the caller workflow. Possible result codes: 0 = ok 1 = error, consistent 2 = error, inconsistent
`resultText`	String	Used to pass a result text back to the caller workflow.
`SOM_INSTANCE`	Object	*For internal use.* This parameter *must* be passed back.

In addition, a number of system case-packet variables are implicitly passed to the action workflow; of interest to HP Service Provisioner are the following system case-packet variables:

- `SERVICE_ID` – this case-packet variable contains the identifier (the service id) of the product instance that is currently being processed.
- `SOM_INSTANCE` – this case-packet variable holds a reference to the product instance (i.e. the root element of the "product instance tree"). This variable must never be modified by action workflows.
- `SOM_PATH` – this case-packet variable contains a pointer to the instance within the "product instance tree" that is currently being processed. This variable must never be modified by action workflows.

For examples of action workflows with workflow contracts, see the five sample RFS action workflows (their names starting with `R_`) that are part of the "SOM Demo" solution; see also the Section "Deploy SOM Demo Solution" on page 38.

Reading Characteristic Values from Action Workflows

There are two ways to read values from characteristics into case-packet variables in an action workflow:

- **Directly from the `parameters` HashMap:** This is done by using the standard HP Service Activator syntax for reading values from maps using curly braces. The advantage of using this syntax is that characteristic values can be accessed from *any* workflow node.
  Example: `parameters{"package"}`
- **Using the `SOMGetCharacteristics` workflow node:** When using this workflow node it is possible to access multiple characteristic values as well as multiple characteristic *shadow* values (i.e. previous values) in a single operation. This workflow node is described in more detail in Chapter 12.

An example of the use of the `SOMGetCharacteristics` workflow node is shown below:

```
<Process-Node>
  <Name>SOMGetCharacteristics</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
                                          SOMGetCharacteristics
    </Class-Name>
    <Param name="param2" value="ringback_id"/>
    <Param name="param3" value="phone_num"/>
    <Param name="shadow2" value="true"/>
    <Param name="shadow3" value="true"/>
    <Param name="variable0" value="ringback_id"/>
    <Param name="variable1" value="phone_num"/>
    <Param name="variable2" value="shadow_ringback_id"/>
    <Param name="variable3" value="shadow_phone_num"/>
  </Action>
</Process-Node>
```

In this example the *current* values of the characteristics `ringback_id` and `phone_num` are assigned to two case-packet variables by the same names. In addition, the *previous* values of the same two characteristics are assigned to the two case-packet variables `shadow_ringback_id` and `shadow_phone_num`, respectively.

NOTE The use of shadow characteristics only makes sense when used in the context of *modify* operations.

Testing for Modified Characteristics

HP Service Provisioner also includes a workflow rule node that can be used to test whether some or all characteristics of the currently processing instance have been modified; the name of the workflow node is `SOMCharacteristicsModified`. An example of its use is given below:

```
<Rule-Node>
  <Name>CharModified?</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
                                          SOMCharacteristicsModified
    </Class-Name>
    <Param name="modified_characteristics" value="modifiedList"/>
    <Param name="param0" value="phone_num"/>
    <Param name="param1" value="ringback_id"/>
  </Action>
</Rule-Node>
```

In this example, the workflow node will test whether any of the two characteristics `phone_num` and `ringback_id` have been modified for the currently processing instance; if no `param` parameters are given the workflow node will test *all* the instance's characteristics for any changes. If any of the two characteristics have been modified, the workflow execution will continue along the "true" branch; otherwise the workflow execution will continue along the "false" branch. If there are any characteristics that have been modified, the workflow node parameter `modified_characteristics` can be used to return a list of those characteristic names that have been modified.

This workflow node is described in more detail in Chapter 12.

Writing Characteristic Values from Action Workflows

When an action workflow has finished its tasks it will typically need to store some values into the RFS characteristics. For this purpose, the `SOMAssignResult` workflow node is available. This workflow takes any number of case-packet variables as input and stores them in the characteristics of the currently processing instance. By default they are stored in characteristics of the same name as the case-packet variables. However, it is also possible to store case-packet variables in differently named characteristics.

An example of the use of the `SOMAssignResult` workflow node is shown in the following:

```
<Process-Node>
  <Name>SOMAssignResult</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.SOMAssignResult
    </Class-Name>
    <Param name="param0" value="constant:bandwidth"/>
    <Param name="variable0" value="bw"/>
    <Param name="variable1" value="card"/>
  </Action>
</Process-Node>
```

In this example the case-packet variable `bw` is stored in the characteristic named `bandwidth`, and the case-packet variable `card` is stored in the characteristic by the same name.

Interaction with Subscription Repository

IMPORTANT

In the vast majority of cases, there should be no need for action workflows to interact directly with Subscription Repository; all communication with Subscription Repository should be left to HP Service Provisioner. The input and output characteristics should suffice in most action workflows.

It is possible to interact with Subscription Repository through the `SRModule` described in Chapter 12 on page 95. To interact with Subscription Repository, one of the following workflow nodes can be used:

- **`SOMCreateProductInstance:`** Creates a product instance based on a product specification (the product specification object can be retrieved using the `SOMValidateRequest` workflow node).
- **`SOMDeleteProductInstance:`** Deletes a product instance in Subscription Repository.
- **`SOMGetProductInstance:`** Retrieves a product instance from Subscription Repository.
- **`SOMUpdateProductInstance:`** Updates a product instance in Subscription Repository.
- **`SOMUpdateProductInstanceState:`** Updates the state of a product instance state in Subscription Repository.

All these workflow nodes should be used with utmost care (except `SOMGetProductInstance`). The workflow nodes are described in more detail in Section “Workflow Nodes for Product Instance Access” on page 107.

Integration with Trueview Inventory

If the solution uses Trueview as the inventory system, the `TrueviewModule` must be configured in the *`$ACTIVATOR_ETC`*`/config/mwfm.xml` configuration file.

To send requests to Trueview (by means of the `TrueviewModule`) the workflow node `TVWSRequest` can be used. The “SOM Demo” solution comes with three sample workflows that demonstrate how to interact with Trueview:

- `SOM_Demo_customer` – creates a customer object in Trueview
- `SOM_Demo_customer_delete` – deletes a customer object in Trueview
- `SOM_Demo_customer_get` – retrieves a customer object from Trueview

Please read the description of the `TVWSRequest` workflow node (Chapter 12 page 113) for a full description of how to use it (including an example).

9 Delta Operations

In the previous version of HP Service Provisioner a product instance would always have a fixed number of child instances, except in cases where a product instance was modified using a manual design step (see Section “Manual Design and Assign” on page 28). So, once the product instance tree was created it was not possible to add or delete CFSs at a later time.

As described in Chapter 2 two new mechanisms for adding/deleting CFSs to/from an existing product instance have been added in HP Service Provisioner 7.0. These are:

- **Modify-add/modify-delete operations:** These operations can be used to add/delete CFSs to/from an existing product instance by sending a service request containing the CFSs to be added/delete and the positions in the product instance tree in which they are to be added/deleted. Read the Section “Modify Operations” on page 25 for an introduction to “modify-add” and “modify-delete” operations.

- **Conditional child removal:** This is a mechanism that can be used to remove child instance branches or nodes from a product instance based on whether or not some characteristic values meet a set of predefined conditions. This topic is explained in more detail in the Section “Conditional Child Removal” on page 31.

In addition to these two new mechanisms, HP Service Provisioner version 7.0 has support for even more powerful mechanisms for creating and modifying product instances which allows full control over the number of child CFSs for a product instance. These new mechanisms for modifying product instances is referred to as *delta operations*.

With *delta operations* it is possible for the northbound system in a *modify* operation to specify the structure of the *new* product instance tree down to the CFS level (i.e. not including RFSs or EWIs). HP Service Activator will then pass the *new* product instance tree and the *existing* product instance tree through its *Delta Engine* which calculates the differences between the two trees as well as the sequence of operations that are required to get from the old product instance tree to the new product instance tree.

This is a *completely* different approach to modify operations compared to the modify mechanisms that are described in Chapter 2. Instead of having to specify the *changes* that are to be done to an existing product instance (down to a rather detailed technical level), delta operations makes it possible to specify the desired end result. It is then up to the Delta Engine to calculate how to reach that end result.

Variable Cardinality CFSs

If all product instances (based on the same specification) would *always* have the same number of CFSs (and RFSs/EWIs), then it would not be possible to specify that one or more CFSs need to be added (or deleted).

In order to support that a product specification can be instantiated into product instances with a variable number of occurrences, HP Service Provisioner 7.0 introduces support for *variable cardinality* CFSs; this means that it is possible for CFSSs to define (in the catalog) the *minimum*

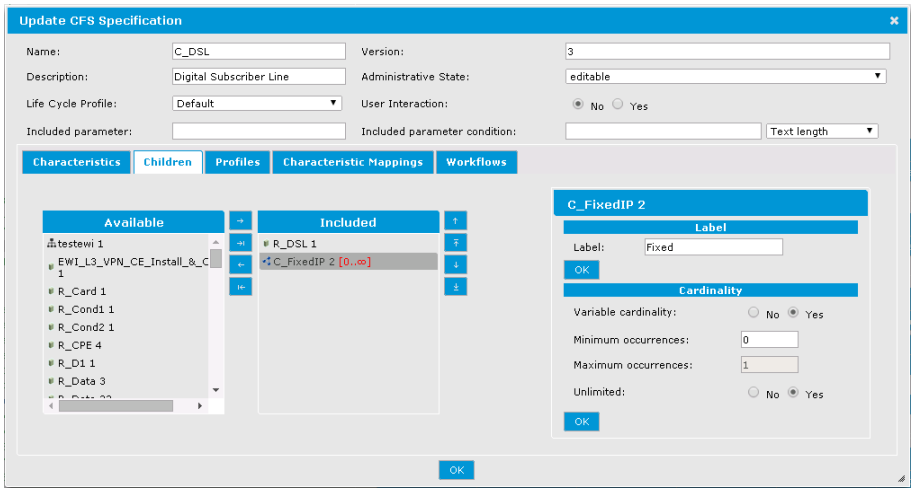
and the *maximum* number of occurrences there must be for CFSs instantiated from CFSSs. The following rules apply when specifying the minimum and maximum number of occurrences for CFSs:

- **Minimum occurrences:** Must be set to a value equal to or greater than 0. If not specified, Service Provisioner will assume that the value is 1.

- **Maximum occurrences:** Must be set to a value equal to or greater than 1 and always greater than or equal to the value specified for minimum occurrences. The value *unlimited* is also supported.

Setting the minimum and maximum number of occurrences for CFSs is done in the catalog. Figure 31 shows an example of an “Update CFS Specification” UI where the “Children” tab is selected. To specify the minimum and maximum number of occurrences, click on the child CFSS in the “Included” list. This will bring up the “Label” and “Cardinality” areas shown in the lower-right corner. Then change the value of the “variable cardinality” radio button to `Yes` and fill in the two areas labeled “Minimum occurrences” and “Maximum occurrences”. If an unlimited number of maximum occurrences is desired (which is the case in this example), then change the value of the “Unlimited” radio button to `Yes`. Finally, click the `[OK]` button in the “Cardinality” area to apply the values for the minimum and maximum number of occurrences in memory. To save the values in the catalog, click the `[OK]` button centered in the bottom of the “Update CFS Specification” UI.

Figure 31 **CFSS with Child with a Variable Number of Occurrences**

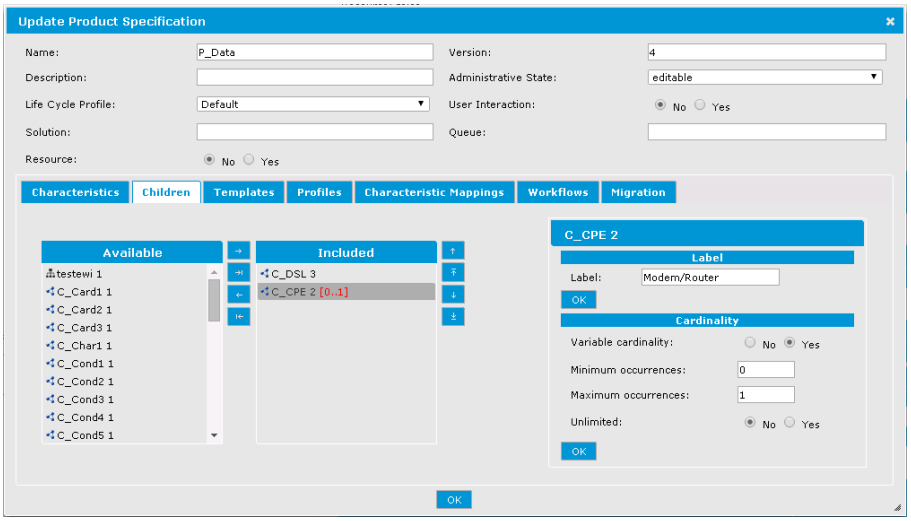


It is also possible to specify a label for child specification. This label will serve as the default label value if not overwritten explicitly by the service request. To specify a label, enter a value (string) in the “Label” field and click the `[OK]` button shown just below the “Label” field.

NOTE Labels can be specified for CFSS children, RFSS children, as well as EWIs children whereas variable cardinality can only be specified for CFSS children.

Only CFSSs and product specifications can contain child CFSSs with a variable number of occurrences. Figure 32 shows an example of a product specification that contains two CFSSs. The CFSS “C_DSL” has not specified the minimum or the maximum number of occurrences; this means that this CFSS is *mandatory*. The other CFSS, “C_CPE”, has minimum occurrences set to zero and maximum occurrences set to 1; hence, this CFSS is *optional*.

Figure 32 **Product Specification with a Mandatory and an Optional Child CFSS**



The “Catalog Management” UI displays the minimum and maximum number of occurrences of its child CFSSs using the following syntax:

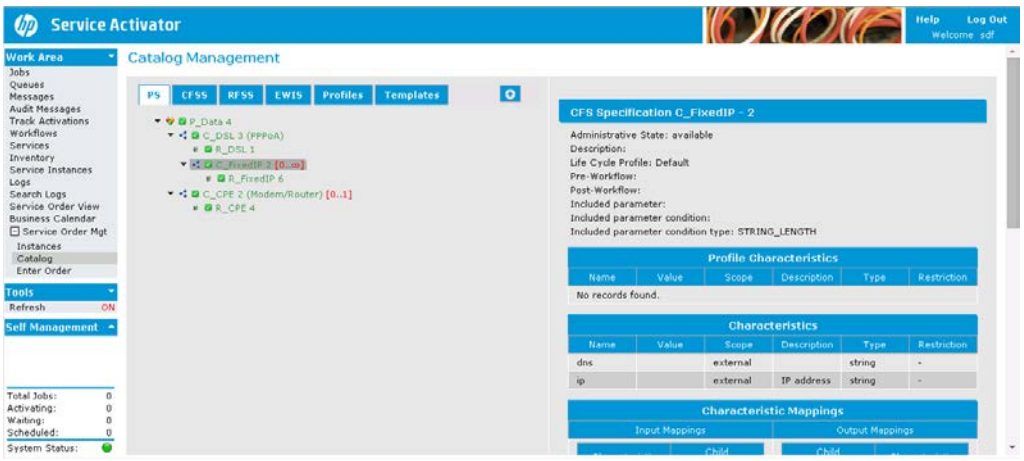
- `[`*`minimum occurrences`*`..`*`maximum occurrences`*`]`

  — If both minimum and maximum occurrences are set to 1, nothing will be shown

  — If the maximum number of occurrences is *unlimited*, then the mathematical infinity symbol (∞) is shown

An example of a product specification where two out of three child CFSSs have variable cardinality is illustrated in the screenshot of the “Catalog Management” UI shown in Figure 33. The CFSS “C_DSL” is mandatory in this example. The CFSS “C_FixedIP” is displayed with a `[0..∞]` suffix to indicate that it can occur from zero to an infinite number of times. The CFSS “C_CPE” is suffixed with `[0..1]` to indicate that it is optional.

Figure 33 **Product Specification containing CFSSs with Variable Number of Occurrences**



Requirements

In order to make use of HP Service Provisioner's support for a variable number of CFSs in product instances, as well as the thereof following support for delta operations, the following capabilities are required by the northbound system:

- The northbound system needs to be aware of PSs as well as CFSSs; this includes knowledge of the characteristics of CFSSs (in order to be able to assign values to these).

- The northbound system should also be able to set the labels of CFSs so that they can be used for identification in subsequent *modify* operations. CFS label do not necessarily need to be unique; however, if the north-bound system needs to be able to uniquely identify CFSs, it is recommended that the northbound system generates unique labels for the CFSs.

Partial and Full Instance Trees

As mentioned in the previous section, the northbound system needs to have knowledge of PSs as well as CFSSs in order to make use of product specifications with CFSS with variable cardinality. In order for the northbound system to create a product instance based on a specification with children with a variable number of occurrences, the northbound system needs to send an instance tree structure to HP Service Provisioner with the product as the root and a number of CFS branches. Such an instance tree structure is referred to as a *partial instance tree*.

When HP Service Privisioner receives a service request containing a partial instance tree, it will pass the partial instance tree through an algorithm that does the following:

- It *validates* the partial instance tree to ensure that its CFSs match the CFSSs (and their minimum/maximum occurrences) specified in the catalog.

- It *expands* the partial instance tree by inserting RFSs and EWIs in the positions dictated by the structure of the product specification tree.

The instance tree that results from this algorithm is called a *full instance tree* because it contains everything (product instance, CFSs, RFSs, and EWIs).

IMPORTANT

A *partial instance tree* is an instance tree containing a product instance with all its CFS children; a partial tree cannot contain RFS or EWI children.

A *full instance tree* is an instance tree containing a product instance with all its CFS, RFS, and EWI children; a full instance tree is automatically generated by HP Service Provisioner based on a partial instance tree and the product specification tree.

After the full instance tree has been generated, HP Service Provisioner will run the fulfillment processes for the entire tree in a depth-first manner, exactly like it is done for all other product instance trees. Please read Chapter 2 on page 19 for a detailed description of HP Service Provisioner's fulfillment processes.

Northbound API

This section describes the northbound API extensions that have been introduced in HP Service Provisioner 7.0 to support variable cardinality CFSs and delta operations. This is an extension to the API described in the Section "Northbound API" on page 39.

Requests received on the northbound interface to create a new product instance or modify an existing product instance (with variable cardinality CFSs) must – in addition to the content described in the Section "Northbound API" on page 39 – contains *one or more* CFS elements with the following contents for each CFS element:

CFS name and version	Identifies the CFSS in the catalog. Mandatory.
CFS label	A label applied by the northbound system that can be used to identify a CFS. Labels do no need to unique. Optional.
Characteristics	List of names and corresponding values of CFS characteristics. Optional.
Template name and version	Identifies a template from which to take values to assign to the characteristics of the CFS instance. Optional.
CFS instance	One or more CFS instances following the syntax described in this list; i.e. a tree of CFSs can be specified in this way. Optional.

A *partial instance tree* is equivalent to a product name and version along with its CFS instance trees specified in a service request. Upon reception of a service request containing a partial instance tree HP Service Provisioner will validate it and expand it to form the *full instance tree* before commencing the fulfillment processes.

Example **WS/SOAP Request Containing CFS Instances**

An example of a SOAP request to activate an instance of the service shown in Figure 33 is shown below:

```
<Request>
  <RequestType>activate</RequestType>
  <RequestId>3226</RequestId>
  <ProductName>P_Data</ProductName>
  <ProductVersion>4</ProductVersion>
  <Customer>Acme</Customer>
  <CFS>
    <Name>C_DSL</Name>
    <Version>3</Version>
    <CFS>
      <Name>C_FixedIP</Name>
      <Version>2</Version>
      <Label>WebServer</Label>
      <Characteristic name="ip">10.20.30.55</Characteristic>
    </CFS>
  </CFS>
  <CFS>
    <Name>C_CPE</Name>
    <Version>2</Version>
  </CFS>
</Request>
```

When HP Service Provisioner receives this service request it will – based on the contents of the request – generate the following partial instance tree:

```
P_Data.4 --|
           |-- C_DSL.3 --|
           |             |-- C_FixedIP.2 (WebServer)
           |
           |-- C_CPE.2
```

When this partial instance tree has passed through the “validate and expand” algorithm, the result will be the following full instance tree which will then be run through the fulfillment processes (the added parts are marked in green):

```
P_Data.4 --|
           |-- C_DSL.3 --|
           |             |-- R_DSL.1
           |             |
           |             |-- C_FixedIP.2 (WebServer) --|
           |                                           |-- R_FixedIP.6
           |
           |-- C_CPE.2 --|
                         |-- R_DSL.1
```

Delta Operations

As explained in the beginning of this chapter, product instances that exist in HP Service Provisioner can be modified using so-called *delta operations*. A delta operation is identical to a *modify* operation with the exception that the *modify* service request must contain a partial tree (i.e. a tree structure with a product instance and all its CFS children).

For the *Default* life-cycle profile the states that support *modify* operations are:

Designed	When in this state, the operation is actually called *amend*; this operation is semantically identical to a *modify* operation.
Provisioned, Active	Both of these states support a *modify* operation. These states also have support for a *test* operation (that leads back to the same state); the *test* operation cannot be used for delta operations.

The other two life-cycle profiles supported by HP Service Provisioner both support *modify* operations for product instances in the *Active* state.

When Service Provisioner receives a *modify* service request (with the service id of the product instance to be modified) containing a partial instance tree, it will do the following:

- The old product instance tree will be read into memory based on the service id from the service request.
- The received partial instance tree will be validated and expanded into a full instance tree; this tree will be the new instance tree
- Finally, HP Service Provisioner will pass the old and the new instance trees to its delta engine which will calculate the differences between the two trees; the differences will be expressed using a so-called *delta tree*.

When the delta engine compares two nodes (one from the old instance tree and one from the new instance tree) it will consider the two nodes to match if the following conditions are met:

- The types of the nodes are identical (i.e. both are CFSs, RFSs, or EWIs)
- The names and versions of the nodes are identical
- The labels of the two nodes are identical

HP Service Provisioner’s algorithm to calculate the delta tree is designed in a way that strives to perform the best possible match between the old and the new instance tree; this is done to ensure that it is possible to keep service disruption to a minimum when running the fulfillment processes to get from the previous situation to the new situation.

NOTE

If two or more instances belong to the same variable cardinality specification in the catalog, the algorithm gives no importance to the order of the instances. For instance, if the old instance tree contains instance A followed by instance B (both belonging to the same specification) and the new instance contains instance B followed by instance A, then the algorithm will see the two as being identical.

Delta Tree

A delta tree contains a product instance as its root as well as CFS, RFS, and EWI child instances. In contrast to an ordinary product instance tree which expresses the current structure of a product instance, a delta tree expresses how to get from the old product instance structure to the new product instance structure. It does so by including branches and nodes that represent the following:

- Nodes that exist in both the old and the new instance trees and that have *not* been modified; i.e. the characteristic values of the nodes in the old and new product instances are all identical. Such nodes will be tagged as `KEEP`.
- Nodes that exist in both the old and the new instance tree and that *have* been modified; i.e. at least one characteristic value of the nodes in the old and new product instances is different. Such nodes will be tagged as `MODIFY`.
- Nodes that exist in the old instance tree, but not in the new instance tree. Such nodes will be tagged as `DELETE`.
- Nodes that exist in the new instance tree, but not in the old instance tree. Such nodes will be tagged as `ADD`.

Example **Delta Tree Calculation**

This example shows how a delta tree could look like based on an old and a new product instance tree. Consider the following old product instance tree:

```
P_Data.4 --|
           |-- C_DSL.3 --|
           |             |-- R_DSL.1
           |             |
           |             |-- C_FixedIP.2 (WebServer) --|
           |                                           |-- R_FixedIP.6
           |
           |-- C_CPE.2 --|
                         |-- R_DSL.1
```

Now, assume that Service Provisioner receives a service request that wants to modify the product instance tree structure. The new product instance tree looks as follows:

```
P_Data.4 --|
           |-- C_DSL.3 --|
                         |-- R_DSL.1
                         |
                         |-- C_FixedIP.2 (WebServer) --|
                         |                             |-- R_FixedIP.6
                         |
                         |-- C_FixedIP.2 (Email+FTP) --|
                                                       |-- R_FixedIP.6
```

If these two trees are passed through HP Service Provisioner delta engine, the resulting delta tree will look as shown below:

```
P_Data.4 --|
           |-- C_DSL.3 --|
           |             |-- R_DSL.1
           |             |
           |             |-- C_FixedIP.2 (WebServer) --|
           |             |                             |-- R_FixedIP.6
           |             |
           |             |-- C_FixedIP.2 (Email+FTP) --|
           |             |                             |-- R_FixedIP.6
           |
           |-- C_CPE.2 --|
                         |-- R_DSL.1
```

Nodes that carry the `ADD` tag are marked in green and nodes that carry the `DELETE` tag are marked in red. The remaining nodes are either carry the `KEEP` or the `MODIFY` tag; this example makes no distinction between `KEEP` and `MODIFY` tags because characteristic values are not considered here.

Fulfillment Processes for Delta Trees

Once a delta tree has been generated the fulfillment processes for actually transforming the old product instance tree into the new product instance tree is as follows (in the listed sequence).

1. The old and the new product instance trees are deleted; they are not used anymore. Only the delta tree is used onwards.
2. All nodes that carry the `DELETE` tag are moved through the fulfillment states from their current state to the retired state (e.g. *Active → Provisioned → Reserved → Designed → Terminated → Retired*); if action workflows are defined for the nodes for these state transitions, they will be executed. Once this process is finished, the nodes are deleted from the delta tree.
3. All nodes that carry the `ADD` tag are moved through the fulfillment states from the initial state to the current state of the remaining nodes (e.g. *Initial → Checked → Designed → Reserved → Provisioned → Active*); if action workflows are defined for the nodes for these state transitions, they will be executed.
4. A *modify* state transition will be carried out for the entire instance tree (e.g. *Active → Active*); if action workflows are defined for this state transitions, they will be executed.

The result of all these steps will be that the delta tree is no longer really a delta tree; now it's identical to the new product instance tree. In the final step, the product instance tree will be written to Subscription Repository and an asynchronous message will be emitted.

Delta Operation Example

This section gives a complete example of delta operations. It starts by creating a new product instance (based on a service request containing a partial instance tree) which is then structurally modified. Finally, the example demonstrates a simple modify operation where just a single characteristic value is changed.

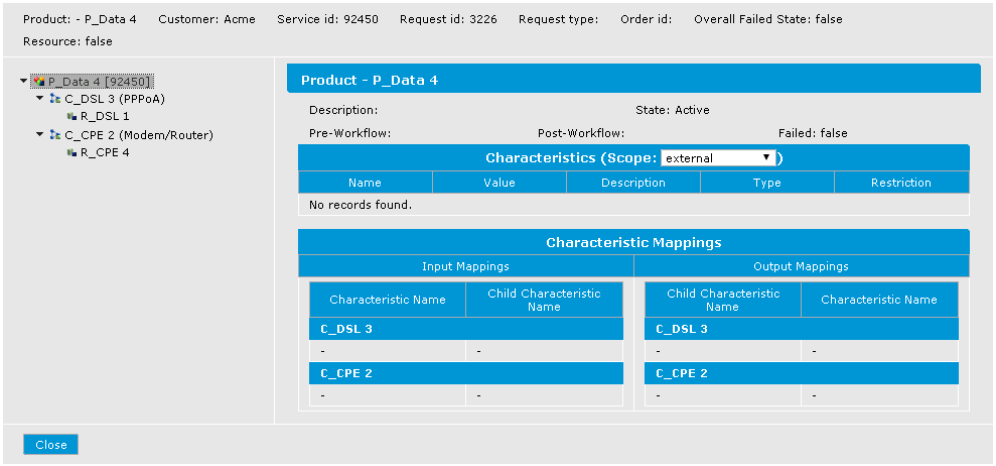
Consider the product specification shown in Figure 33. This product specification mimics a "Data Connectivity" product where it is possible to order a DSL connection and a CPE (for instance, a DSL modem/router). In addition, a number of fixed IP addresses can be assigned to the DSL connection; for instance, if the customer needs to host a web server or an email server.

Assume, that a customer opts for a simple product of this type; i.e. the product shall contain DSL connectivity and a CPE. The following WS/SOAP service request is sent to HP Service Provisioner to create this service:

```
<Request>
  <RequestType>activate</RequestType>
  <RequestId>3226</RequestId>
  <ProductName>P_Data</ProductName>
  <ProductVersion>4</ProductVersion>
  <Customer>Acme</Customer>
  <CFS>
    <Name>C_DSL</Name>
    <Version>3</Version>
  </CFS>
  <CFS>
    <Name>C_CPE</Name>
    <Version>2</Version>
  </CFS>
</Request>
```

The screenshot in Figure 34 shows the product instance that results from this WS/SOAP service request; the figure also shows that the two expected RFSs (“R_DSL” and “R_CPE”) have been added to their correct positions by the “validate and expand” algorithm (because they were both present in the product specification tree in the catalog).

Figure 34 **Product Instance Created from Partial Product Instance Tree**



Now, assume that the customer wants to replace the CPE with one of his own; hence, he does not want a CPE to be part of the subscription anymore. At the same time he wishes to get two fixed IP addresses (10.20.30.40 and 10.20.30.41) assigned to the DSL service to run a web server and an email server.

The WS/SOAP request to fulfill the customer's needs could look like the following example:

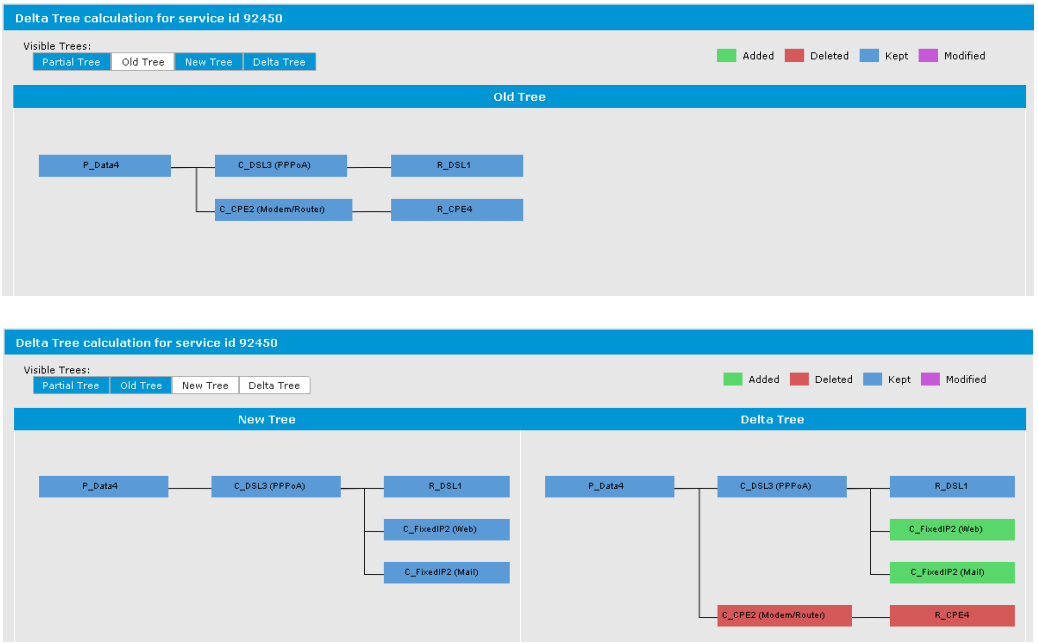
```
<Request>
  <RequestType>modify</RequestType>
  <RequestId>3251</RequestId>
  <ProductName>P_Data</ProductName>
  <ProductVersion>4</ProductVersion>
  <Customer>Acme</Customer>
  <ServiceId>92450</ServiceId>
  <CFS>
    <Name>C_DSL</Name>
    <Version>3</Version>
    <CFS>
      <Name>C_FixedIP</Name>
      <Version>2</Version>
      <Label>Web</Label>
      <Characteristic name="ip">10.20.30.40</Characteristic>
    </CFS>
    <CFS>
      <Name>C_FixedIP</Name>
      <Version>2</Version>
      <Label>Mail</Label>
      <Characteristic name="ip">10.20.30.41</Characteristic>
    </CFS>
  </CFS>
</Request>
```

While the fulfillment processes are ongoing (i.e. while Service Provisioner is transforming the customer's product into the new product) it is possible to get an overview of the delta operations in HP Service Provisioner's UI. Under the "Processing" tab in the "Instance Management" UI there is a row for each active fulfillment process. By clicking the middle button (of the tree buttons in the right side of each table row) it is possible to bring up a graphical representation of the partial product instance tree, the old product instance tree, the new product instance tree, and the delta tree. Figure 35 shows this UI displaying a graphical representation of the old instance tree, the new instance tree, as well as the delta tree.

NOTE

The two "R_FixedIP" RFSs are not visible in the "new tree" and the "delta tree" areas because the right part of the trees have been truncated. They do exist in both trees, though.

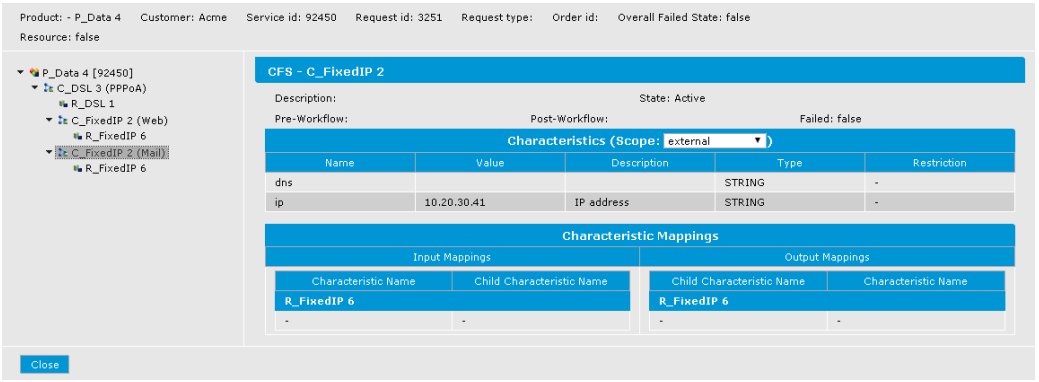
Figure 35 **UI for Displaying Old and New Instance Trees and Delta Tree**



The delta tree shown in the UI uses red and green colors to highlight nodes that carry the `DELETE` and `ADD` tag, respectively. From this, it is clear that the “C_CPE” is to be deleted whereas two “C_FixedIP” CFSs are to be added (as well as their two child RFSs).

The product instance tree after the delta operations are finished are shown in Figure 36. Note that the value of the `ip` characteristic in the “C_FixedIP” CFS labeled “Mail” is “10.20.30.41” as specified in the WS/SOAP service request.

Figure 36 **Product Instance after Delta Operation (CFSs Added and Deleted)**

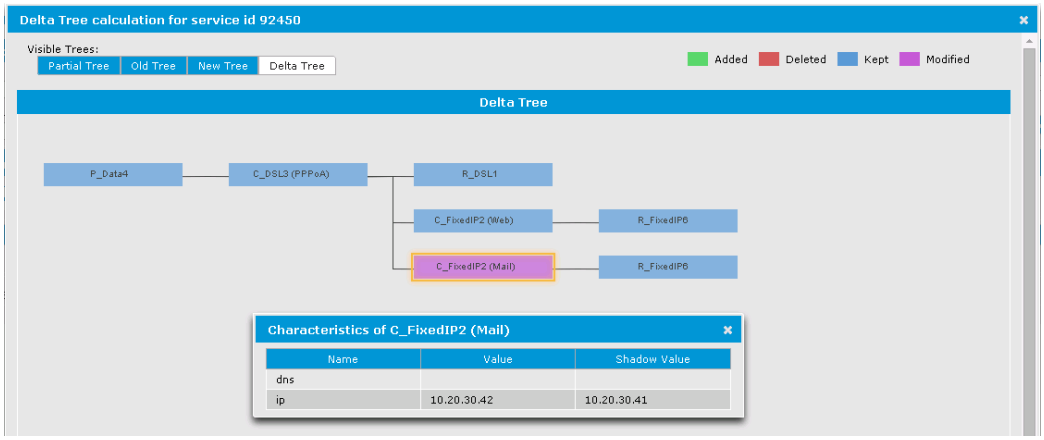


Finally, the customer wishes to change the IP address of his email server (the “C_FixedIP” CFS labeled “Mail”). The following WS/SOAP request shows how this could be done (note that the value of the `ip` characteristic has changed to “10.20.30.42”):

```
<Request>
  <RequestType>modify</RequestType>
  <RequestId>3251</RequestId>
  <ProductName>P_Data</ProductName>
  <ProductVersion>4</ProductVersion>
  <Customer>Acme</Customer>
  <ServiceId>92450</ServiceId>
  <CFS>
    <Name>C_DSL</Name>
    <Version>3</Version>
    <CFS>
      <Name>C_FixedIP</Name>
      <Version>2</Version>
      <Label>Web</Label>
      <Characteristic name="ip">10.20.30.40</Characteristic>
    </CFS>
    <CFS>
      <Name>C_FixedIP</Name>
      <Version>2</Version>
      <Label>Mail</Label>
      <Characteristic name="ip">10.20.30.42</Characteristic>
    </CFS>
  </CFS>
</Request>
```

Figure 37 shows the “delta tree” UI displaying how the delta tree looks like for fulfilling this modify request. The delta tree contains no red or green nodes because no nodes are deleted or added. One node has a purple color to indicate that one or more characteristics have changed values for this node. By clicking on the node, it is possible to bring up a window that displays the characteristic values (current as well as shadow values). In this example, it is clear that the value of the `ip` characteristic has been changed from “10.20.30.41” to “10.20.30.42”.

Figure 37 **Delta Tree UI Displaying Modified Characteristic Values**



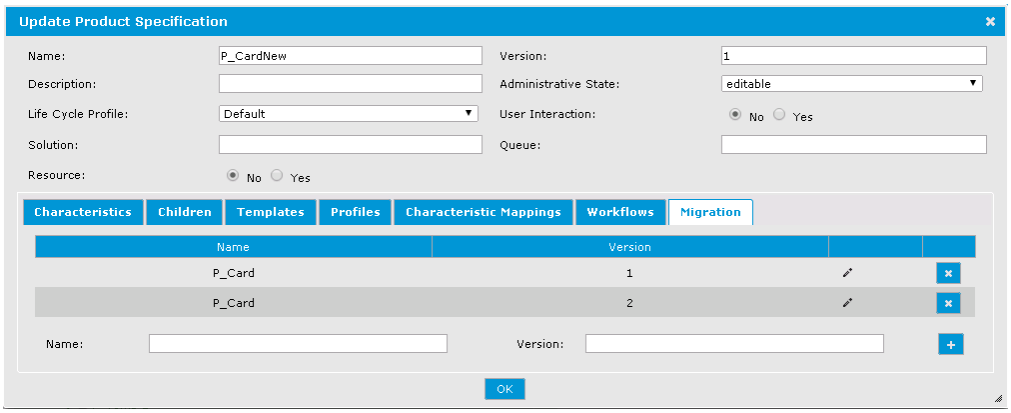
Migration between Products

All delta operations described so far in this chapter have all assumed that the old product instance tree and the new product instance tree were both instances of the same product specification. However, it is also possible to migrate a product instance from one product specification type to another. This is done exactly in the same way as any other delta operation; a service request containing a partial instance tree is sent to Service Provisioner who will then calculate a delta tree that can change the existing product instance to the new product instance belonging to a different product specification.

In principle, it would be possible to migrate from any type of product to any other type; but in most cases the two specifications are expected to be similar. Otherwise, if the two product types have no commonalities whatsoever, then the delta tree will basically delete all existing CFSs and create new ones to match the new specification.

One condition must be met in order to be able to migrate from one product type to another. The product specification for the new product must have the product specification of the old product in its “migration list”. The “migration list” is a list that contains zero or more product names and versions that identify products that can be migrated into a product of this specification. Figure 38 shows the “Update Product Specification” UI with its “Migration” tab selected. In this case, it is possible to migrate products of the “P_Card” specification (version 1 and 2) to this product type. If the “migration list” is empty it will not be possible to migrate any product instances into product instances of this type.

Figure 38 **Update Product Specification UI Displaying the Migration Tab**



10 Dependencies

This chapter describes dependencies which has been introduced as of version 7.0 or HP Service Provisioner. Dependencies can only exist between service instances, not between service specifications.

By using dependencies it is possible to model resources that are consumable by other instance. As an example, a product instance may model a network link with a capacity of 100 Mbps. This resource can then be consumed by two other instances, each consuming 50 Mbps; or by ten other instances, each consuming 10 Mbps.

Terminology

In HP Service Provisioner all dependencies have a single *source* and a single *destination*; if instance *A* depends on instance *B*, then instance *A* is the *source* and *B* is the *destination* of the dependency. The following restrictions apply for dependencies:

- Dependencies can only exist between instances; not between specifications
- Only product instances can be *destinations* of dependencies; such product instances are also referred to as *resource* product instances; a resource product instance can be the destination of multiple dependencies
- All instance types (product instances, CFSs, RFSs, as well as EWIs) can be the *source* of dependencies; an instance can be the source of multiple dependencies

NOTE — A product instance that is the destination of one or more dependencies is also said to have *ingoing* dependencies. Likewise, an instance that is the source of one or more dependencies is also said to have *outgoing* dependencies.

In order for a product instance to act as a resource, two conditions must be met. The first condition is that its specification must have the “Resource” flag set to “true” (see the Section “Resource” on page 53). The second condition is that the product instance must set its *capacity*; this is done from an action workflow belonging to the product instance. As alluded to by its name, the *capacity* is an integer value that is used to model the capacity of a resource product instance.

A dependency in HP Service Provisioner will always use a certain amount of the capacity offered by a resource product instance; this amount is referred to as the dependency’s *used capacity* (shorthand form: *used*). The sum of the *used capacities* of all dependencies that have the same resource their targets can never exceed the capacity of the resource.

The source instance must always be the creator of a dependency. Two parameters are needed for a source instance in order to create a dependency:

- **Service id of the resource instance:** A source instance uniquely identifies a resource instance through the service id of the resource instance.
- **Used capacity:** The used capacity (a.k.a. “consumed capacity”) is needed so that the resource instance can check whether the dependency can be created (i.e. whether there is enough free capacity) and to adjust the total amount of capacity that is in use for the resource instance.

Dependency Example

Before describing how to manage resource instances and dependencies, the following example may assist in better understanding dependencies.

Consider a VPN instance (with service id = 1) and a Site instance (with service id = 2). The VPN as well as a Site instance are resources; their capacities are 20 and 2, respectively:

```
VPN  (SID:1) – Capacity:20 - Used:0
Site (SID:2) – Capacity:2  - Used:0
```

Also, consider a SiteAttachment instance (with service id = 3) that is used to model a connection between the VPN and the Site instances. The SiteAttachment is modeled as a product instance with two CFSs (for simplicity, RFSs are left out of this example):

```
SiteAttachment (SID:3) --|
                         |-- VPN_Ref (CFS)
                         |
                         |-- Site_Ref (CFS)
```

The SiteAttachment product instance contains the VPN_Ref CFS and the Site_Ref CFS that are intended to act as sources for dependencies to the VPN and Site instances, respectively.

When the fulfillment processes are executed the action workflows for VPN_Ref and Site_Ref will be executed after one another. When the two action workflows have created the two dependencies, the situation will look as follows for the VPN and Site instances:

```
VPN (SID:1)  – Capacity:20 – Used:1
               Used by [SID:3 – Used_capacity:1]
Site (SID:2) – Capacity:2  – Used:1
               Used by [SID:3 – Used_capacity:1]
```

This means that the VPN instance knows that it is currently being used by an instance with service id = 3 and the used capacity is 1. The situation is similar for the Site instance.

For the SiteAttachment instance, the situation will look as follows:

```
SiteAttachment (SID:3) --|
                         |-- VPN_Ref  [Depends on SID:1 – used: 1]
                         |
                         |-- Site_Ref [Depends on SID:2 – used: 1]
```

Managing Resource Product Instances

HP Service Provisioner provides three workflow nodes that can be used from within the action workflows of product instance resources to do the following:

- **`SOMResourceUpdateCapacity:`** Set and modify the capacity of the resource.
- **`SOMResourceGetCapacityUsage:`** Get the capacity of the resource as well as its used capacity.
- **`SOMResourceListIngoingDependencies:`** Get a list of service ids of all instances that have ingoing dependencies to this resource (along with the used capacities of each dependency).

The three workflow nodes are described in details in Chapter 12.

Managing Dependencies

Five workflow nodes are provided by HP Service Provisioner to create, list, modify, replace, and remove dependencies. The workflow nodes are to be used in action workflows of instances that have (or are going to have) outgoing dependencies. The five workflow nodes are:

- **`SOMResourceCreateDependency:`** Create a dependency from the current instance (product instance, CFS, RFS, or EWI) to a resource product instance.
- **`SOMResourceListOutgoingDependencies:`** Get a list of all service ids of product instance resources that this instance has outgoing dependencies to. The workflow node also returns a list of values representing how much capacity is used for each outgoing dependency.
- **`SOMResourceModifyDependency:`** Modify the used capacity of an existing dependency from this instance (i.e. change the currently used capacity to a new value that may be higher or lower).
- **`SOMResourceRemoveDependency:`** Remove an existing dependency emanating from this instance.
- **`SOMResourceReplaceDependency:`** Remove an existing dependency from this instance and create new dependency to a new product instance resource; i.e. this node effectively replaces one dependency with another dependency. The functionality of this node is equivalent to executing `SOMResourceRemoveDependency` followed by executing `SOMResourceCreateDependency`.

The five workflow nodes for managing (outgoing) dependencies are described in more detail in Chapter 12.

11 Scheduled Requests

HP Service Provisioner has support for scheduled requests which means that it is possible to define one or more schedule events to be triggered at a specified time in the future. Schedule events can be either one-off events or repeating events. The scheduling functionality is based to HP Service Activator's `SchedulerModule` which means that while a process is waiting in the scheduler it has zero memory footprint.

The scheduling functionality in HP Service Provisioner is based on a simple principle: A scheduled event is functionally equivalent to the reception of a scheduled service request. Hence, anything that is possible as a result of the arrival of a service request is also possible using the HP Service Provisioner's scheduling functionality.

Schedule API

The API for setting up scheduled requests in HP Service Provisioner is an extension to the already existing northbound API (see the Section "Northbound API" on page 39). To set up one or more schedules, one or more "schedule" element need to be added to the service request as the last element (or elements).

At a high level a service request containing two schedules may look as described in the following simple example:

- **Service Request:** "Create DSL service and put it into *Reserved* state" (i.e. *reserve* operation)
- **Schedule 1:** "On May 1st, put the service into *Active* state" (*activate* operation)
- **Schedule 2:** "On August 31st, tear down the service" (*retire* operation)

NOTE The order in which schedules are defined has no significance.

A schedule element in HP Service Provisioner has the following contents:

Start time Mandatory. This is the time to trigger this schedule event (or the first time in case this is a repeating schedule). There are two ways to specify the start time:

- **Integer value:** When using this syntax, convert the time to the number of seconds since January 1, 1970 UTC.
- **Time string:** When using this syntax, use the time format `yyyy-MM-dd HH:mm:ss` (UTC). For an explanation of the time format string, please read the Javadocs for the class `java.text.SimpleDateFormat`.

Ignore Missed Events	Optional, default value is “false”. This expects a boolean value (“true” or “false”). If the value is “true”, then HP Service Provisioner will not make any attempts to trigger a schedule event that has been missed (e.g. because the start time has already expired at the time when the service request was received). Otherwise, if the value is “false”, a missed scheduling event will result in Service Provisioner attempting to trigger the schedule event at the earliest time possible.
Repeat	Optional. A schedule may contain a single repeat section. If the repeat section is present, it must contain the following: • *Frequency*: Mandatory. This is the interval between repeating schedule events. If set to an integer it specifies the number of seconds between repetitions. If the integer value carries any of the suffixes `s`, `m`, `h`, `d`, or `w`, the value must be specified in seconds, minutes, hours, days, or weeks, respectively. • *End time*: Optional. After this time there will be no more events based on this schedule. The supported syntax is identical to that of *start time*. If no end time is specified, the repeating schedule will continue infinitely.
Request	The schedule must contain a service request. See the Section “Northbound API” on page 39 for a description of service requests. Since a service request *may* contain schedules and a schedule must contain a service request, a service request can, in fact, contain several levels of schedules. HP Service Provisioner will always set the service ids of requests embedded inside schedules to the value of the service id of the product instance that this request has been operating on. Therefore, if any of the embedded requests contain service ids they will be silently overwritten.

NOTE For a full specification of the syntax for WS/SOAP service requests with schedules, please study the WSDL document; the Section “Northbound API” on page 39 explains how the WSDL can be generated.

Scheduling Events

If HP Service Provisioner receives a service request that contains one or more schedules, it will *first* finish the operation dictated by the “outer” request (i.e. the part of the request that is *not* embedded inside any schedules). *Then*, when the operation is finished, Service Provisioner will pass on the request to Service Provisioner’s scheduling engine that does the following (in the listed order):

- For all schedules that are defined at the top level (i.e. not schedules within schedules) the scheduling engine will calculate a list of timestamps (date and time) of their next schedule event.
- If there are more than one element in the list, the list will be sorted in ascending order and the first element will be elected as the next schedule event to be triggered. If the elected schedule is a one-off schedule it will be stripped away from the request; otherwise, if the elected schedule is a repeating schedule (and the end time has not been passed) it will remain in the list of schedules.
- The “outer” request will be replaced with the request embedded in the newly elected schedule, and the workflow job that processes this service will place itself in HP Service Activator scheduler and wait until it is woken up.
- When the workflow job is woken up (i.e. at the time of the schedule event), it will run exactly the same steps as if a new service request had just been received via the northbound interface.

Example **WS/SOAP Service Request with a Simple Schedule**

The following example shows a WS/SOAP request containing a simple schedule (some mandatory parts of the request that are not important for this example have been left out):

```
<Request>
  <RequestType>reserve</RequestType>
  <RequestId>r-12241</RequestId>
  <ProductName>DSL</ProductName>
  <ProductVersion>7</ProductVersion>
  ... list of characteristic values, etc. ...
  <Schedule>
    <StartTime>2015-02-20 05:30:00</StartTime>
    <Request>
      <RequestType>active</RequestType>
    </Request>
  </Schedule>
</Request>
```

This effect of this request is that a new service named DSL (version 3) will be created immediately and the fulfillment processes will stop in the *Reserved* state. Then, on February 20, 2015 (at 5:30 in the morning) an *activate* request will be injected which means that the service will end up in the *Active* state and therefore be available for consumption by the customer. Since there if only one schedule in this request (and it is not a repeating schedule), the workflow job will not be passed to the scheduling engine after the *Active* state is reached.

Schedule Modification

HP Service Provisioner supports three operations that can be used to modify the schedule (or schedules) of product instances that currently have schedules attached to them. The three schedule modification operations are:

- **Delete schedule:** A delete schedule operation will simply delete *all* schedules that are currently attached to a product instance (identified by its service id) and then leave the product instance in the state it is currently in.

- **Replace schedule:** A replace schedule operation will overwrite all schedules that are currently attached to a product instance (identified by its service id) with the schedule or schedules that exist in the new service request.

- **Append to schedule:** This operation can be used to append one or more schedules to an existing list of schedules attached to a product instance. This operation can, for instance, be used to set a future *retire* date/time for an existing instance.

The API for modifying a schedule is identical to the API described in the Section “Schedule API” on page 89 with the exception that an attribute named `modify` needs to be specified and set to one of the following values:

- **DELETE:** To delete the schedule.

- **REPLACE:** To replace the existing schedule.

- **APPEND:** To append a new schedule to the list of existing schedules.

Example **WS/SOAP Service Request to Append a Schedule**

The following example shows a WS/SOAP request that appends a schedule (requesting a *retire* operation on August 12, 2015 at 8:00 in the morning) to an existing product with service id 17913:

```
<Request>
  <RequestType>noop</RequestType>
  <RequestId>r-12241</RequestId>
  <ServiceId>17913</ServiceId>
  <Schedule modify="APPEND">
    <StartTime>2015-08-12 08:00:00</StartTime>
    <Request>
      <RequestType>retire</RequestType>
    </Request>
  </Schedule>
</Request>
```

Notice that the request type is set to `noop`. This it done because this request is only meant to append a schedule to the product instance; *not* to move it to another state.

Repeating Schedule Example

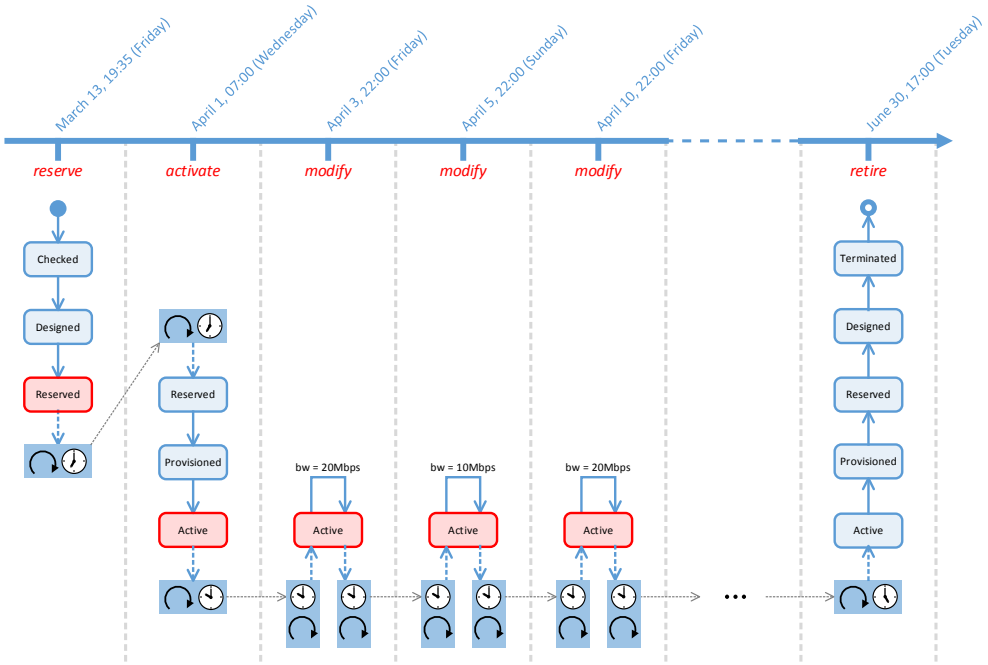
This section contains an example of a service request containing a list of schedules. The service request creates a DSL service and brings it into active state for 3 months; during this time, the bandwidth of the DSL service is modified every Friday and every Sunday (to provide higher bandwidth in the weekend). This is an example of how such a WS/SOAP service request could look like (some mandatory parts of the request that are not important for this example have been left out):

```
<Request>
  <RequestType>reserve</RequestType>
  <RequestId>r-12241</RequestId>
  <ProductName>DSL</ProductName>
  <ProductVersion>3</ProductVersion>
  <Characteristic name="bw">20Mbps</Characteristic>
  <Schedule>
    <StartTime>2015-04-01 07:00:00</StartTime>
    <Request>
      <RequestType>activate</RequestType>
    </Request>
  </Schedule>
  <Schedule>
    <StartTime>2015-06-30 17:00:00</StartTime>
    <Request>
      <RequestType>retire</RequestType>
    </Request>
  </Schedule>
  <Schedule>
    <StartTime>2015-04-03 22:00:00</StartTime>
    <Request>
      <RequestType>modify</RequestType>
      <Characteristic name="bw">20Mbps</Characteristic>
      <Repeat>
        <Frequency>1w</Frequency> <!-- Repeat every week -->
      </Repeat>
    </Request>
  </Schedule>
  <Schedule>
    <StartTime>2015-04-05 22:00:00</StartTime>
    <Request>
      <RequestType>modify</RequestType>
      <Characteristic name="bw">10Mbps</Characteristic>
```

```
      <Repeat>
        <Frequency>1w</Frequency> <!-- Repeat every week -->
      </Repeat>
    </Request>
  </Schedule>
</Request>
```

Figure 39 depicts how the service instance is created and how it traverses the service fulfillment states based on scheduled requests. Between April 1st and June 30th repeated *modify* operations toggle the value of the bandwidth characteristic `bw` between the two values `20Mbps` and `10Mbps`.

Figure 39 **Service with Repeating Schedule and Predefined Start and End Dates**



12 Workflow Manager Module and Node Library

This chapter lists the workflow manager modules and workflow nodes that are included in HP Service Provisioner and describes their parameters. The content of this chapter is primarily of interest to system integrators.

NOTE HP Service Provisioner includes additional workflow nodes than those listed in this chapter. The additional nodes are meant for internal use only, and their behavior may be changed in the future without notice.

Workflow Manager Modules

SRModule

`com.hp.ov.activator.mwfm.engine.module.som.SRModule`

This module communicates with Subscription Repository via WS/SOAP. The configuration of this module is mandatory in order for HP Service Provisioner to function. To use the module to interact with Subscription Repository, you need to use the workflow nodes described in the Section "Workflow Nodes for Product Instance Access" on page 107. To use Subscription Repository this module must be configured in the file *$ACTIVATOR_ETC*`/config/mwfm.xml`. The name of the module must be `ServiceOrderManagement`.

In order to optimize performance, the module maintains an in-memory cache of all product catalog objects from Subscription Repository. The time between refreshing the cache is configurable.

Table 5 SRModule Parameters

Name	Required	Description	Default
ws_url	Yes	The URL to use to communicate with Subscription Repository.	None
username	No	The username to use to connect to Subscription Repository. If Subscription Repository does not require authentication, you may skip this parameter (not recommended).	None
password	No	The password to use to connect to Subscription Repository. The password may either be plain text or encrypted format (if the *encrypted_password* parameter is set to "true").	None

Name	Required	Description	Default
encrypted_password	No	Set this parameter to “true” if you wish to specify the password in encrypted format.	`false`
cache_retry_interval	No	The time in milliseconds to retry refreshing the cache in case of communication errors.	`10000`
cache_refresh_interval	No	The time in minutes between refreshing the cache during normal operation. (0 means, no automatic refresh.)	`60`
debug	No	If this parameter is set to “true” the module will log all communication in the JBoss “server.log”. This parameter should not be set to “true” in production environments.	`false`
retry_count	No	The number of times to retry connecting to Subscription Repository in case the connection is lost (0 means “no retry”).	`3`
retry_interval	No	The interval between connection retry attempts (in milliseconds) when a workflow node tries to communicate with Subscription Repository through this module.	`10000`
min_threads	No	The minimum number of Java threads that will be created to process requests.	`1`
max_threads	No	The maximum number of Java threads that will be created to process requests. This is the number of simultaneous requests that can be processed. Additional requests will be queued until one of the Java threads becomes available.	`3`
store_audit	No	If this value is set to “false” HP Service Provisioner will not store any audit messages. Please read the Section “Audit Module Configuration” on page 36 for a description of how to enable auditing.	`true`
sleep_time	No	The HP Service Provisioner orchestration workflows contain a number of Sleep workflow nodes that will slow down the fulfillment processes ***if*** this parameter is set to an integer value larger than zero. The value of this parameter is in milliseconds. This parameter exists mainly for debugging and demonstration purposes. **IMPORTANT:** This parameter should never be set in production environments.	`0`
debug_workflow	No	If this value is set to “true”, HP Service Provisioner will write a lot of tracing information in the `server.log`. **IMPORTANT:** This parameter should never be set in production environments.	`false`

Example **Sample SRModule Configuration**

```
<Module>
  <Name>ServiceOrderManagement</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.som.SRModule
  </Class-Name>
  <Param name="ws_url"
     value="http://10.10.7.8:8080/subscriptionrepository/operations"/>
  <Param name="username" value="hpsp"/>
  <Param name="password" value="verySecret"/>
  <Param name="encrypted_password" value="false"/>
</Module>
```

TrueviewModule

`com.hp.ov.activator.mwfm.engine.module.TrueviewModule`

The `Trueview` module communicates with the Trueview inventory system via WS/SOAP. To use the module to interact with Trueview, you need to use the `TVWSRequest` workflow node described on page 113.

The module must be configured in the file *`$ACTIVATOR_ETC`*`/config/mwfm.xml`. There are no requirements for the name of the module.

Table 6 **TrueviewModule Parameters**

Name	Required	Description	Default
`ws_url`	Yes	The URL to use to communicate with Trueview.	None
`username`	Yes	The username to use to connect to Trueview.	None
`password`	Yes	The password to use to connect to Trueview.	None
`encrypted_password`	No	Set this parameter to “true” if you wish to specify the password in encrypted format.	`false`
`retry_count`	No	The number of times to retry connecting to Trueview in case the connection is lost (0 means “no retry”).	`3`
`retry_interval`	No	The interval between connection retry attempts (in milliseconds) when a workflow node tries to communicate with Trueview through this module.	`10000`
`min_threads`	No	The minimum number of Java threads that will be created to process requests.	`1`
`max_threads`	No	The maximum number of Java threads that will be created to process requests. This is the number of simultaneous requests that can be processed. Additional requests will be queued until one of the Java threads becomes available.	`3`

Example **Sample TrueviewModule Configuration**

```
<Module>
  <Name>trueview</Name>
  <Class-Name>
    com.hp.ov.activator.mwfm.engine.module.TrueviewModule
  </Class-Name>
  <Param name="username" value="hpsp"/>
  <Param name="password" value="verySecret"/>
  <Param name="encrypted_password" value="false"/>
  <Param name="ws_url"
         value="http://10.10.7.9:8012/tnp-ws/services"/>
  <Param name="retry_count" value="10"/>
  <Param name="retry_interval" value="3000"/>
  <Param name="max_threads" value="20"/>
</Module>
```

Workflow Nodes for Accessing Characteristics

The workflow nodes listed in this section can be used to access the characteristics of an instance from the instance's action workflows.

SOMAssignResult

`com.hp.ov.activator.mwfm.component.builtin.som.SOMAssignResult`

This workflow process node writes any number of case-packet variable values from the action workflow of an instance that currently being processed to the instance's characteristics. The node must be used from an action workflow because it relies on specific values of system case-packet variables set by the "SOMAction" and "SOMController" workflows.

By default the node writes the case-packet variable values to characteristics with the same name as the case-packet variables; it is, however, possible to assign values to characteristics with names that are different from the case-packet variable names.

Table 7 **SOMAssignResults Parameters**

Name	Required	Description	Default	Type
variable0, variable1, ... variableN	Yes	Case-packet variables which are to be assigned to characteristics (of identical names, unless *paramN* is specified).	None	Any
param0, param1, ... paramN	No	Names of the characteristics. If not specified it is assumed that the characteristic names are identical to the case-packet variable names.	None	String

Example **SOMAssignResult – use in workflow**

This workflow node assigns values to the charactistics named `bandwidth`, `port`, and `card`. The values are taken from the case-packet variables with the names `bw`, `pt`, and `card`, respectively.

```
<Process-Node>
  <Name>SOMAssignResult</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.SOMAssignResult
    </Class-Name>
    <Param name="param0" value="constant:bandwidth"/>
    <Param name="param1" value="constant:port"/>
    <Param name="variable0" value="bw"/>
    <Param name="variable1" value="pt"/>
    <Param name="variable2" value="card"/>
  </Action>
</Process-Node>
```

SOMCharacteristicsModified

`com.hp.ov.activator.mwfm.component.builtin.som.SOMCharacteristicsModified`

This workflow rule can be used to test whether any characteristic of an instance that currently being processed have been modified. It does so by comparing current characteristic values to their corresponding shadow value. If there are any changes, the workflow processing will continue along the “true” branch emanating from this rule node; otherwise the workflow processing will continue along the “false” branch emanating from this node.

If no parameters are specified, the node will check *all* characteristics for any changes; otherwise, if any *`paramN`* parameters are specified, the node will only check those listed here for changes. The node may, optionally, return a list of those characteristic names that have been modified.

Table 8 **SOMCharacteristicsModified Parameters**

Name	Required	Description	Default	Type
`param0, param1, ... paramN`	No	Names of the characteristics to check for changes. If not specified the node will check all characteristics that belong to this instance.	None	String
`modified_ characteristics`	No	If specified, the node will return a list of characteristic names (strings) that have been modified.	None	Object

Example **SOMCharacteristicsModified – use in workflow**

This workflow node checks whether any of the two characteristics `phone_num` and `ringback_id` have been modified. If none of the two characteristics have been modified the case-packet variable `modifiedList` will contain an empty list; otherwise it will contain a list containing those of the two characteristics that have been modified.

```
<Rule-Node>
  <Name>SOMCharacteristicsModified?</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
                                          SOMCharacteristicsModified
    </Class-Name>
    <Param name="modified_characteristics" value="modifiedList"/>
    <Param name="param0" value="phone_num"/>
    <Param name="param1" value="ringback_id"/>
  </Action>
</Rule-Node>
```

SOMGetCharacteristics

`com.hp.ov.activator.mwfm.component.builtin.som.SOMGetCharacteristics`

This workflow process node retrieves any number of characteristic values and writes them to case-packet variables in an action workflow of an instance that currently being processed. The node can retrieve current characteristic values as well as previous (also known as “shadow” characteristic values. By default, the workflow retrieves values from characteristics with names matching the names of the case-packet variables; it is, however, possible to get values from characteristics of different names.

Table 9 SOMGetCharacteristics Parameters

Name	Required	Description	Default	Type
variable0, variable1, ... variable	Yes	Case-packet variables which are to be assigned to characteristics (of identical names, unless *paramN* is specified).	None	Any
param0, param1, ... paramN	No	Names of the characteristics. If not specified it is assumed that the characteristic names are identical to the case-packet variable names.	None	String
shadow0, shadow1, ... shadowN	No	Set this to `true` to get the characteristic’s shadow value instead of its current value.	`false`	Boolean

Example SOMGetCharacteristics – use in workflow

This workflow node assigns the two case-packet variables `ringback_id` and `phone_num` with the values of the characteristics by the same names. In addition, shadow values of the same two characteristics are assigned to the case-packet variables `shadow_ringback_id` and `shadow_phone_num`, respectively.

```
<Process-Node>
  <Name>SOMGetCharacteristics</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
                                          SOMGetCharacteristics
    </Class-Name>
    <Param name="param2" value="ringback_id"/>
    <Param name="param3" value="phone_num"/>
    <Param name="shadow2" value="true"/>
    <Param name="shadow3" value="true"/>
    <Param name="variable0" value="ringback_id"/>
    <Param name="variable1" value="phone_num"/>
    <Param name="variable2" value="shadow_ringback_id"/>
    <Param name="variable3" value="shadow_phone_num"/>
  </Action>
</Process-Node>
```

Workflow Nodes for Managing Dependencies

The workflow nodes listed in this section can be used to manage instances with ingoing as well as outgoing dependencies. Dependencies are described in Chapter 10.

SOMResourceCreateDependency

```
com.hp.ov.activator.mwfm.component.builtin.
  som.SOMResourceCreateDependency
```

This workflow process node creates a dependency from the currently processing instance to a target product instance resource. The product instance must have been marked as a resource in the catalog in order to successfully create a dependency.

Table 10 **SOMResourceCreateDependency Parameters**

Name	Required	Description	Default	Type
target_id	Yes	The service id of the target product instance resource to which to create the dependency.	None	String
capacity	No	The capacity to request from the target product instance resource.	`1`	Numeric

Example **SOMResourceCreateDependency – use in workflow**

This is an example of a workflow node that attempts to create a new dependency from the current instance to a resource product instance identified by the value of the case-packet variable `tid`. The used capacity of the dependency is given by the value of the case-packet variable `usedCap`.

```
<Process-Node>
  <Name>SOMResourceCreateDependency</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMResourceCreateDependency
    </Class-Name>
    <Param name="capacity" value="usedCap"/>
    <Param name="target_id" value="tid"/>
  </Action>
</Process-Node>
```

SOMResourceGetCapacityUsage

```
com.hp.ov.activator.mwfm.component.builtin.
  som.SOMResourceGetCapacityUsage
```

The workflow process nodes get information from the currently processing product instance resource about its current (total) capacity as well as its used capacity. The used capacity can never exceed the capacity.

Table 11 **SOMResourceGetCapacityUsage Parameters**

Name	Required	Description	Default	Type
capacity	Yes	The name of the case-packet variable in which to return the total capacity of this product instance resource.	None	Numeric
used	Yes	The name of the case-packet variable in which to return the used capacity of this product instance resource.	None	Numeric

Example **SOMResourceGetCapacityUsage – use in workflow**

In this example the workflow node reads the total capacity into the case-packet variable `capacity` and the used capacity is read into the case-packet variable `used`.

```
<Process-Node>
  <Name>SOMResourceGetCapacityUsage</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMResourceGetCapacityUsage
    </Class-Name>
    <Param name="capacity" value="capacity"/>
    <Param name="used" value="used"/>
  </Action>
</Process-Node>
```

SOMResourceListIngoingDependencies

```
com.hp.ov.activator.mwfm.component.builtin.
  som.SOMResourceListIngoingDependencies
```

This workflow process node retrieves a list of all service ids that have ingoing dependencies to the currently processing product instance resource as well as a list of values representing how much capacity is used by each source service id. The sizes of the two returned lists will always be identical.

Table 12 **SOMResourceListIngoingDependencies Parameters**

Name	Required	Description	Default	Type
capacities	Yes	The name of the case-packet variable in which to return the list of used capacities (integer values) of the dependencies that have this resource as their targets.	None	Object
dependencies	Yes	The name of the case-packet variable in which to return the list of service ids (string values) of the instances that depend on this resource.	None	Object

Example **SOMResourceListIngoingDependencies – use in workflow**

In this example the workflow node reads the list of service ids of the instances that have ingoing dependencies to this resource into the case-packet variable `dep_list`. In addition, the used capacities of all dependencies are returned to the case-packet variable `cap_list`.

```
<Process-Node>
  <Name>SOMResourceListIngoingDependencies</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMResourceListIngoingDependencies
    </Class-Name>
    <Param name="capacities" value="cap_list"/>
    <Param name="dependencies" value="dep_list"/>
  </Action>
</Process-Node>
```

SOMResourceListOutgoingDependencies

`com.hp.ov.activator.mwfm.component.builtin.som.SOMResourceListOutgoingDependencies`

This workflow process node retrieves a list of all service ids of the product instance resources that the dependencies emanating from this instance have as their targets. In addition, a list of values representing how much capacity is used by each of the outgoing dependencies is retrieved. The sizes of the two returned lists will always be identical.

Table 13 **SOMResourceListOutgoingDependencies Parameters**

Name	Required	Description	Default	Type
capacities	Yes	The name of the case-packet variable in which to return the list of used capacities (integer values) of all dependencies that have this instance as their sources.	None	Object

Name	Required	Description	Default	Type
dependencies	Yes	The name of the case-packet variable in which to return the list of service ids (string values) of the product instance resources that this instance depend on.	None	Object

Example **SOMResourceListOutgoingDependencies – use in workflow**

In this example the workflow node reads the list of service ids of the product instance resources that this instance depend on into the case-packet variable `dep_list`. In addition, the used capacities of all dependencies are returned to the case-packet variable `cap_list`.

```
<Process-Node>
  <Name>SOMResourceListOutgoingDependencies</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMResourceListOutgoingDependencies
    </Class-Name>
    <Param name="capacities" value="cap_list"/>
    <Param name="dependencies" value="dep_list"/>
  </Action>
</Process-Node>
```

SOMResourceModifyDependency

`com.hp.ov.activator.mwfm.component.builtin.som.SOMResourceModifyDependency`

This workflow process node modifies an existing dependency emanating from the currently instance by changing its used capacity to a new value. The new capacity must be a value larger than zero and not exceed the capacity of the target product instance resource.

Table 14 **SOMResourceModifyDependency Parameters**

Name	Required	Description	Default	Type
capacity	No	The new capacity to request from the target product instance resource.	1	Numeric
target_id	Yes	The service id of the target product instance resource to which to modify the used capacity of an existing dependency.	None	String

Example **SOMResourceModifyDependency – use in workflow**

In this example the workflow node attempts to modify the used capacity of an outgoing dependency that has the product instance resource with the service id given by the case-packet variable `tid` as it target. The new capacity to request for this dependency is given by the case-packet variable `usedCap`.

```
<Process-Node>
  <Name>SOMResourceModifyDependency</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMResourceModifyDependency
    </Class-Name>
    <Param name="capacity" value="usedCap"/>
    <Param name="target_id" value="tid"/>
  </Action>
</Process-Node>
```

SOMResourceRemoveDependency

`com.hp.ov.activator.mwfm.component.builtin.som.SOMResourceRemoveDependency`

This workflow process node removes a dependency from the current instance to a product instance resource with a given service id.

Table 15 SOMResourceRemoveDependency Parameters

Name	Required	Description	Default	Type
target_id	Yes	The service id of the target product instance resource to which to delete a dependency emanating from this instance.	None	String

Example SOMResourceRemoveDependency – use in workflow

In this example the workflow node attempts to remove a dependency from this instance to the product instance resource with the service id given by the case-packet variable `tid`.

```
<Process-Node>
  <Name>SOMResourceRemoveDependency</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMResourceRemoveDependency
    </Class-Name>
    <Param name="target_id" value="tid"/>
  </Action>
</Process-Node>
```

SOMResourceReplaceDependency

```
com.hp.ov.activator.mwfm.component.builtin.
  som.SOMResourceReplaceDependency
```

This workflow process node replaces a dependency from the current instance to a product instance resource with a given service id with a new dependency to another product instance with another service id. The used capacity of the new dependency must also be provided to this workflow node.

Table 16 **SOMResourceReplaceDependency Parameters**

Name	Required	Description	Default	Type
capacity	No	The capacity to request from the new target product instance resource.	`1`	Numeric
old_target_id	Yes	The service id of the old target product instance resource to which to delete a dependency emanating from this instance.	`None`	String
new_target_id	Yes	The service id of the new target product instance resource to which to create a new dependency emanating from this instance.	`None`	String

Example **SOMResourceReplaceDependency – use in workflow**

In this example the workflow node attempts to replace a dependency from this instance to the product instance resource with the service id given by the case-packet variable `old_tid` with a new dependency to the product instance resource with the service id given by the case-packet variable `new_tid`. The used capacity of the new dependency is given by the case-packet variable `usedCap`.

```
<Process-Node>
  <Name>SOMResourceReplaceDependency</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMResourceReplaceDependency
    </Class-Name>
    <Param name="capacity" value="usedCap"/>
    <Param name="new_target_id" value="new_tid"/>
    <Param name="old_target_id" value="old_tid"/>
  </Action>
</Process-Node>
```

SOMResourceUpdateCapacity

```
com.hp.ov.activator.mwfm.component.builtin.
  som.SOMResourceUpdateCapacity
```

This workflow process set or updates the capacity of a product instance resource. This workflow node can only be used in action workflows associated with product instances whose specifications have been marked as resources in the catalog.

Table 17 **SOMResourceUpdateCapacity Parameters**

Name	Required	Description	Default	Type
capacity	Yes	The capacity to of this product instance resource. This values cannot be lower than the currently used capacity.	None	Numeric

Example **SOMResourceUpdateCapacity – use in workflow**

In this example the workflow node sets the capacity of this product instance resource to a value specified by the case-packet variable `cap`.

```
<Process-Node>
  <Name>SOMResourceUpdateCatacity</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMResourceUpdateCapacity
    </Class-Name>
    <Param name="capacity" value="cap"/>
  </Action>
</Process-Node>
```

Workflow Nodes for Product Instance Access

The workflow nodes listed in this section can be used for direct access to product instances. In most cases, there should be no reason to use these nodes in action workflows.

SOMCreateProductInstance

`com.hp.ov.activator.mwfm.component.builtin.som.SOMCreateProductInstance`

This workflow node creates a product instance in Subscription Repository and/or in memory based on the name and version of the product specification.

Table 18 **SOMCreateProductInstance Parameters**

Name	Required	Description	Default	Type
product_name	Yes	The name of the product specification from which to create a product instance.	None	String
product_version	Yes	The version of the product specification from which to create a product instance.	None	String
customer	Yes	The value of the customer attribute to set for this product instance.	None	String
id	Yes	The service id of the product instance created by this workflow node.	None	String

Name	Required	Description	Default	Type
template_name	No	The name of a template from which to use characteristic values when creating this product instance.	None	String
template_version	No	The version of a template from which to use characteristic values when creating this product instance.	None	String
display_label	No	A display label to set for this product instance.	""	String
order_id	No	An order id to be assigned to this product instance.	None	String
request_id	No	A request id to be assigned to this product instance.	None	String
request_type	No	A request type to be assigned to this product instance.	None	String
state	No	A life-cycle state to be assigned to this product instance (including all its children).	""	String
characteristics	No	A Java map containing characteristics to set for this product instance. Keys in the map are characteristic names (strings) and values in the map are characteristic values.	None	Object
in_memory	No	If set to "true" the product instance will only be created in memory; not persisted to Subscription Repository. If set to "false" the product instance will be created in Subscription Repository *and* in memory.	false	Boolean
product_instance_var	Yes	The object case packet variable name where the created product Instance will be returned to.	None	Object

Example **SOMCreateProductInstance – use in workflow**

This example creates a product instance (name and version are "DSL" and "3", respectively) with service id = 22314 and sets its state to *Reserved*. Characteristics values are assigned from a template; characteristics values are explicitly assigned in this example.

```
<Process-Node>
  <Name>SOMCreateProductInstance</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.som.
        SOMCreateProductInstance
     </Class-Name>
     <Param name="product_name" value="DSL"/>
     <Param name="product_version" value="3"/>
     <Param name="customer" value="Acme"/>
     <Param name="id" value="22314"/>
     <Param name="template_name" value="DSL_GOLD"/>
     <Param name="template_version" value="1"/>
     <Param name="product_instance_var" value="prodInstance"/>
  </Action>
</Process-Node>
```

SOMDeleteProductInstance

`com.hp.ov.activator.mwfm.component.builtin.som.SOMDeleteProductInstance`

This workflow node deletes a product instance in Subscription Repository by its identifier.

Table 19 SOMDeleteProductInstance Parameters

Name	Required	Description	Default	Type
id	Yes	The service id of the Product instance to be deleted.	None	String
ignore_non_existing_instance	No	If set to “false” the node will fail if the product instance does not exist. Otherwise, if set to “true” the node will not treat a non-existing product instance as an error	false	Boolean

Example SOMDeleteProductInstance – use in workflow

This example deletes a product instance with service id 22314. If the product instance does not exist, the workflow node will *not* fail.

```
<Process-Node>
  <Name>SOMAssignResult</Name>
  <Action>
     <Class-Name>
       com.hp.ov.activator.mwfm.component.builtin.som.
         SOMDeleteProductInstance
     </Class-Name>
     <Param name="id" value="22314"/>
     <Param name="ignore_non_existing_instance" value="true"/>
  </Action>
</Process-Node>
```

SOMGetProductInstance

`com.hp.ov.activator.mwfm.component.builtin.som.SOMGetProductInstance`

This workflow node retrieves a product instance from Subscription Repository by its identifier. Optionally, the node can overwrite the current order id, request id, and request type with new values. In that case, it is important to note that the updated order id, request id, and request type are only stored in memory. The `SOMUpdateProductInstance` workflow node needs to be called in a subsequent state in order to store the new values in Subscription Repository.

Table 20 SOMGetProductInstance Parameters

Name	Required	Description	Default	Type
id	Yes	The service id of the Product instance to be retrieved.	None	String
order_id	No	An order id that may be associated with the product instance.	None	String
request_id	No	A request id that may be associated with the product instance.	None	String
request_type	No	A request type that may be associated with the product instance.	None	String
ignore_non_existing_instance	No	If set to “false” the node will fail if the product instance does not exist. Otherwise, if set to “true” the node will not treat a non-existing product instance as an error	false	Boolean
product_instance_var	Yes	The object case packet variable name where the retrieved product instance will be returned.	None	Object

Example SOMGetProductInstance – use in workflow

This example retrieves a product instance with service id = 22314 from Subscription Repository and stores the result in the case-packet variable `prodInstance`. If the product instance does not exist, the workflow node will fail.

```
<Process-Node>
  <Name>SOMGetProductInstance</Name>
  <Action>
     <Class-Name>
       com.hp.ov.activator.mwfm.component.builtin.som.
       SOMGetProductInstance
     </Class-Name>
     <Param name="id" value="22314"/>
     <Param name="product_instance_var" value="prodInstance"/>
  </Action>
</Process-Node>
```

SOMUpdateProductInstance

`com.hp.ov.activator.mwfm.component.builtin.som.SOMUpdateProductInstance`

This workflow node either stores a newly created product instance or it updates (replaces) an existing product instance in Subscription Repository. The `SOMUpdateProductInstance` workflow node should be used very carefully in action workflows because it may have a disrupting impact on the ongoing fulfillment processes (if updating the product instance for which the current fulfillment process is executing).

Table 21 **SOMUpdateProductInstance Parameters**

Name	Required	Description	Default	Type
product_instance	Yes	The product instance object to be stored or updated.	None	Object
order_id	No	An order id that may be associated with the product instance.	None	String
request_id	No	A request id that may be associated with the product instance.	None	String
request_type	No	A request type that may be associated with the product instance.	None	String
store	No	If this parameter is set to "true" the node will operate in strict *store* mode (as opposed to the default mode which is *update* mode). When running in *store* mode the node will fail if a product instance with an identical service id exists in Subscription Repository.	false	Boolean

Example **SOMUpdateProductInstance – use in workflow**

The following example stores (or updates) a product instance in Subscription Repository. The product instance is read from the case-packet variable `prodInstance`.

```
<Process-Node>
  <Name>SOMUpdateProductInstance</Name>
  <Action>
     <Class-Name>
       com.hp.ov.activator.mwfm.component.builtin.som.
         SOMUpdateProductInstance
     </Class-Name>
     <Param name="product_instance" value="prodInstance"/>
  </Action>
</Process-Node>
```

SOMUpdateProductInstanceState

`com.hp.ov.activator.mwfm.component.builtin.som.SOMUpdateProductInstanceState`

This workflow node updates the state of a product instance (optionally, including all child instances) and may (optionally) store the result in in Subscription Repository.

Table 22 SOMUpdateProductInstanceState Parameters

Name	Required	Description	Default	Type
instance	Yes	The (product) instance object for which to assign a new state value.	None	Object
state	Yes	The new value to set as the state of the instance	None	String
recursive	No	If set to “true” the state will be set on this instance and all its children; otherwise, the state will only be set on this instance.	false	Boolean
in_memory	No	If set to “true” the state will only be updated in memory; otherwise, it will also be written to Subscription Repository.	false	Boolean

Example SOMUpdateProductInstanceState – use in workflow

The following example sets the state of the product instance (including all child instances) stored in the case-packet variable `prodInstance` to *Active* and updates the product instance in Subscription Repository.

```
<Process-Node>
  <Name>SOMUpdateProductInstanceState</Name>
  <Action>
     <Class-Name>
       com.hp.ov.activator.mwfm.component.builtin.som.
         SOMUpdateProductInstanceState
     </Class-Name>
     <Param name="instance" value="prodInstance"/>
     <Param name="state" value="Active"/>
     <Param name="recursive" value="true"/>
  </Action>
</Process-Node>
```

Workflow Node for Accessing Trueview Inventory

TVWSRequest

`com.hp.ov.activator.mwfm.component.builtin.TVWSRequestNode`

This workflow node is used to send requests to and receive requests from Trueview Inventory via WS/SOAP. The node uses the `TrueviewModule` described on page 97 for the actual communication with Trueview.

The process for communicating with Trueview is the same in all cases, regardless of the operation. An input object is created using custom Java code, and this object is then passed to the `TVWSRequest` node (using the input parameter) along with an operation (using the operation parameter).

For a complete list of operations, please read the Trueview Javadocs and refer to Trueview's WSDL file.

Table 23 TVWSRequest Parameters

Name	Required	Description	Default	Type
module_name	Yes	The name of the Trueview Inventory module to be used.	None	String
operation	Yes	The name of the web service operation.	None	String
input	Yes	The request object to pass to the TrueviewModule.	None	Object
response	Yes	The case-packet variable in which to store the response object received from Trueview Inventory.	None	Object

Example **TVWSRequest – use in workflow**

The following example uses a HP Service Activator `Java` process node to create a `CreateCustomer` object which is subsequently passed to the `TVWSRequest` workflow node in order to create a customer in Trueview.

```
<Process-Node>
  <Name>Java</Name>
    <Action>
      <Class-Name>
        com.hp.ov.activator.mwfm.component.builtin.JavaNode
      </Class-Name>
      <Param name="expression" value="constant:func()"/>
      <Param name="in_scope" value="inputObject,JOB_ID"/>
      <Param name="javacode" value="constant:
       public void func()
       {
         com.tieroneoss.tnp.networkresources.CustomerT ct =
           new com.tieroneoss.tnp.networkresources.CustomerT();
         ct.setName(&quot;SOM_Demo&quot;);
         ct.setCustomerType(&quot;CUSTOMER&quot;);
         ct.setCustomerCode(&quot;SYSTEM&quot;);
         com.tieroneoss.tnpnml.CreateCustomer cc =
           new com.tieroneoss.tnpnml.CreateCustomer();
         cc.setCustomer(ct);
         inputObject=cc;
       }"/>
    </Action>
  <Next-Node>TVReq</Next-Node>
</Process-Node>
<Process-Node disablePersistence="false">
  <Name>TVReq</Name>
  <Action>
    <Class-Name>
      com.hp.ov.activator.mwfm.component.builtin.TVWSRequestNode
    </Class-Name>
    <Param name="input" value="inputObject"/>
    <Param name="module_name" value="trueview"/>
    <Param name="operation" value="createCustomer"/>
    <Param name="response" value="responseObject"/>
  </Action>
</Process-Node>
```

NOTE The value of the `javacode` parameter has been formatted in this example for improved readability. It is recommended that you use Java templates; read the documentation for the Java workflow node in the document *HP Service Activator Workflows and the Workflow Manager* for additional information.