HP Service Provisioner

User's and System Integrator’s Guide

Edition: V62-1A

**for Microsoft Windows® Server 2008 R2, HP-UX 11i v3,
Red Hat Enterprise Linux 6.4**

Manufacturing Part Number: None

October 18, 2013

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.

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.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

Document id: p158-pd026301

Contents

1 Introduction to HP Service Provisioner 7
Standards 8
Catalog Driven 9
Service Specifications and Service Instances 9
Service Orders and Product Instances 10

2 Fulfillment Processes 13
RFS State Transitions 15
Processing Direction 15
Rollback 16
Manual Design and Assign 18
Force Operations 18

3 Implementation Architecture 21

4 Installation 23
Deploying HP Service Provisioner 23
Configuring HP Service Provisioner 23
Installing HP Service Provisioner License 24
Localization 24
Deploy SOM Demo Solution 25

5 Client Integration 27
Northbound API 27
Order Entry UI 28

6 Editing the Service Catalog 31
Profiles 31
RFS Specifications 34
CFS Specifications 35
Product Specifications 36
Import and Export of Catalog Content 38

7 Monitoring and Interacting With Running Orders 39
Inspecting Service Orders and Product Instances 39
Performing Manual Design 41
Interacting with RFS State Transition Workflow Jobs 43

8 Processes for Resource Facing Services 45
RFS Workflow Contract 45
Storing Case-Packet Values into RFS Characteristic Values 46
Interaction with Subscription Repository 47
Integration With Trueview Inventory Integration 47

9 Workflow Manager Module and Node Library 49
Workflow Manager Modules 49
Workflow Manager Nodes 52

Contents

In This Guide

This guide provides a general decription 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.
- **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:

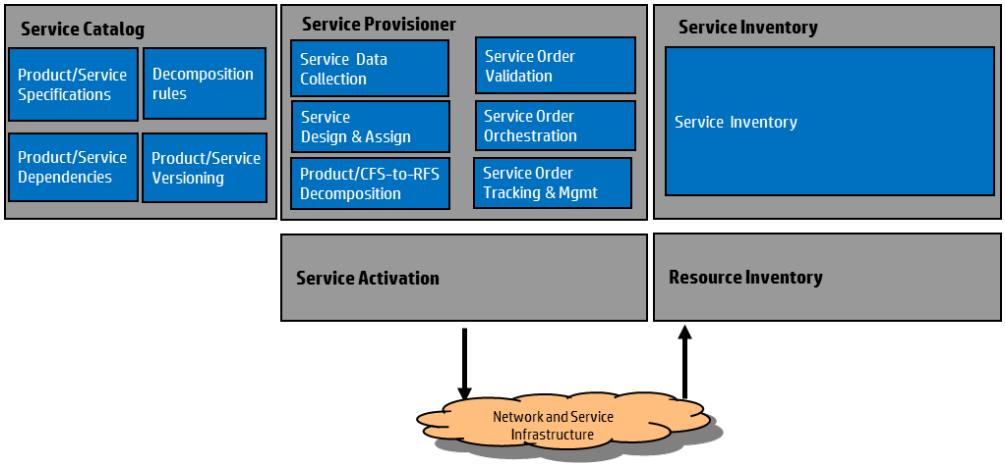
- *HP Service Activator, System Integrator's Overview*
- *HP Service Activator, User's and Administrator's Guide*
- *HP Service Activator, Workflows and the Workflow Manager*
- *HP Subscription Repository v1.3, User Guide*
- *HP Subscription Repository v1.3, Developer Guide*
- *HP Subscription Repository v1.3, Installation and Administration Reference*
- *HP Trueview 2.0, Install Guide*
- *HP Trueview 2.0, Admin Guide*
- *HP Trueview 2.0, 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.

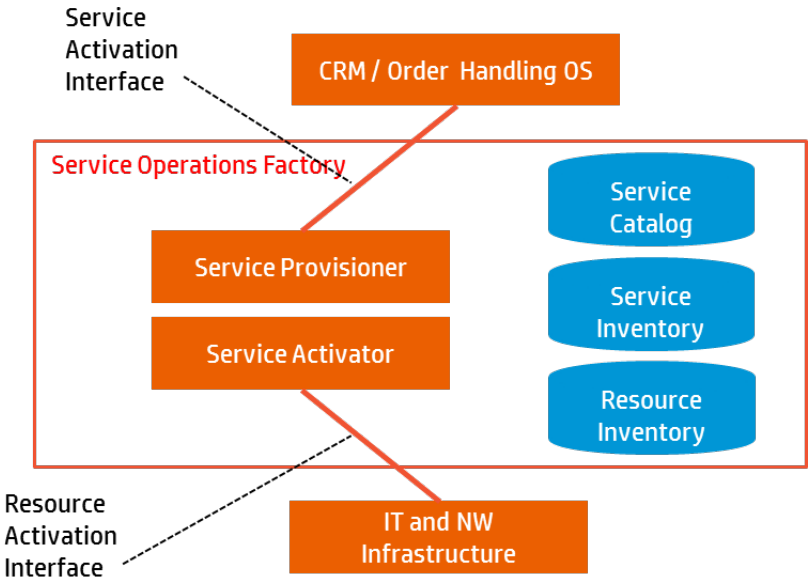
Figure 1 HP OSS Fulfillment



HP Service Provisioner is a framework that supports both development and deployment of a 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: Service Provisioner and Service Activator, and three data stores: the service catalog, the service inventory and the resource inventory. Service Provisioner interfaces to all three data stores and is closely integrated with 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 manual. 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 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 must support JMS. For more details about interactions on the interface including the contents of request and response 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 service 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.

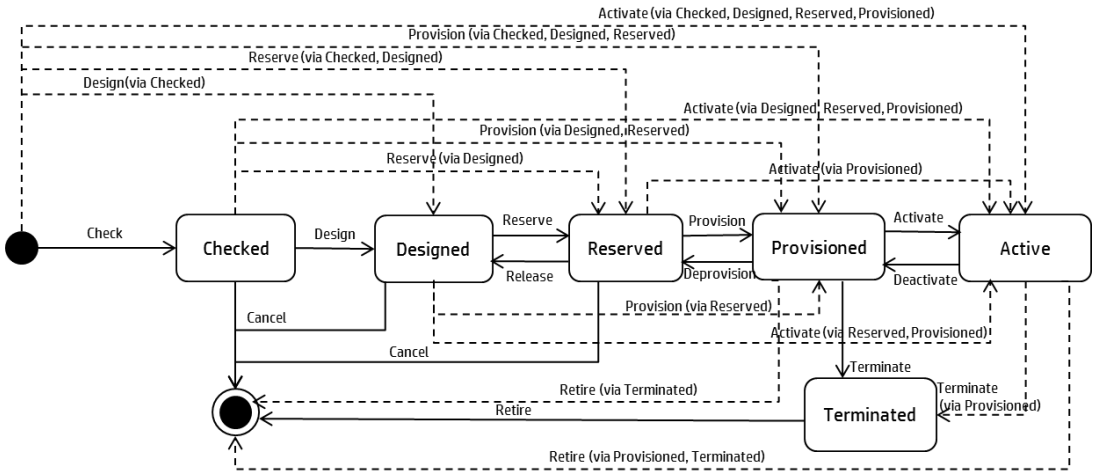
SID clearly distinguishes between the specifications of products and services and instances of them. We will maintain the distinction by using the abbreviations PS for product specification, and CFSS/RFSS for specifications of CFS and RFS, respectively. PSs, CFSSs and RFSSs all belong in the catalog. 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.

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 TMF518. 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.

The basis for service order processing requests is a state model. The life cycle of a service order and the resulting product instance goes through the states shown in Figure 3, starting at the middle left (filled circle) and ending when the service order has been cancelled or the resulting product instance retired at the lower left (circle with filled center).

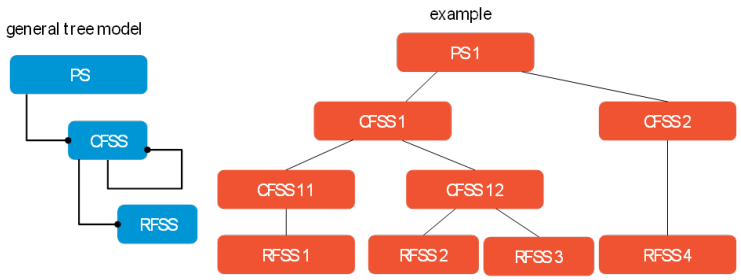
Figure 3 Life Cycle State of Service Orders and Product Instances



Requests supported on the NBI will always be for 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 will reference a product specification which is then looked up in the service catalog by HP Service Provisioner. The PS will be the root of a tree whose branches are CFSSs and RFSSs, and whose leaves are RFSSs. The tree structure of a PS in the catalog is illustrated in Figure 4, which shows the general model of the specification tree and an example. The service order will be created as a tree structure mimicking the specification structure in the catalog.

Figure 4 **Catalog Tree Structure**

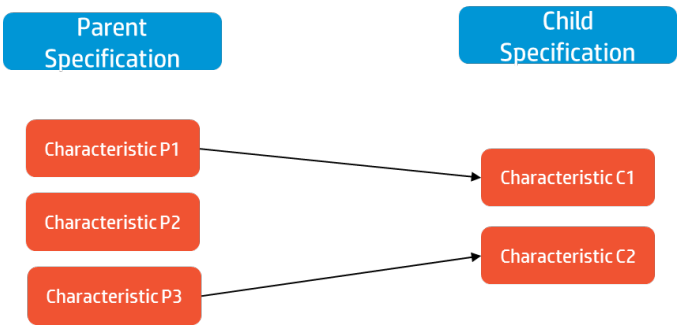


LEGEND	The bullet at the end of a line means multiple occurrences of that object can be connected to one object at the other end. This indicates the branching structure of the tree; in the example CFSS 11 and CFSS 12 are both children of CFSS 1.

When the service order reaches the state designed, a product instance, which is a tree with a similar structure, consisting of service instances, will be persisted in the service inventory (as a tree with a product root and a number of service instance objects as branches and leaves). Subsequent requests for state transitions will then reference the product instance at the root of the tree. For each product specified in the catalog, generally a large number of instances will be created over the time during which the product is orderable.

Every CFSS defines a CFS as consisting of (decomposable into) a number of child services, which can be CFS or RFS, by reference to their specifications (it follows that the service tree must be created bottom-up). 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 or CFSS 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 other way. Similar mappings exist between PSs and their contained (child) CFSSs. An example of an input mapping is shown in Figure 5. The parent has 3 characteristics, P1, P2 and P3. One child is shown; it has 2 characteristics, C1 and C2. The mapping assigns values to both child characteristics, taking them from two of the parent characteristics.

Figure 5 **Characteristic Input Mapping**



Characteristics may be *invariant*, i.e. with values defined once for all instances in the catalog. A collection of characteristics and their values may be defined as a *profile* (the TMF 518 term for this is *service template*). Profile characteristics are used to define fixed properties of a service offering. 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 profile for a subscriber broadband access service.

Each PS, CFSS or RFSS may include one profile. The same profiles may be reused in the definition of multiple products or services.

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 request, possibly through an input mapping from a parent service. In such a case the defined value is not invariant, but serves only as an overridable default.

2 Fulfillment Processes

The possible states of service orders and product instances were introduced in the section “Service Orders and Product Instances” above. The meaning of each state 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.

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.

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 inventory.

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 constituent services as controlled by the catalog, and “assign” means selecting and associating specific resources in the shared infrastructure with an RFS as done by the workflow which is associated with its RFSS in the catalog. 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.

Service orders are managed by requesting state transitions. Some requests imply sequences of transitions. For example it is possible to make a single 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.

The following requests are supported:

Check from non-existing to Checked; however, this request will not persist the product instance in service inventory, the service order is only cached for a period of time; hence the transition may be repeated on a multi-transition request; the purpose is only to retrieve the information whether the transition is feasible

Design from non-existing or Checked to Designed; may combine two transitions

Reserve from non-existing, Checked or Designed to Reserved; may combine two or three transitions

Cancel	from Designed or Reserved to non-existing; may combine two transitions
Provision	from non-existing, Checked, Designed or Reserved to Provisioned; may combine two, three or four transitions
Activate	from non-existing, Checked, Designed, Reserved or Provisioned to Active; may combine two, three, four or five transitions
Deactivate	from Active to Provisioned
Deprovision	from Provisioned to Reserved
Release	from Reserved to Designed
Terminate	from Active or Provisioned to Terminated; may combine two transitions
Retire	from Active, Provisioned or Terminated to non-existing; may combine two or three 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 depth first fashion.

NOTE

The 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 15.

The non-leaf branches of the tree are the root (product instance) and the CFS instance objects. Each non-leaf branch has a set of children at the next level of decomposition. The process for any branch has three steps:

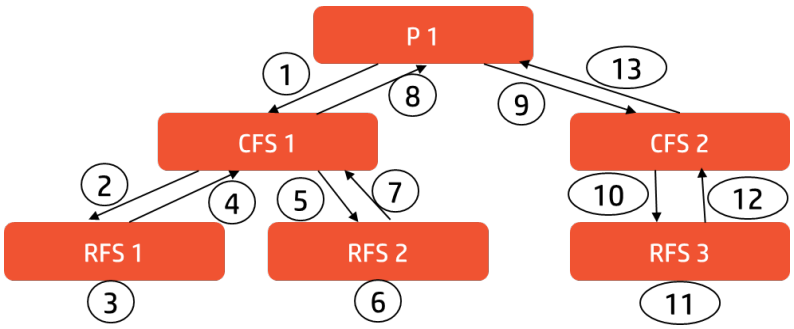
1. Perform the input mapping defined for the branch, to calculate characteristics values for all its child branches; characteristics for which no mapping is defined are not affected
2. Process the child branches in the order defined for the branch. For backward transitions, i.e. from Provisioned to Terminated or from right to left in Figure 3, the branches are processed in inverse order. For the transition to Checked this means creating the child branches, and for the transition to Designed it means persisting them. For RFS leaves it means executing the state transition workflow.

   Note, when the User Interaction flag is set on the request that is being processed by HP Service Provisioner or any of the catalog items contributing to the process instance tree, any transition to the Designed state (forward, backward or failure of the transition from Designed to Reserved) involves manual intervention allowing an operator to redefine the design, i.e. the tree structure for the product. See the section “Manual Design and Assign” below.
3. Perform the output mapping defined for the branch, to calculate impacts to the branch characteristics from updates that have occurred on those of its child branches.

For further description of input/output characteristics mappings, see the section “Product Specifications” in Chapter 5.

An example may help to understand the process. Figure 6 shows a product instance tree with only a single layer of CFSs. The actions performed during one state transition process are shown as tings/ovals with numbers indicating their sequence. Actions number 1, 2, 5, 9 and 10 are input mappings; actions number 3, 6 and 11 are executions of RFS state transitions workflows; finally actions number 4, 7, 8, 12 and 13 are output mappings.

Figure 6 **Tree Traversal Process for a State Transition**



RFS State Transitions

RFS state transition workflows play an important role. For simplicity of management a single workflow is associated with an RFSS in the catalog. This workflow must include the actions for all state transitions for an RFS. Each time it 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. Typical actions to include in the workflow will be, for the most important transitions as they apply to RFSs:

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 service)
Reserved → Designed or Provisioned → Terminated	remove recorded relationships between service inventory and resource inventory and release resources in resource inventory

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 6 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:

```
P 1 →
     CFS 1 →
            RFS 1 →
            RFS 2 →
     CFS 2 →
            RFS 3 →
```

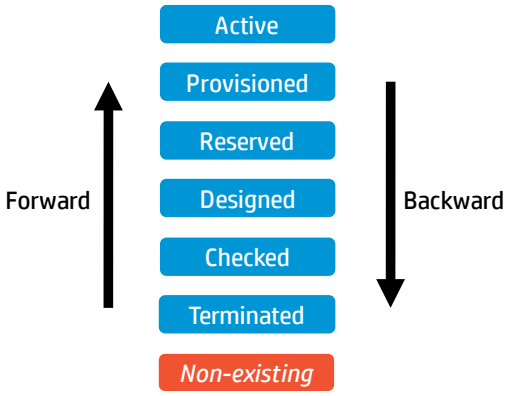
In the case of “backward” processing direction the processing order of the example tree shown in Figure 6 will be as follows:

```
P 1 →
     CFS 2 →
            RFS 3 →
     CFS 1 →
            RFS 2 →
            RFS 1 →
```

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 transition that move towards a *lower* maturity will enable “backward” processing direction.

Figure 7 shows all states ordered by their maturity levels, with the highest maturity level listed at the top and the lowest maturity level the bottom. The two arrows indicate the CFS/RFS processing direction when traversing from one state to another.

Figure 7 Maturity Levels and Processing Direction



Rollback

An attempted state transition may fail in the workflow process of an RFS. When such a failure occurs, the HP Service Provisioner engine will attempt to roll back the processing performed for the same state transition on other RFSs which are part of the product instance. For this reason RFS workflows must generally support the rollback mode. The final state of the service order/product instance will be the one where the failed transition started. Note this implies that a request for multiple state transitions can be partially fulfilled, ending up in an intermediate state different from both start and end states of the request.

During rollback, the CFSs and RFSs that have been successfully processed until the point of failure will be processed once again, albeit in reverse order (and in “rollback” mode). So, in the example shown in Figure 6, if an error occurs during the processing of the workflow for “RFS 3”, the engine will process “CFS 1” again which, in turn, will cause the workflows for ”RFS 2” and ”RFS1” to be called again (in that order) – see the following example:

```
P 1 →
     CFS 1 →
            RFS 1 →
            RFS 2 →
     CFS 2 →
            RFS 3 → ERROR (result = 1)
     CFS 1 ↩
            RFS 2 ↩
            RFS 1 ↩
P 1 ↩
```

LEGEND	The → symbol indicates a "roll forward" operation and the ↩ symbol indicates a "rollback" operation.
IMPORTANT	The failing RFS workflow ("RFS 3" in the example shown above) will *not* be called again with "rollback" mode enabled. Hence, in case of errors in the workflow it must attempt to revert all its actions before terminating.

If an error occurs in during a "backward" state transition the procedure will be similar. I.e. the CFSs and RFSs that have been successfully processed until the point of failure will be processed once again, albeit in reverse order (and in "rollback" mode); see the following example:

```
P 1 →
     CFS 2 →
            RFS 3 →
     CFS 1 →
            RFS 2 →
            RFS 1 → ERROR (result = 1)
            RFS 2 ↩
     CFS 2 ↩
            RFS 3 ↩
P 1 ↩
```

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

```
P 1 →
     CFS 1 →
            RFS 1 →
            RFS 2 →
     CFS 2 →
            RFS 3 → ERROR (result = 1)
     CFS 1 ↩
            RFS 2 ↩ ERROR (result = 1)
```

In this case the "RFS 2" workflow fails when executed in "rollback" mode and this brings the rollback operation to a halt. This means that manual action may be required to clean up the changes caused the "RFS 2" and "RFS 1" workflows and to bring the state of the product instance back into a consistent state.

Finally, there can be cases where rollback is 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:

```
P 1 →
     CFS 1 →
            RFS 1 →
            RFS 2 ↯ ERROR (result = 2)
```

Also in this case, manual action may be required to clean up the changes caused the "RFS 2" and "RFS 1" workflows and to bring the state of the product instance back into a consistent state.

Manual Design and Assign

Design of a product means deciding its components, i.e. the tree of CFSs and RFSs. In general the design is determined once and for all in the catalog, so that the state transition to the Designed 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 or RFSSs), or it can be specified by a parameter in the 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 (as described in the previous section at Figure 6), 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 Chapter 1.

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. Or to change their characteristics.

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 RFS state transition 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 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 workflow algorithm for the state transition from Designed to Reserved a user interaction that works as follows:

1. The 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 a set of force operations that support the following actions:

- **Set the state** of a product instance (and its children) to any of the possible service order states. No RFS workflows will be executed; *only* the state is set.
- **Delete** a product instance from Subscription Repository. No RFS 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 request to HP Service Provisioner SOAP interface with the “force” element set to ‘true’; the request must also contain a service id to identify the product instance. A list of supported force operations is shown in Table 1. Note that the effect of the force operations completely ignores the current state of the product instance.

Table 1 List of Supported Force Operations

Request	Effect
Cancel	Delete the product instance from Subscription Repository.
Retire	Delete the product instance from Subscription Repository (identical to *Cancel*).
Design	Set the state of the product instance to Designed.
Reserve	Set the state of the product instance to Reserved.
Provision	Set the state of the product instance to Provisioned.
Activate	Set the state of the product instance to Active.
Terminate	Set the state of the product instance to Terminated.

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 RFS state transition 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` and `SOMAction`) that execute in the workflow engine of HP Service Activator. The roles of the two 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).
- **`SOMAction`** is responsible for performing a depth-first traversal of all CFSs and RFSs in a product instance and for invoking the processed associated with RFSs (which are also implemented as workflows).

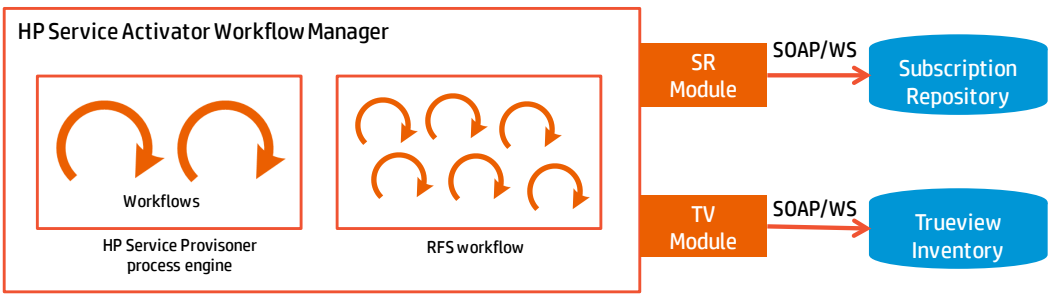
The interaction between the HP Service Provisioner process engine and RFS workflows happens simply by running the latter within the same workflow engine directly under the control of the process engine workflow as the parent job.

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 RFS state transition 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 8, 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 9 on page 49.

Figure 8 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 windows, one for editing the catalog, described in Chapter 5, and one for monitoring and interacting with running orders, described in Chapter 1. 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, and JMSSenderModule)

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 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 the 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 configure three workflow manager components as described in this section.

SRModule and TrueviewModule Configuration

NOTE	Configuration of the SRModule 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 9 (page 49).

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==`.

JMSSenderModule Configuration

The workflows implementing the service order management processes require that HP Service Activator has the `JMSSenderModule` enabled. Refer to *HP Service Activator, Workflows and the Workflow Manager* for details of how to enable and configure the module.

IMPORTANT	The name of the `JMSSenderModule` *must* be `som_jms_sender_queue`.

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 `$JBOSS_ACTIVATOR/WEB-INF/classes/jsf-resources`. When you add support for a new locale, you must also add that locale in file `$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 23.

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

- Five RFS 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 RFS workflows.
- Two JMS test workflows; one for sending dummy messages and one for listening to messages. In order to use the JMS listener workflow, you need to configure a `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 38.

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 second section.

Northbound API

The northbound API of HP Service Provisioner is a web service interface implemented as SOAP over HTTP. 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 over 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 state transitions, and when all have been processed successfully, or when processing of the request ends with a failing transition, the final response message is sent.

Remember all requests refer to a PS in the service catalog. Requests received on the northbound interface have the following contents:

product name and version	identifies the PS in the catalog; this is the starting point for retrieving the complete tree of specifications; name and version must be repeated consistently when multiple requests refer to the same service order / product instance
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
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 (HP Service Activator) user interface to allow operators to locate activity from a customer order
request id	client’s identifier of the request, returned in all responses to allow client to understand what is being responded to

request type	the requested state of the service order; together with current state of the service order/product instance it identifies the sequence of state transitions to perform
manual	optional flag; 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)
priority	optional, indicates priority relative to other requests
customer	name of customer, for display on the user interface
force	optional flag, when set the processing engine enter “force” mode which means that product instances can be forcefully deleted or forced into any state. RFS workflows will not be called when operating in force mode. For more information, please read the Section “Force Operations” on page 18.
characteristics	list of names and corresponding values of product characteristics; these values are the parameters of the service order

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 on final state transition, but rollback successful, 2 for error in final state transition and unsuccessful rollback
- explanatory result text (a single string)
- a data structure with completion information per state transition, for each transition containing information for each service in the tree

Requests to HP Service Provisioner must be sent to the 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 JMS sender module with the name `som_jms_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.

A templates file for the asynchronous response messages is found in the folder `$ACTIVATOR_OPT/solutions/SOM/etc/template_files`.

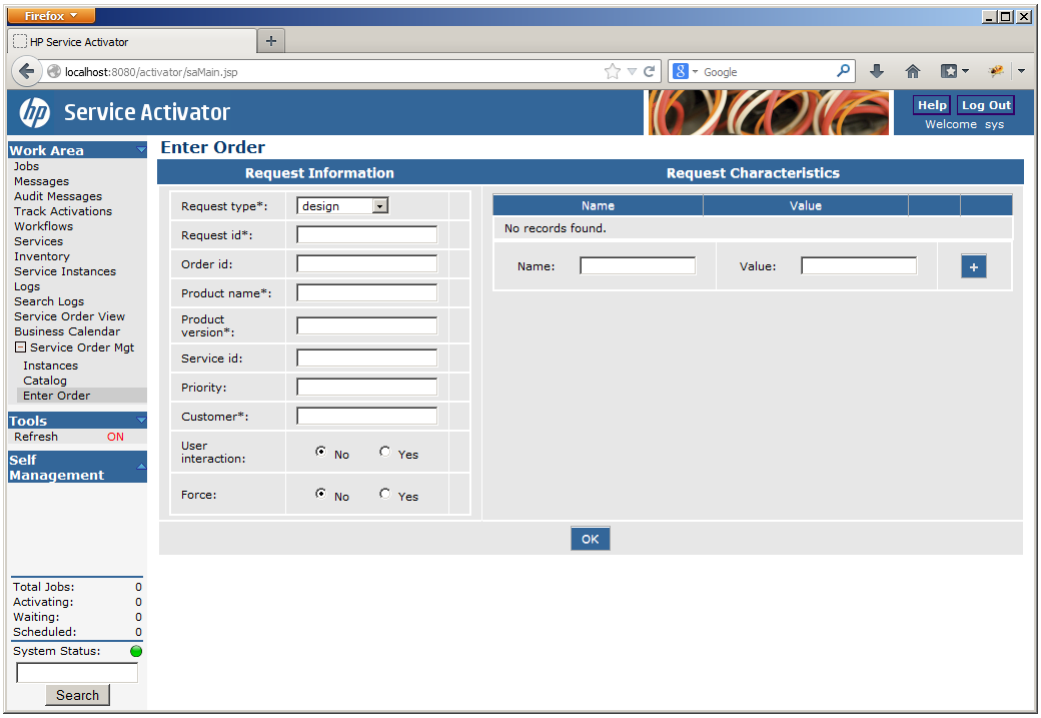
Order Entry UI

Even for orders that are entered manually, asynchronous JMS messages will be sent. To track progress of manually entered orders, use the user interface as described in Chapter 1.

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. It is shown in Figure 9. Select first Service Order Mgt 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 9 **Manual Entry of Service Order**



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 four kinds of items to define in the service catalog: profiles, RFSSs, CFSSs and PSs. Because of dependencies between them, they must in the practical editing process generally be defined in the order mentioned, i.e. bottom up.

All items in the catalog have three 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 invariant. 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 RFSS, CFSS or PS. Editing of characteristics is similar in all those cases and explained (only) in the “Profiles” section.

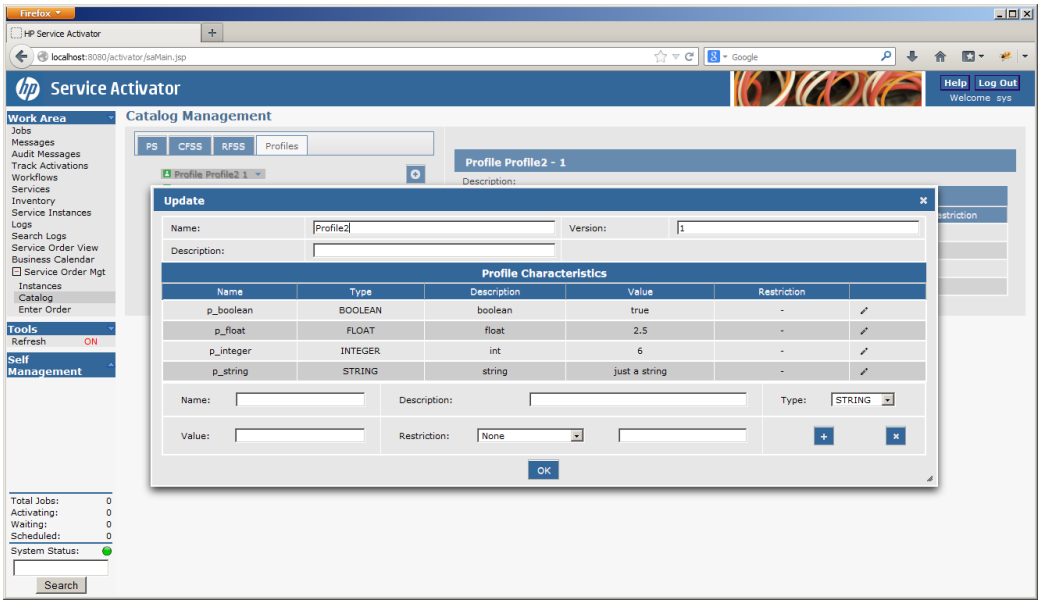
The screen window for editing the catalog is included in the user interface of HP Service Activator as an item that can be selected in the Work Area menu. This is shown in Figure 10 and the following figures. Select first Service Order Mgt and then Catalog, the Catalog Management window will then appear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

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 one profile, but not several. 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 a product or product specification, to overrule the value from the profile.

Figure 10 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).

Figure 10 **Catalog Management – Profile Specification**



To create a new profile definition, click the plus-in-a-circle button under the bar with the tabs, and a Create window will pop up. It is very similar to the Update window that is explained above Figure 10, and works in the same way, but the Name and Version fields are editable and mandatory to fill. Of course, the list of characteristics will initially be empty. A profile must contain at least one characteristic.

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

Once a profile has been defined in the catalog, it can be referenced in 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 10. At the top of the window are fields showing the name, version and description attributes of the profile. Only the description can be changed in the Update window. Next, the window contains a list of the currently defined characteristics, showing for each one, its name, description, type, value and restriction and an edit button.

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 (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 x button.

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

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 default. For characteristics of type string, empty values are allowed.

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

Restrictions are optional and selected from a drop-down list after the type. A specified restriction will be applied to characteristics values when they are received, as service order parameters on the northbound interface, as results from RFS state transition workflows, from characteristics mappings, or from manual entry.

Available restrictions (types) are:

string	enumeration, length, regular expression
integer, float	minimum, maximum, interval
date	pattern

When a restriction type 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 (more details can be found in the Section “Restrictions” on page 33).

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

Restrictions

As described above, it is possible to define restrictions for characteristics. Restrictions are divided into two parts; a restriction type and a restriction value.

The supported restriction types are (listed by characteristic types)

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

After having selected a restriction type (other than “NONE”) a restriction value must be entered. 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 / Float minimum:** An integer value
- **Integer range:** Two integer values separated by a slash (“/”); e.g. “10/200”

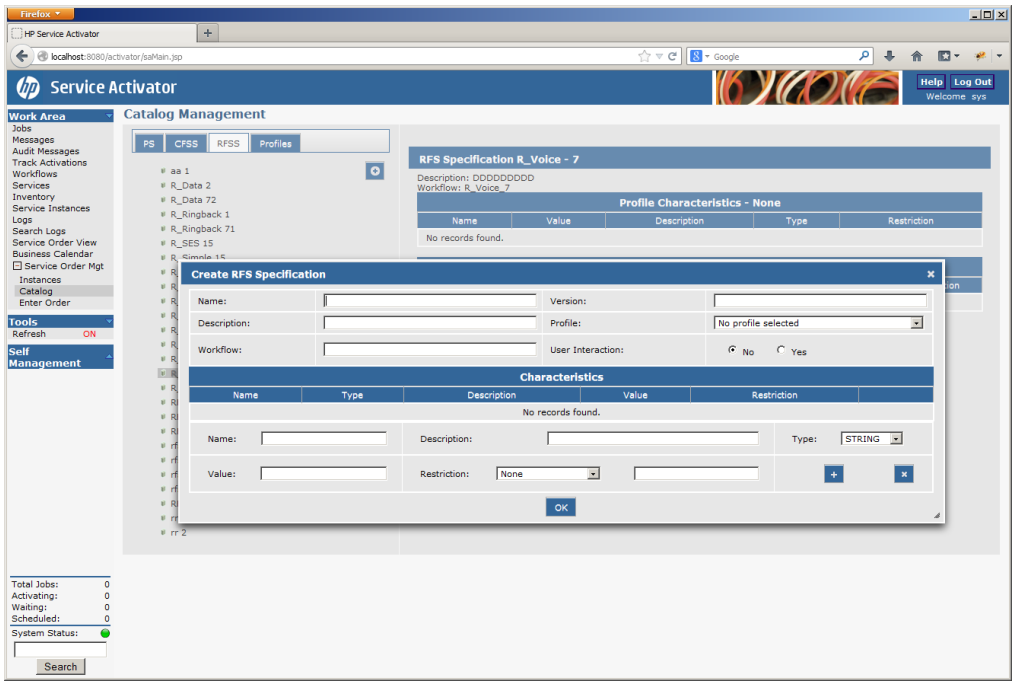
RFS Specifications

Figure 11 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 profile. Profile is optional, note that the RFSS shown in Figure 11 does not have a profile associated. There are no mappings of characteristics, as an RFS 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 RFSSs 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 designed state to finish with a manual action, for any product that includes the specified RFS. It has a similar effect when it appears on a CFSS or on a PS.

Figure 11 Catalog Management – RFS Specification



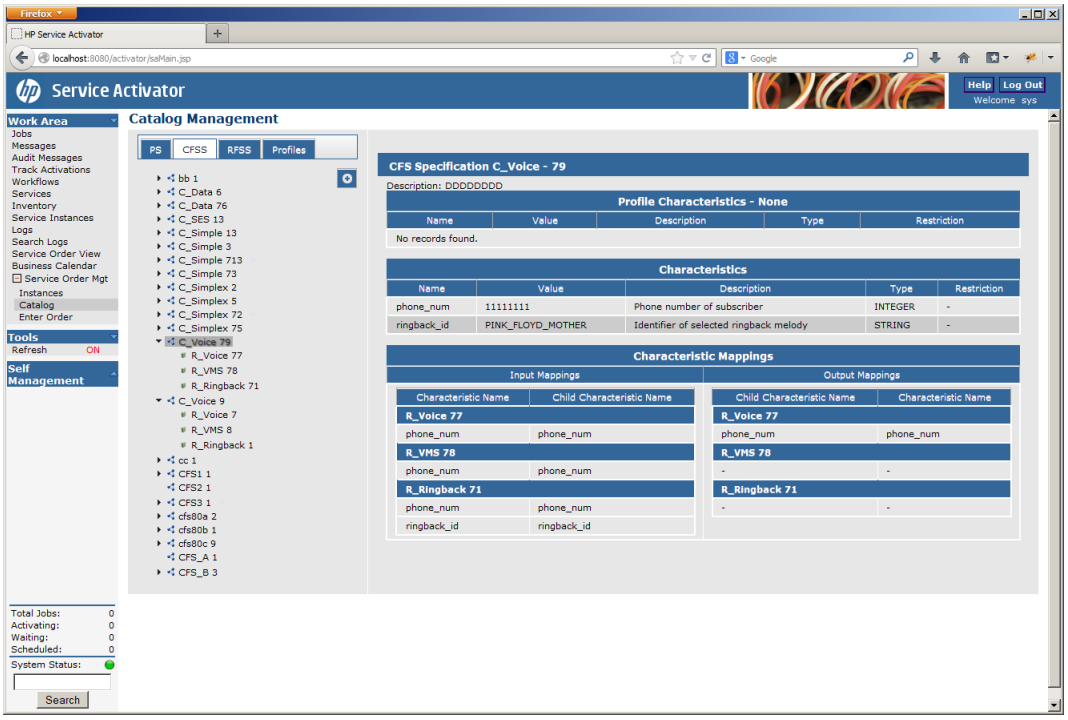
CFS Specifications

CFSSs are quite similar to RFSSs. They can also have contained (child) service specifications, which can be either CFSS or RFSS, and for each child specification input and output mappings of characteristics will exist. Unlike RFSSs they do not have associated workflows.

Figure 12 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, you can edit or delete existing CFSSs and create new ones as described in the section “Profiles” above. In addition you can edit the list of child service specifications and the associated mappings; please refer to the section “Product Specifications” below, where those steps are described for the very similar case.

Figure 12 Catalog Management – CFS Specification



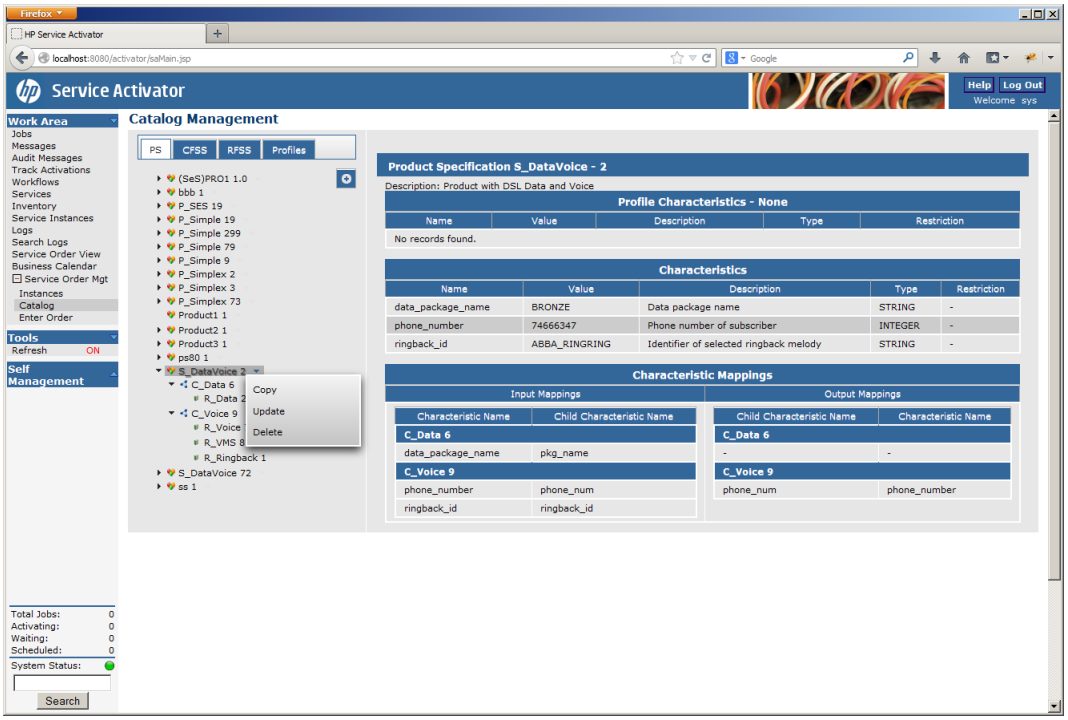
Product Specifications

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

Figure 13 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 profiles, you can edit or delete existing PSs and create new ones as described in the section “Profiles” above. The editing steps for the attributes and characteristics are very similar. Figure 13 also shows the menu which appears when you click the arrow button to the right of the PS.

Figure 13 Catalog Management – View Product Specification



With PSs (and with CFSSs) you can also edit the list of child service specifications. For a PS all children must be CFSSs, for a CFSS there can also be RFSSs. Figure 14 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 (shown in Figure 13). 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’ bar. The ‘Available’ column shows 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 you have 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 ->| or empty the list of included by clicking on |<-.

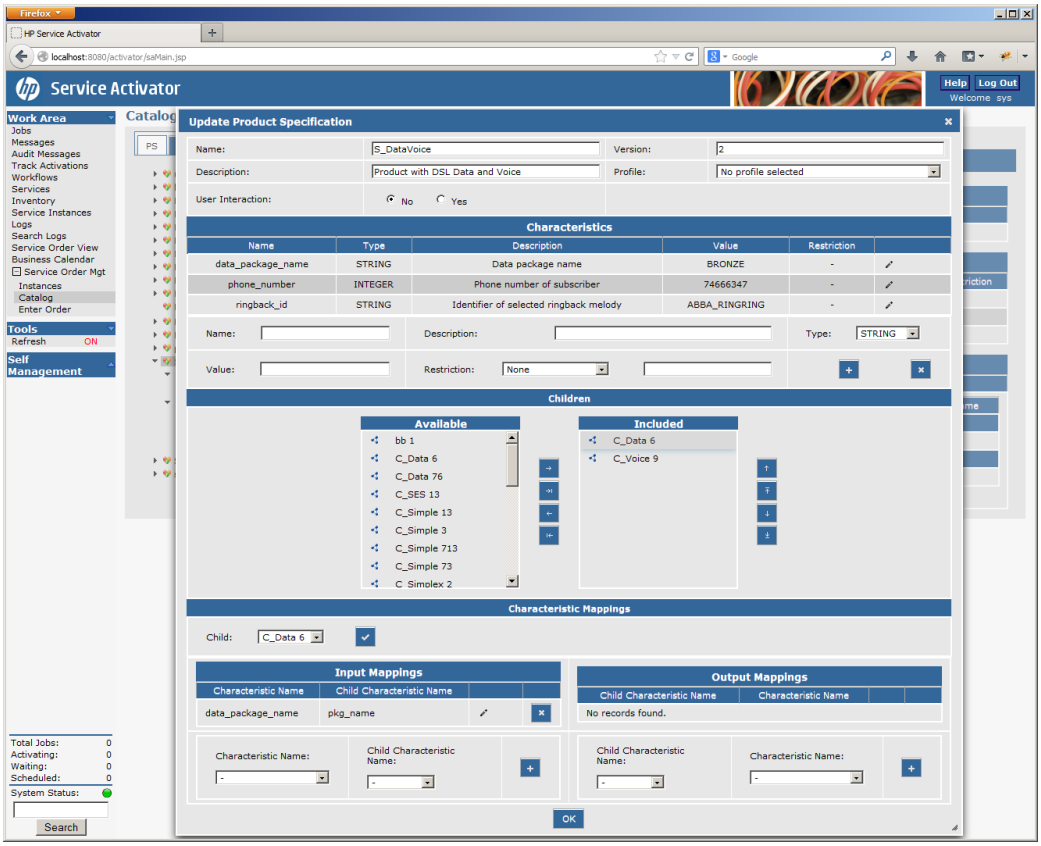
Remember the order of the children is significant. It determines the order in which they are processed during state transitions. Select a child and use the up/down buttons on the right to move it up or down in the sequence.

When you have selected and ordered the child specifications, work in the area at the bottom of the window to edit the characteristic mappings. The PS or CFSS 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, select it in the list and click the x button. 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 complete PS (or CFSS) definition, click the OK button to commit it.

Figure 14 **Catalog Management – Update Product Specification**



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 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 local)
`-verbose`	generates verbose output
`-help`	outputs usage information

Exactly one of export and import 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 and product/service specifications 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, to make entry to the state complete.

Finally, depending on how a specific solution is implemented, RFS state transition 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.

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

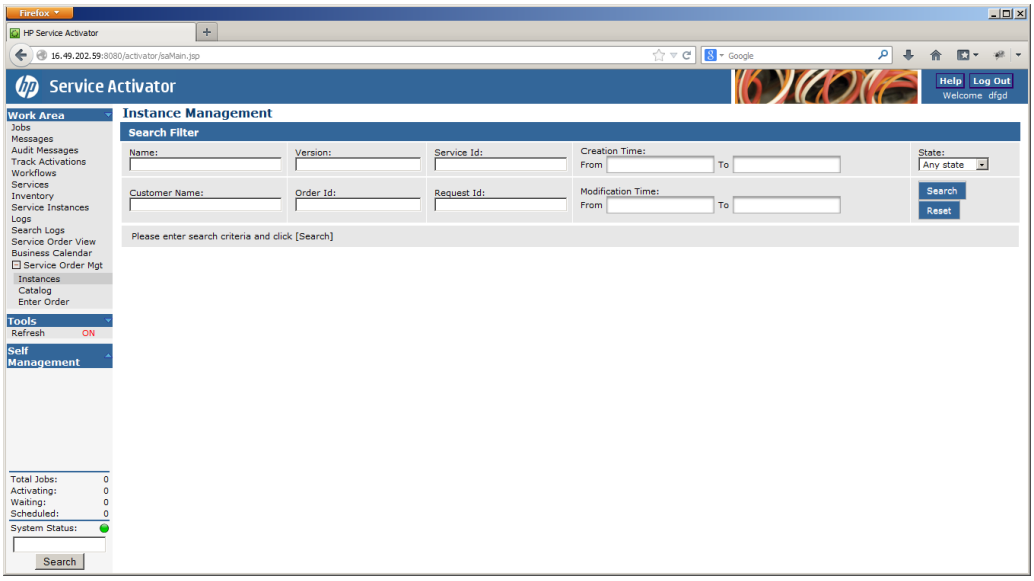
Inspecting Service Orders and Product Instances

Figure 15 shows the Instance Management window which appears when Instances has been selected in the submenu under Service Order Mgt. Initially the window shows the search filter.

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

- product name and version, order id, request id, customer name – 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.

Figure 15 **Instance Management – Search Filter**



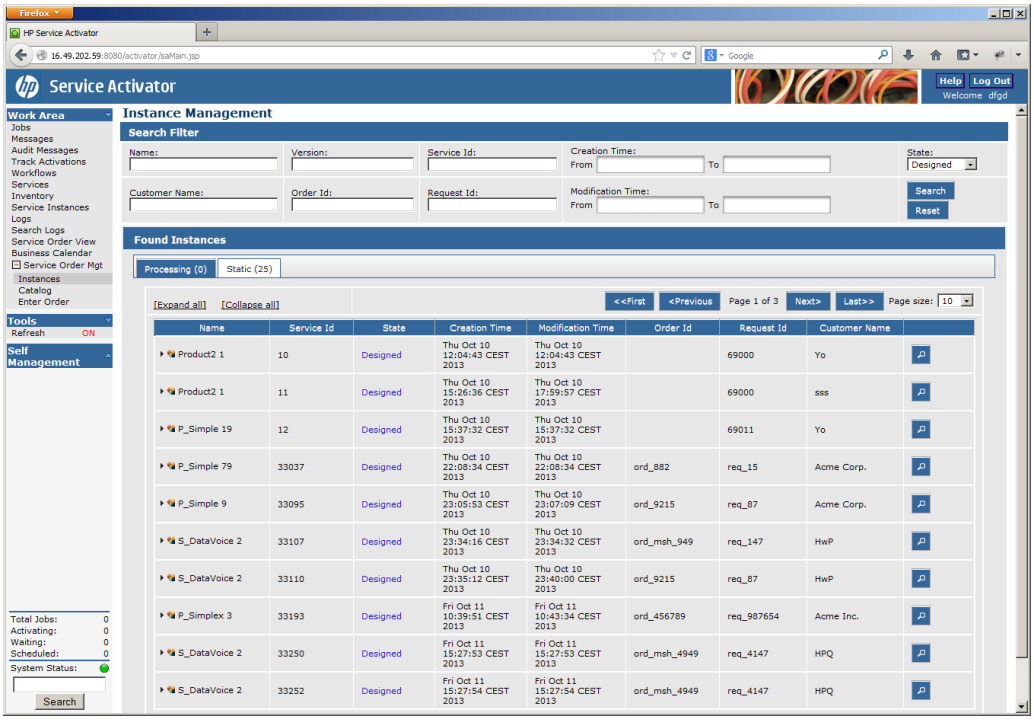
Once the results of a search are retrieved, you can choose by the tabs to view them from a processing perspective or from a static perspective. The static perspective can be used even when there is no order processing activity, typically for active services; you will see the data retrieved from the service 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 for an example.

The view will be very similar regardless of the selected perspective. It shows – see for an example – all the product instances that match the search filter, with their breakdown into CFSs and RFSs 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.

By clicking the looking glass icon you can launch a popup window, as shown in Figure 16, 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 16 **Instance Management - Static View**



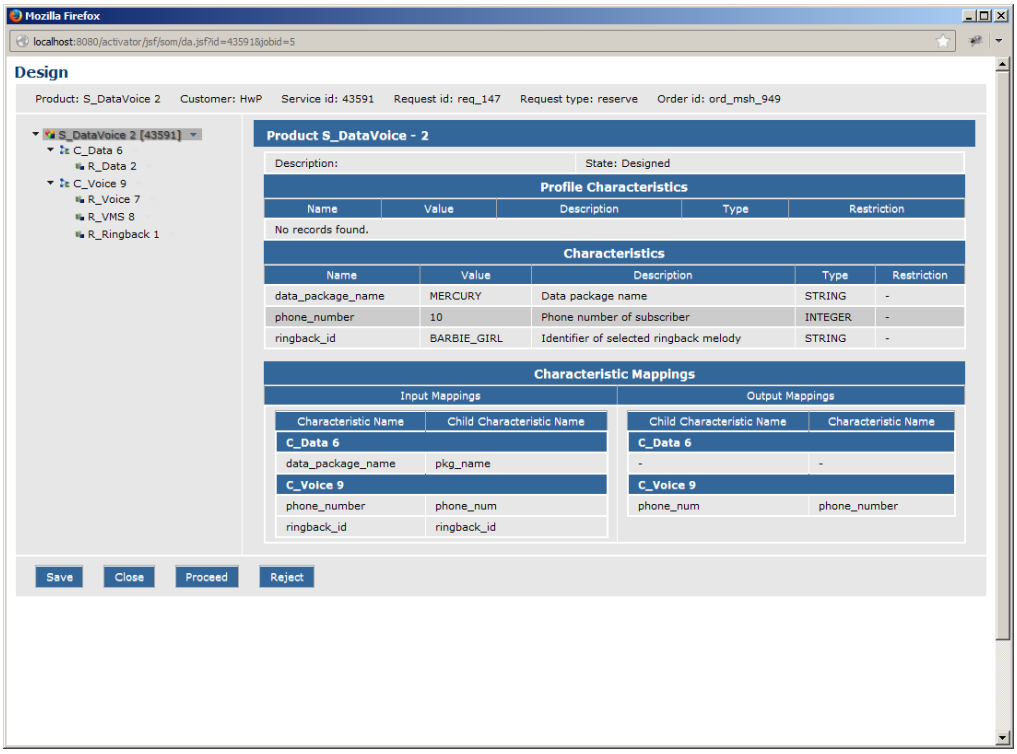
Performing Manual Design

When the user interaction flag is active on a service order, all state transitions into the Designed state allow 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 popup details window that can be launched from the service order processing view as discussed above. An example of the window itself is shown in Figure 17.

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 CFS and RFS from the tree. CFS or RFS 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 17 **Service Order Details – Manual Design**

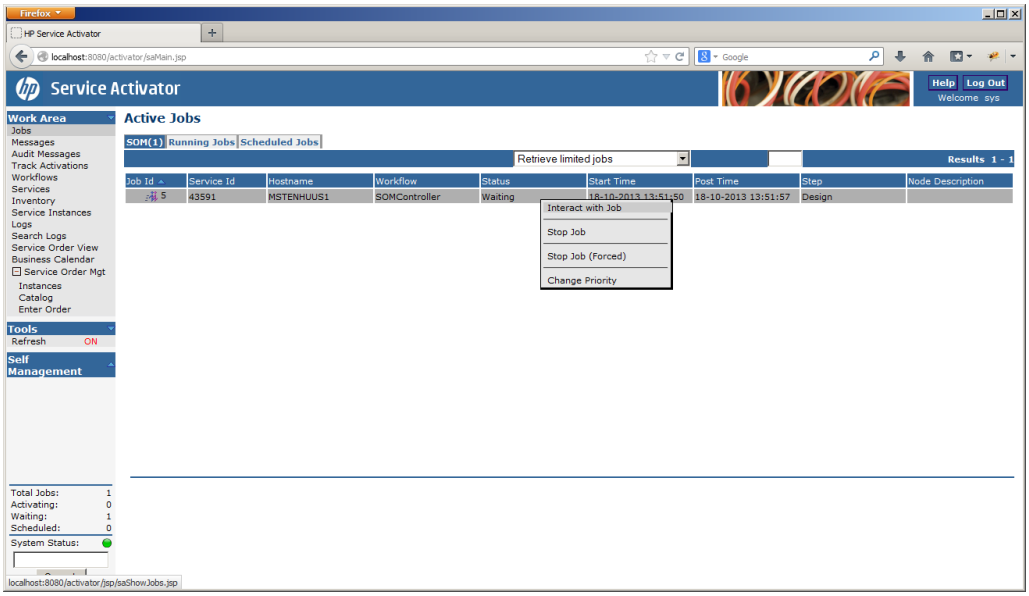


When manual design is complete, press the Proceed button to allow the overall automated service order process to continue. If more work is needed before letting the process continue, press the Save or Close button and return to the order at a later time.

Pressing the Reject button will cause the transition to fail and roll back.

An alternative way, which may be a shortcut, to launch the Manual Design window is to interact with HP Service Activator which runs the HP Service Provisioner processing engine (`SOMController`). View running jobs in the HP Service Activator Jobs list, select the SOM tab, right-click on the `SOMController` job in the Design step and select ‘Interact with Job’. See Figure 18 for an example of the Jobs list.

Figure 18 **HP Service Provisioner Engine in HP Service Activator Jobs List**



Interacting with RFS State Transition Workflow Jobs

State Transition 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, the section “Manual Design and Assign”.

More information and specific instructions must be provided per solution.

8 Processes for Resource Facing Services

This chapter is primarily of interest for the system integrator who will implement the technical processes to be associated in the service catalog with RFSSs.

For each RFS a workflow is required which implements the processes for all state transitions. All these processes must be combined in a single workflow. Different algorithms must be implemented for all the state transitions which require some action. The workflow must therefore begin with branching logic based on the target state, with a sub-branch for those states that can be reached by more than one transition.

All the inputs to the workflow are specified below as part of the workflow contract. In addition to the target and starting state the main inputs are the characteristics for the RFS, with the values they have after input mapping has been performed.

In addition to the inputs the 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 SA 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 RFS workflow by updating values of the case-packed variable holding the characteristics.

For a description of the typical work in state transition algorithms, please read the Section "RFS State Transitions" on page 15.

RFS Workflow Contract

All RFS workflow must have a contract defined; otherwise they cannot be used by the HP Service Provisioner workflows. The RFS workflow contract's input and output parameters are described in Table 2.

Table 2 RFS Workflow Contract

Input Parameter	Type	Description
`parentJobId`	Integer	Used to hold the job id for the workflow that spawned this RFS workflow. Since the RFS workflow is, technically speaking, running as a "macro job" it is currently not used.
`syncToParent`	Boolean	Used to indicate to the RFS workflow whether it needs to sync back to the caller workflow upon completion. In the current version, this parameter is always set to 'false'.
`rollback`	Boolean	Indicates whether or not this RFS workflow is called as part of a rollback operation. If set to 'true' this is a rollback operation; otherwise, if is a "roll forward" operation.

name	String	Contains the name of the RFS that is currently being processed.
version	String	Contains the version of the RFS that is currently being processed.
parameters	Object	Contains a map of all characteristics that are visible to this RFS. 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 that trigger the current fulfillment process (e.g. `activate`, `design`, `reserve`, etc.).
customer	String	Contains the name of the customer.
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 RFS workflow; of interest to HP Service Provisioner are the following system case-packet variables:

- *SERVICE_ID* – this case-packet variable that contains the identifier 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 RFS workflows.
- *SOM_PATH* – this case-packet variable contains a pointer to the RFS within the "product instance tree" that is currently being processed. This variable must *never* be modified by RFS workflows.

For examples of RFS workflows with workflow contracts, see the 5 RFS sample workflows (their names starting with `R_`) that are part of the "SOM Demo" solution; see also, the Section "Deploy SOM Demo Solution" on page 25.

Storing Case-Packet Values into RFS Characteristic Values

When in RFS 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 RFS characteristics. 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 RFS characteristic named bandwidth, and the case-packet variable card is stored in the RFS characteristic of the same name.

Interaction with Subscription Repository

IMPORTANT

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

It is possible to interact with Subscription Repository through the SRModule described in Chapter 9. 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.

All these workflow nodes should be used with utmost care (except SOMGetProductInstance).

Integration With Trueview Inventory Integration

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 (through 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 on page 57 for a full description of how to use it (including an example).

9 Workflow Manager Module and Node Library

This chapter lists the workflow manager modules and workflow nodes that are included in the HP Service Provisioner solution 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. However, the additional nodes are meant for internal use only, and their functionality may be changed without notice.

Workflow Manager Modules

SRModule

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

This module communicates with Subscription Respository via WS/SOAP. To use the module to interact with Subscription Repository, you need to use the workflow nodes described in the Section “Workflow Manager Nodes” on page 52.

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 3 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 (if the *encrypted_password* parameter is set to ‘true’).	None

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. (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 never be set 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 threads that will be created to process requests.	1
max_threads	No	The maximum number of threads that will be created to process requests. This is the number of simultaneous requests that can be processed. Other requests will be queued until one of the threads becomes available.	3

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 57.

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

Table 4 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. The password may either be plain text or encrypted (if the *encrypted_password* parameter is set to 'true').	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 threads that will be created to process requests.	1
max_threads	No	The maximum number of threads that will be created to process requests. This is the number of simultaneous requests that can be processed. Other requests will be queued until one of the 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 Manager Nodes

SOMAssignResult

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

This workflow node writes a number of results from an executing RFS workflow to the RFS instance that currently processing. The node *must* be from RFS workflow because it relies on specific values of system case-packet variables set by the "SOMAction" and "SOMController" workflows.

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

Table 5 SOMAssignResult Parameters

Name	Required	Description	Default	Type
variable0, variable1, ... variable	Yes	Case-packet variables which are to be assigned to RFS characteristics (of identical names, unless *paramN* specified).	None	Any
param0, param1, ... paramN	No	Name of the RFS 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

```
<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>
```

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 a product specification object.

Table 6 SOMCreateProductInstance Parameters

Name	Required	Description	Default	Type
product_specification	Yes	The object holding the production specification from which to create a product instance.	None	Object
id	Yes	The identifier for the product instance to be created.	None	String
customer	Yes	The name of the customer to identify with this product instance.	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
product_instance_var	Yes	The object case packet variable name where the created Product Instance will be returned.	None	Object
in_memory	No	If set to 'true' the product instance will only be created in memory; not in Subscription Repository. If set to 'false' the product instance will be created in Subscription Repository and *in* memory (referenced by the case-packet variable assigned to the node parameter *product_instance_var*).	false	Boolean

Example SOMCreateProductInstance – use in workflow

```
  <Process-Node>
    <Name>SOMCreateProductInstance</Name>
    <Action>
       <Class-Name>
  com.hp.ov.activator.mwfm.component.builtin.som.SOMCreateProductInstance
       </Class-Name>
       <Param name="customer" value="customer"/>
       <Param name="id" value="serviceId"/>
       <Param name="product_instance_var" value="prodInstance"/>
       <Param name="product_specification" value="prodSpec"/>
    </Action>
  </Process-Node>
</Nodes>
```

SOMDeleteProductInstance

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

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

Table 7 SOMDeleteProductInstance Parameters

Name	Required	Description	Default	Type
id	Yes	The identifier of 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

Example SOMDeleteProductInstance – use in workflow

```
<Process-Node>
  <Name>SOMAssignResult</Name>
  <Action>
     <Class-Name>
com.hp.ov.activator.mwfm.component.builtin.som.SOMDeleteProductInstance
     </Class-Name>
     <Param name="id" value="instanceId"/>
  </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 order to store the new values in Subscription Repository.

Table 8 SOMGetProductInstance Parameters

Name	Required	Description	Default	Type
id	Yes	The identifier of the Product instance.	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
product_instance_var	Yes	The object case packet variable name where the retrieved Product Instance will be returned.	None	Object
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 **SOMGetProductInstance – use in workflow**

```
<Process-Node>
  <Name>SOMGetProductInstance</Name>
  <Action>
     <Class-Name>
com.hp.ov.activator.mwfm.component.builtin.som.SOMGetProductInstance
     </Class-Name>
     <Param name="id" value="id"/>
     <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) and existing product instance in Subscription Repository. The `SOMUpdateProductInstance` should not be used in RFS workflows because it may have a disrupting impact on the ongoing SOM processes.

Table 9 **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"; i.e. if a product instance with an identical identifier exists in Subscription Repository, then the node will fail.	false	Boolean

Example **SOMUpdateProductInstance – use in workflow**

```
  <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>
```

SOMValidateRequest

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

The primary role of this workflow node is to validate a service request and as a result return a product specification as an output parameter. In can, however, also be used simply to look up a product specification object in the catalog based on a product name and (optionally) version.

Table 10 **SOMValidateRequest Parameters**

Name	Required	Description	Default	Type
product_name	Yes	The name of the product specification in the catalog.	None	String
product_version	No	The version of the product specification. If this parameter is not specified the most recent version of the product specification will be used.	*Most recent version*	String
request_type	Yes	The request type for the service request.	None	String

product_specification_var	Yes	The case-packet variable in which to return the product specification object.	false	Object
user_interaction	Yes	Boolean input and output variable to control whether or not manual user interaction will be required. If the value of the input parameter is 'true' it will not be changed by this node. If, on the other hand, the value of the input parameter is 'false', the output value will be set to 'true' *if* the product specification has 'user interaction' enabled.	false	Boolean

Example **SOMValidateRequest – use in workflow**

```
<Process-Node>
  <Name>SOMValidateRequest</Name>
  <Action>
     <Class-Name>
     com.hp.ov.activator.mwfm.component.builtin.som.SOMValidateRequest
     </Class-Name>
     <Param name="product_name" value="prodName"/>
     <Param name="product_specification_var" value="prodSpec"/>
     <Param name="product_version" value="prodVer"/>
     <Param name="request_type" value="constant:activate"/>
     <Param name="user_interaction" value="manual"/>
  </Action>
</Process-Node>
```

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 50 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.

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**

```
<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="true">
  <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.