HP Service Activator Test & Diagnostic Product

User's and System Integrator's Guide

Version: 7.0.0

For

Red Hat Enterprise Linux 6.5 operating system

Manufacturing Part Number: None

October 27, 2015

Legal Notices

Warranty.

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

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

Restricted Rights Legend.

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

Hewlett-Packard Company
United States of America

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

Copyright Notices.

Trademark Notices.

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

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

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

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

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

EnterpriseDB® is a registered trademark of EnterpriseDB.

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

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

UNIX® is a registered trademark of the Open Group.

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

Document id: HPSA TD-pd000202

Install Location Descriptors

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

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

Conventions

The following typographical conventions are used in this guide.

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

In This Guide

This guide provides a general description of the HP Service Activator Test & Diagnostic Product (HPSA T&D) as well as the detailed information needed

- to define test & diagnostic actions specific to a CSP services and products and to maintain the definitions

and

- to interact with the HPSA T&D solution to execute test & diagnostics actions trouble-shooting internal network problems as well as serving subscriber calls for assistance when their products or services present problems and difficulties

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

- **Introduction to HP Service Activator Test & Diagnostic Product:** This chapter contains a general description of HPSA T&D from a functional 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 how to use the product.

- **Implementation Architecture:** This chapter describes how HPSA T&D is implemented on the combined platforms of HP Service Activator, the HP UCA Automation (UCA) as well as HP Unified OSS Console (UOC).

- **T&D Installation:** This chapter contains information about how to install and configure HPSA T&D product. Description of how to localize HPSA T&D, setting up a clustered environment and an introduction to the example solution are also included.

- **T&D Solution Configuration:** This describes the configuration of a HPSA T&D Solution adapting it to a CSP's problem management scenarios and environment. . It is relevant for the system integrator who will implement a specific Test & Diagnostic solution, and for the technical product manager who will maintain it.

- **T&D Solution Delivery:** This chapter aim at the system integrator and focuses on the delivery and deployment process of a HPSA T&D solution in a specific CSP environment. The T&D Example scenario is used throughout to explain and exemplify different aspects.

- **Test & Diagnostic Actions:** This chapter, aimed at system integrators, explains how to implement process functionality for Test & Diagnostic actions in the form of HP Service Activator workflows. The T&D Example scenario is used throughout to explain and exemplify different aspects.

- **T&D REST Interface:** This appendix provides details on northbound API of the HPSA T&D Platform.

Audience

The audience for this guide includes all groups who need information about the HPSA T&D:

- CSP solution architects who will evaluate the product

- Systems integrators who will plan and deliver test & diagnostic solutions, including architects as well as developers of test & diagnostic actions and adaptors for external system integration.
- CSP technical product managers who will maintain specific test & diagnostic processes and external system integration components
- CSP operators who will work with the HPSA T&D at runtime: handling subscribers support calls and execute the T&D processes to effectively reach a conclusion on the reported problems.
- CSP operators who will work with the HPSA T&D at runtime: using the actions of the T&D framework for trouble shooting internal network problems, monitor the solution and interact with running T&D requests

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 UCA Automation 1.1 Install Guide*
- *HP UOC V2.1.3 – Installation Guide*

Acronyms

Acronym	**Expansion**
CPE	Customer Premises Equipment
CSP	Communication Service Provider
CSS	Cascading Style Sheet
HPSA	HP Service Activator
json	JavaScript Object Notation
MVC	Model View Controller

NOC	Network Operations Center
OSS	Operation Support System
REST	The “Representational State Transfer” interface
SuT	Service under Test
TT	Trouble Ticket
UCA	HP UCA Automation
UOC	HP Unified OSS Console

1 Introduction to HPSA T&D

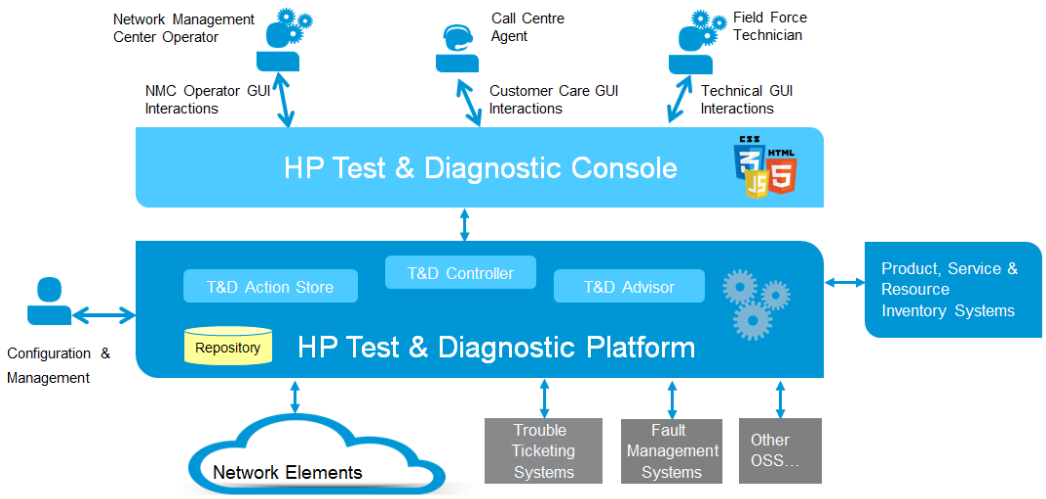
The HP Service Activator Test & Diagnostic Product (HPSA T&D) is a test and problem management solution that supports the CSP in standardizing, automating and optimizing test and diagnostic processes to ensure repeatability and consistency, to provide faster resolution and to assure that the products ordered and provisioned are functioning as desired.

HPSA T&D focus on solving problems the CSP may experience in their customer care, within their network infrastructure management and assuring customer SLAs. HPSA T&D improves the efficiency of problem management and improves the customer's experience.

Some of the main issues in problem management, which the HPSA T&D solve, are:

- Ever developing service offerings and changing infrastructure require configurable tools that can be adapted easily to the dynamic environment.
- Increasingly complex services with complex dependencies require advanced automation tools to validate and diagnose correctly.
- Service down-time is generally unacceptable and requires expedient and correct diagnosis and repair in a customer-centric world.
- The ability to validate services becomes increasingly important with increasing focus on security and fraud management.

Figure 1 **HP Service Activator Test & Diagnostic Product Architecture**



The HPSA T&D solution architecture is illustrated in Figure 1. It consists of the following two main components:

- The HP Service Activator Test & Diagnostic Console (HPSA T&D Console).
- The HP Service Activator Test & Diagnostic Platform (HPSA T&D Platform).

The HPSA T&D Console is the point of interaction for human operators (NOC operators, call agents, field force …). The HPSA T&D Console is a user-friendly, intuitive and customizable GUI for specific operator roles that provides all the facilities (e.g. search …) that allows the operator to view relevant subscriber, service and resource information details related to the case and guides the operator to determine and select the proper T&D Actions and request their execution.as well as authentication and Single-sign-on facilities. The HPSA T&D Console implementation is using modern Web 2.0 technologies.

The HPSA T&D Console interacts with the HPSA T&D Platform for execution of T&D Actions that e.g. executes tests, queries information, or implements other relevant parts of the diagnostic processes.

The HPSA T&D Platform is the process and information back-end of the solution where all T&D Actions are defined and executed and where the integration points with external OSS systems, network elements and/or dedicated test resources are implemented. Hence, the HPSA T&D Console is independent of the detailed system integration points and the details of the T&D Actions.

The T&D Actions implement the product/service specific steps required to diagnose a problem, to validate a product/service, etc. and these may be requested by the HPSA T&D Console or by other external systems.

The HPSA T&D Platform is based on the proven HP Service Activator (HPSA) modular framework, which provides multi-vendor support, high-availability, high-performance and customization flexibility.

HPSA T&D is a framework that provides capabilities that supports both development and deployment of a broad range of test & diagnostic solutions:

- It applies to call center management functions that serves subscriber's support calls with the intuitive and easy to use call agent interface that minimizes the required training time and optimizes the time to resolution of subscriber problems.
- It supports service validation operations that pro-actively verifies correct behavior of newly purchased services before assigning it to the subscriber for usage thereby avoiding negative impact of customer experience.
- It supports integration with OSS and BSS systems that may benefit from the available T&D actions and implemented features.

The common capability lies in orchestrating test and diagnostic sequences and performing actions on the systems and services under test. The client using the T&D capabilities may be different from use case to use case. The specific behavior is visible in different business logic as well as specific user interface capabilities.

The following list some typical use-cases:

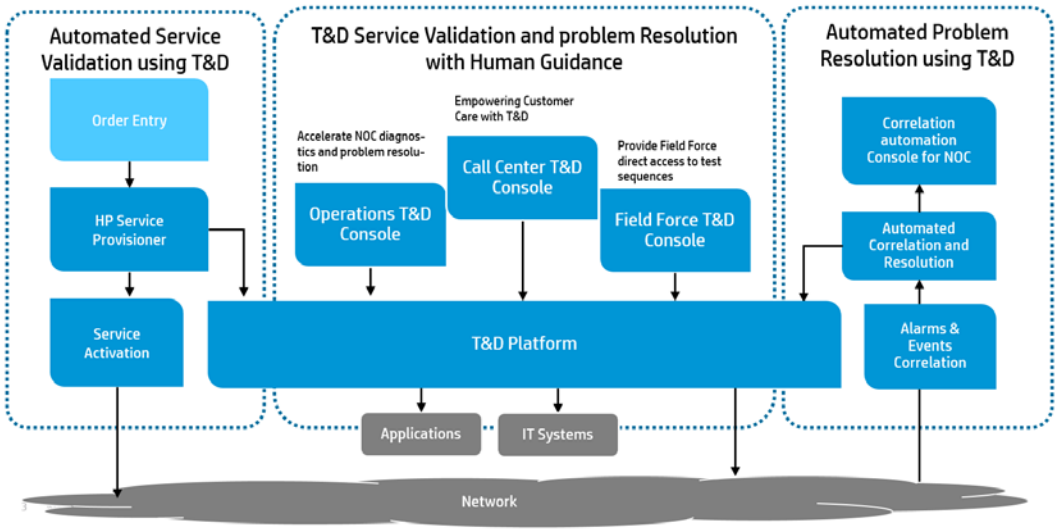
- T&D with Human Guidance
    1. Call Center Management
    2. Self-service T&D
    3. T&D for Field Technicians
    4. T&D for NOC operators

- T&D for Service Validation
    1. Automated Fulfillment Service Test
    2. Operator assisted Fulfillment Service Test
    3. Service Configuration validation
    4. T&D for Revenue Assurance/Fraud detection
- T&D for Service Assurance
    1. Automated problem diagnosis
    2. Automated problem resolution

A schematic view of these different use-cases of HPSA T&D as part of HP OSS Fulfillment and HP OSS Assurance is show in Figure 2.

The main use-case is T&D with Human Guidance illustrated in the middle section of Figure 2. It covers different operator roles and scenarios including Call Center Agents, NOC operators as well as external Field Force technicians.

Figure 2 **HPSA T&D Use-cases and Positioning in HP OSS Fulfillment/Assurance**



In these use cases, the operator is the decision point in the T&D resolution process. The operator decides the next step based on the information successively gained by executing T&D Actions, evaluating the information provided by the T&D Actions and on the guidance provided by the T&D Advisor (see Figure 1).

The T&D Advisor uses a state-based approach to provide configurable guidance to operators. This helps the operators to proceed fast and safely through complex diagnostic processes.

The guidance consists of configured lists of the recommended action and possible actions known to most effectively lead to a resolution of the problem and which the operator can select among in a particular state of the process.

Even other kind of information can be configured as operator guidance, e.g. internal help texts that could inform the operator on what to ask a subscriber about at this point in the process, or what to be cautious about not to break the dialog with the subscriber, etc.

The T&D Advisor allows the operator to stay in control at the decision points in the processes and make the final selection among the probably few proposed actions. Hence, an operator's experience and intuition may be used to its full advantage, at the same time that less experienced operators are helped to select the best next steps that are known to represent the shortest path to resolution.

Being a framework, HPSA T&D does not contain the complete solution for any specific test & diagnostic scenario. It contains the tools to ease the development and maintenance of such solutions, to implement the required T&D Actions and GUI interaction points and it contains the runtime engine for execution of the solution.

In terms of its technical implementation HPSA T&D is built on top of HP Service Activator software extended with the HP UCA Automation platform as the runtime engine. Additionally, the T&D Console providing the operational GUI is a set of components for the HP Unified OSS Console.

The process to build and maintain a specific HPSA T&D solutions is a main topic of this document.

2 Implementation Architecture

This chapter provides and architectural overview of the HPSA T&D framework and introduces concepts that are important to understand in order to use HPSA T&D as intended.

It contains a description of how T&D processes are supported by HPSA T&D and introduces some of the innovative features and details of HPSA T&D that are allowing T&D processes to be adapted to any CSP's test and diagnostics problem management requirements.

HPSA T&D supports state-based configuration of the T&D process in contrast to a more traditional programmatic approach. This allows CSPs to change or extend their evolving T&D processes on their own, when that is desired.

HPSA T&D includes the T&D Advisor that uses the state-based approach to provide configurable guidance to human operators. This helps the operators to proceed fast and safely through complex diagnostic processes.

HPSA T&D Console

The operator facing graphical user interface is implemented using the HP Unified OSS Console framework and provides a role based operational interface for human operators (see Figure 1 and Figure 3).

The HP Unified OSS Console (UOC) solution is a new generation data visualization platform that provides a flexible and open user interface compliant with modern web standards. It is specialized for Operation Support Systems (OSS) and is a generic web framework that facilitates the integration of various OSS software systems by supporting extensions (pluggable modules) to add features on the web browser (client) or to integrate new or existing domain services.

It provides a flexible, modern, responsive, dynamic and open user interface compliant with modern web standards that are able to run on any devices (tablet, phone, desktop…).

The UOC consist of server and client parts. Some of the technologies used in the client parts consist of AnguarJS, HTML5, and CCS3. The server parts which must be installed on a Linux server consist of NodeJS and Express web server application frameworks.

The HPSA T&D Console is implemented as an UOC value pack that provides a T&D specific workspace.

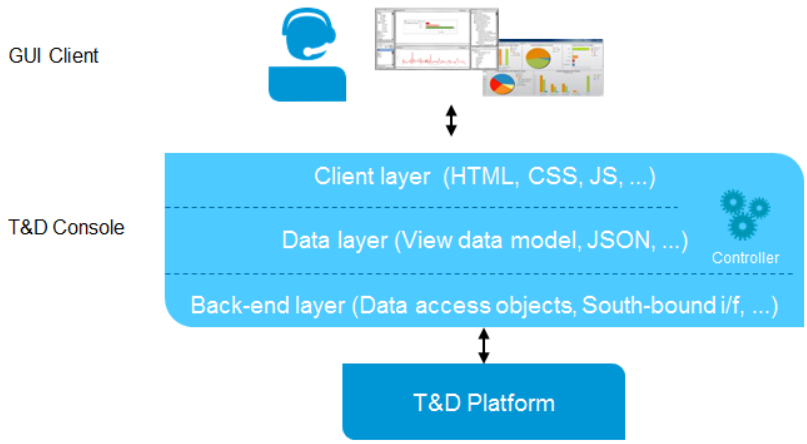
The HPSA T&D Console architecture consists of 3 layers (see Figure 3):

- The Client Layer
  The top layer consists of the GUI representations, controls and widgets used and the look-and-feel of the GUI. This layer implements the View part of the MVC pattern and it uses client side (e.g. JavaScript) and service-side technologies to provide the required items.

- The Data Layer
  This layer implements the Model part of the MVC pattern and application and uses client side (e.g. JSON objects) as well as service-side data objects to populated the widgets, controls and other GUI items.

- The Back-end Layer
  This layer provides a Data layer interface that provides general data access and store functionalities. It hides the specific structure of the T&D Platform data and how the data is obtained via the T&D platform API.

Figure 3 **High-level HPSA T&D Console Architecture**



The HPSA T&D Console provides login/client authorization functionalities, where the role of a specific client is determined. This is implemented using the HPSA T&D Platform based authentication and roles features.

The HPSA T&D Console allows T&D Session operations that e.g. may define of a new T&D Session or allow the resumption of a previously discontinued T&D Session.

The HPSA T&D Console provides the easy to use interaction points that allow the manual selection of the proper T&D Actions and request their execution by the HPSA T&D Platform, using the guidance data provided by the T&D Advisor functions.

All HPSA T&D Console functional data, which e.g. allow selection of the proper T&D Actions or query of subscriber service and resource information, etc., is provided by the T&D Platform. The T&D Console does not by itself keep state or persistent data related to the T&D Session.

The T&D Session is updated by the HPSA T&D Platform with the current state of the T&D Process and the results of the tests and other data relevant for the current state. Additionally, this information is saved (persisted in the repository) by the HPSA T&D Platform so it is available as historical data for later review and reporting.

HPSA T&D Platform

The HPSA T&D Platform uses HPSA to provide a framework where a CSP's business processes are supported, and the T&D Advisor provides guidance to the operator to complete the processes as effective as possible and where a CSP's specific problems and test functions may be automated and implemented by sets of T&D Actions.

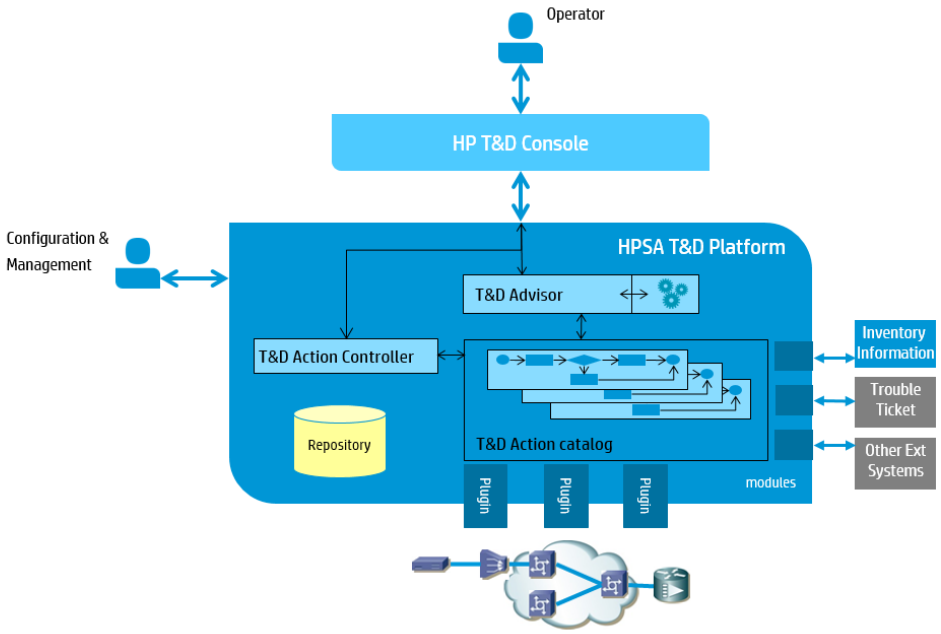
HPSA is a system designed to define, organize and automate configuration and activation tasks on network elements, EMSs and OSS systems and application servers, necessary to manage operational services, including the problem management and service validation test processes.

HPSA is an open system architecture that allows:

- Easy integration of new external components through the plug-in and re-usable module architecture.
- Easy definition of re-usable processes and automation functions through the Workflow components.
- Fast addition of system specific integration with provisioning systems and other external BSS/OSS.
- Easy definition of grouped tasks (e.g. tasks composed of multiple basic or atomic tasks).
- Transactional processing which includes Revert changes ("rollback") in the case of unsuccessful interactions/changes.

The HPSA T&D Platform implementation is based on the HP UCA Automation product and is reusing some of the assets and investments already made in HP UCA Automation. The HP UCA Automation product consists of several components of which it is the “UCA HPSA Foundation Value Pack” that is reused and extended by the HPSA T&D Platform. The action request/response interface of UCA is maintained allowing existing UCA Automation based solutions to integrate with the HPSA T&D Platform as the T&D execution platform and reusing created T&D Actions.

Figure 4 **High-level HPSA T&D Platform Architecture**



The HPSA T&D Platform maintains a catalog of T&D actions which may be selected (invocated) from the HPSA T&D Console or from other north-bound systems.

T&D Actions implement specific functions for the CSP’s problem management, validation and possibly repair of subscriber services. Below are some example functions that may be implemented by T&D Action:

- Execution of Service Validation/Audit Actions
- Execution of Tests Actions to identify problems: Fast tests can be performed to check basic problems
- Redirection of problems: If the problem is not solved immediately, a TT can e.g. be escalated to a higher/another level
- Execution of T&D Actions may be requested be an Operator or a remote system

The HPSA T&D Platform provides access to information on external systems that may be required by the HPSA T&D Console, by the T&D Actions and/or by the T&D processes. The details of how to get/update information on a specific system is confined in the HPSA T&D Platform and the specific details of the external systems and their integration details are hidden from the HPSA T&D Console and other north-bound systems.

The following summarizes the main functions of the HPSA T&D Platform:

- Associates a T&D Session with a specific T&D process, where the current client role, process state and results, historical data, etc. is persisted and available for e.g. resuming a T&D Session at its previous state.
- Provides the T&D Action request interface for the HPSA T&D Console and other external system that may need to request T&D Action execution.
- Provides a single, common integration point towards Network Elements and Test systems as well as external OSS/BSS systems relevant for the T&D Actions.
- Definition of T&D Actions and their required input/output parameters. This include grouping and reuse of existing Actions as building blocks to produce more advanced, or complete or otherwise different T&D Actions.
- Executes the T&D Actions associated specific Problems and Service Types as requested by the HPSA T&D Console and/or other external systems.
- Provides configurable Operator guidance of the T&D end-to-end processes via the T&D Advisor.
- Provides centralized T&D execution logging and management facilities of T&D historical data.
- Provides a GUI interface for the configuration and management of Problems, Symptoms, associated Service types and the T&D Advisor and its guidance features.
- A repository for maintaining persistent data related to T&D Action definitions and their associated Roles, Problems and Service types, to configuration data and look-up table related to the T&D Advisor, and for maintaining T&D Session data and historical data.

HPSA T&D Console and T&D Platform Interface

The HPSA T&D Platform provides a REST interface which north-bound systems like the HPSA T&D Console uses to request execution of T&D Actions and to query for information.

A REST interface may be considered as a lightweight Web Services interface using the HTTP protocol elements for the four CRUD (Create/Read/Update/Delete) operations and using the URLs

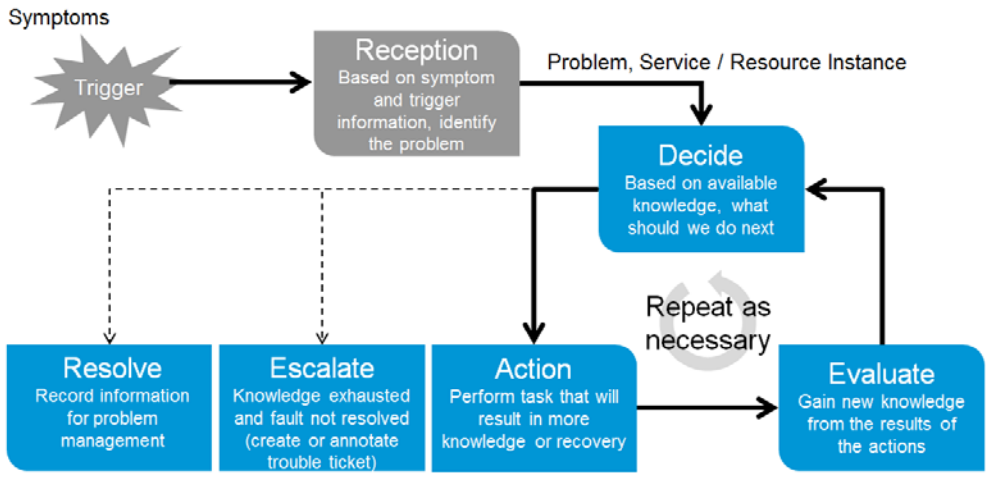
to represents the objects or resources of the request. REST is considered being simple and easy for new clients/north-bound systems to integrate with.

The REST interface is described in more details in appendix A-1 North-bund ReST API.

HPSA T&D Process

HPSA T&D allows a CSP to implement the resolution process following the general steps illustrated below.

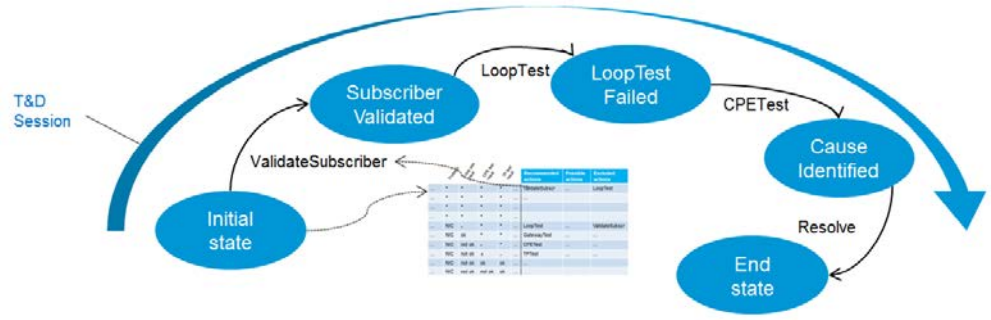
Figure 5 **General T&D process**



The diagnostic process is a considered a repeated execution of Decide, Action and Evaluation steps. In the human guided use cases control of this loop is performed by the operator where the decision step is based on the information successively gained by executing T&D Actions, evaluating the information responded by the T&D Actions and guided by the T&D Advisor recommendations.

The T&D Solution does not implement full end-to-end process as a complete programmatic flow but supports configuring it using a state based look-up mechanism which is used by the T&D Advisor to provide the best next action to execute.

Figure 6 State based T&D Advisor guided end-to-end process



This provides a greater level of configurability of the required T&D Processes and easier modification and adaptation of existing processes to new and changing requirements as the processes are constructed using the T&D Actions as building blocks.

The end-to-end T&D process, its status and results obtained so far is maintained as a T&D Session which allows suspension and resumption of T&D Session at a later time as well as dispatching T&D Session to other operators if so desired.

The State-based Approach

To understand the state-based approach supported by HPSA T&D a small number of concepts need to be explained:

- State-item
- State-vector
- State-index

State-item

A diagnostic process, proceeds in steps from some initial conditions related to e.g. a reported problem, through some incremental steps where the problem is undergoing some tests, towards some final steps where the problem is e.g. resolved or escalated as Figure 3 illustrated.

Each step in this T&D process is described by an associated state that includes the problem and the diagnostic information achieved so far by executed T&D Actions, etc. The state consists of all the information relevant for the problem's T&D process.

To manage this total state, it is divided into individual state-items, each representing a relevant, specific and independent aspect of the diagnostic process. The total state consists of the collection of these state-items.

To make this somewhat abstract notation of the state and its state-items a little more concrete, look at the following common diagnostic problem: My car engine cannot start.

The diagnostic process will likely involve checks of the starter motor, the fuel level and the ignition system. The problem and each of these check points are state-items. Together they represent the relevant state of the diagnostic process. A car consists of many other components not relevant for this "Engine not starting" problem, e.g. the braking system.

Initially in the diagnostic process of this example, the state consists only of the 'problem' state-item ("Engine not starting").

After having e.g. executed the “Check starter motor” the ‘starter motor’ state-item may be OK and the state now consists of the ‘problem’ and the ‘starter motor OK’.

For a little more advanced vehicle diagnostic system, the situation is more complex. E.g. the initial state will also consists of other aspects than the ‘problem’: E.g. the ‘product type’ (Car, Truck, Lawn-cutter, …), its ‘technology’ (Petrol, Diesel, Electric, …) , etc. each with obvious consequences for which types of problems are prevailing, which state-items are relevant and which tests makes sense.

In the real CSP scenarios HPSA T&D is addressing, the number of problems, technology types, and relevant state-items are more numerous and varied than this simple example includes, but the principles are exactly the same.

The state-items are objects defined and configured in HPSA T&D. A state-item represents a key aspect of the diagnostic process. The state-items are defined and configured according to the need and nature of the problems to diagnose, the type of the services and the type and nature of the CSP’s infrastructure, etc.

The state-items maintains persistent data for the T&D process, making e.g. results of previous test available for later steps in the diagnostic process.

State-vector

The state-vector represents the total state of a diagnostic process. The state-vector is the collection of state-items that are involved in the specific diagnostic process. The state-vector is a dynamic object that initially may consist of only a few state-items (like ‘problem’, ‘product’ ...) and it is extended with new state-items as these are defined by e.g. executing T&D Actions.

The state-vector is the set of aspects or conditions in the diagnostic process that defines the ‘point’ the diagnostic process has reached. Moreover, it defines the best ways to proceed further in the diagnostic process.

Continuing our small “Engine not starting” example, when in the initial state, running the “Check starter motor” test could be configured as the best next step. After having executed this test and it is OK, the state-vector is extended with the result and then the “Check fuel” could be the recommended next step in this new state.

If the starter motor is not working, (test failed), the state-vector could now identify a point where it is not possible to resolve the problem further and it has to be escalated to a garage to handle the problem.

The state-vector is an internal object that is dynamically updated by the executed T&D actions to reflect the current state of the diagnostic process. The next sections describe how it is used to provide the guidance to the operator on which next T&D Actions to execute.

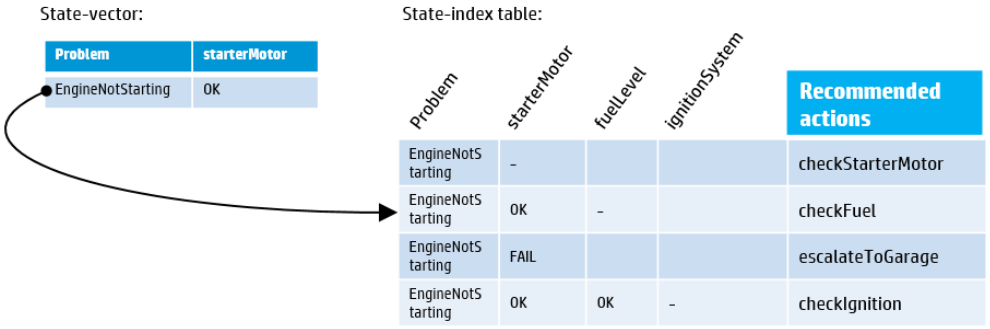
State-index

A state-index is a configured value of the state-vector that is used to point out an entry in the guidance look-up table. The current state-vector value is automatically compared by HPSA T&D with the defined state-indexes. The entry with a state-index matching the current state-vector is used to look-up the guidance to provide to the operator.

The state-indexes, and the guidance is configured into the system according to the need and nature of the problems to diagnose, the type of the services, the type and nature of the CSP’s infra-structure, etc.

Finishing our small “Engine not starting” example, the state-index look-up table could look like shown in Figure 7:

Figure 7 State-vector and State-index look-up table example.



The figure illustrates the current state-vector after having executed the suggested ‘checkStartMotor’ action. The state-vector consists of the ‘problem’ and the ‘starterMotor’ state-items having values ‘EngineNotStarting’ and ‘OK’ respectively.

At this point in the T&D process, the state-vector is matching the indicated entry in the state-index table. This entry provides the guidance in the form of the recommended action ‘checkFuel’.

If the ‘checkStartMotor’ action had failed, the recommended action would have been to escalate the problem to the garage.

In real CSP scenarios HPSA T&D is addressing, the number of problems, technology types, and relevant state-items are more numerous than this simple example includes, but the principles are the same.

The state-based approach supports breaking the overall complex processes into smaller, manageable parts each focusing on independent steps of the diagnostic process.

The state-items and state-index entries associated a specific problem may be configured separately and independently of other problems. The solution is then configured problem by problem to gradually provide the full functionality of a specific HPSA T&D solution.

Operator Guidance

The previous section defined some of the configuration items that must be created and defined in HPSA T&D to model and represent a CSP’s specific T&D processes.

When the state items have been configured, and the required T&D Actions have been defined and implemented, and the state-index lookup table has been configured, the T&D Advisor will provide the configured recommended actions as a selection list to the operator.

In the above small example, only a single recommended action was configured, but in more realistic scenarios, multiple actions may be relevant at a given point in the T&D process.

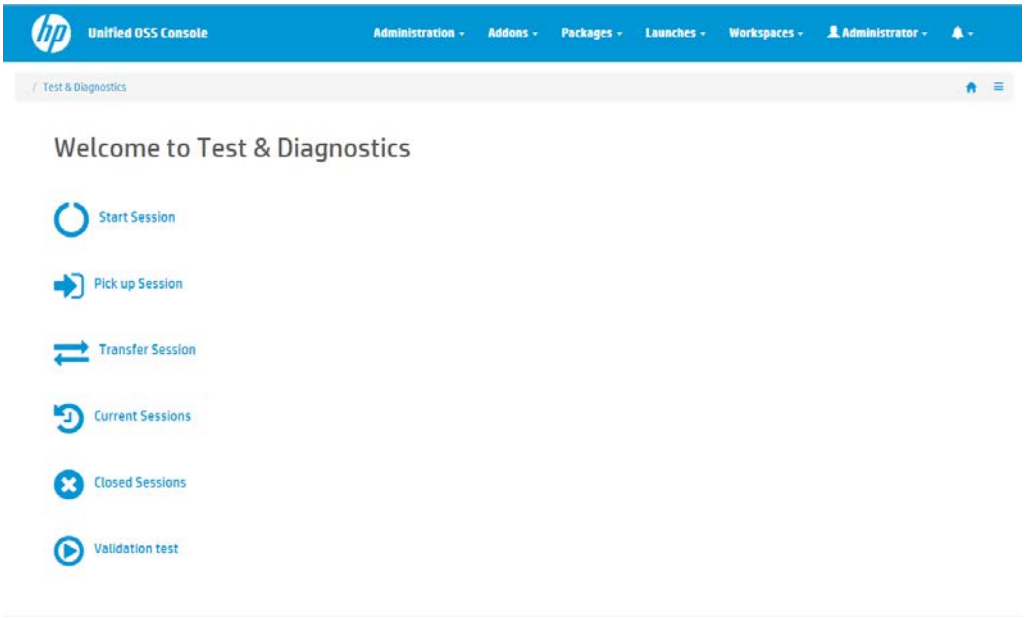
HPSA T&D also allows configuration of ’Possible actions’ which is an extended set of actions an operator could select from, if found relevant. Even other kind of information can be provided as operator guidance, e.g. internal help texts that could inform the operator on what to ask a subscriber about at this point in the process, or what to be cautious about not to break the dialog with the subscriber, etc.

The T&D Advisor allows the operator to stay in control of the decision points defined by the state-indexes and make the final selection among the probably few proposed actions. Hence, an operator's experience and intuition may be used to its full advantage, at the same time that less experienced operators are helped to select the best next steps that are known to represent the shortest path to resolution. For operation and administration of Subscription Repository and Trueview Inventory please refer to the documentation for those products.

The primary user interface for HPSA T&D is implemented by the HPSA T&D Console and is described in Chapter 4 HPSA T&D Solution Configuration.

The main GUI of the T&D Console allows you to manage T&D Sessions and to execute specific Validation tests. The Figure 8 below illustrates the main T&D GUI.

Figure 8 T&D Console Main GUI



The main view presented to the operator is role dependent menu and may e.g. not provide the Validation test entry if the operator doesn't have the Validation role associated.

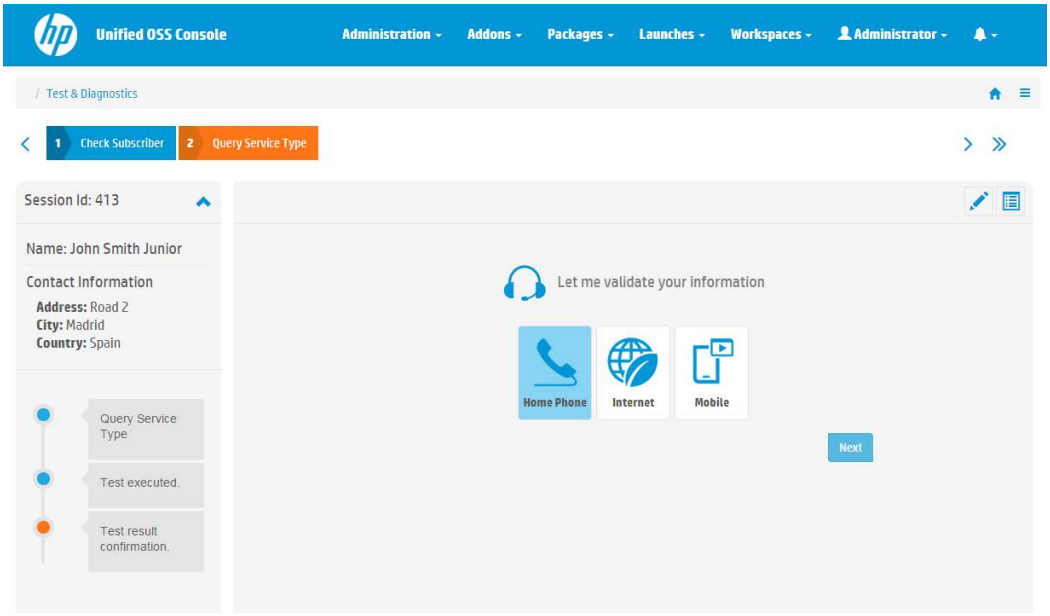
Typically an operator may start a new session (Start Session), search and possibly resume an already active session (Pick up Session/Current Sessions) or dispatch an active session to a new/different operator (Transfer Session).

The operator may also search historical data and inspect the details of these completed sessions (Closed Sessions).

The appearance of an actual T&D session depends on how the state-items and state-indexes are defined and how the display data is configured.

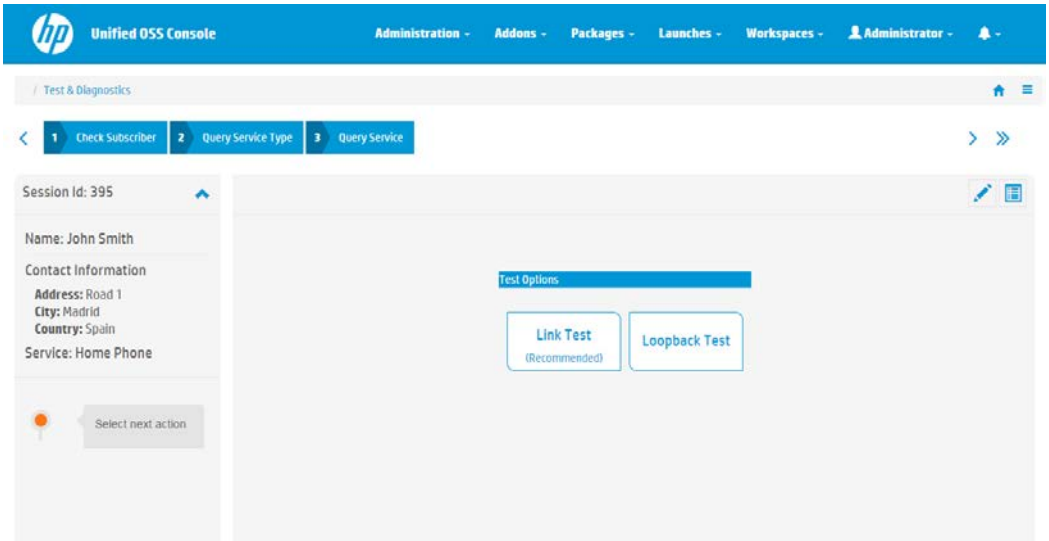
The example in Figure 9 below illustrates a selection form where icons are used to represent the different service types owned by the customer.

Figure 9 **Example showing the use of icons for selection among the customer's service types**



Operator Guidance will appear as a list of selectable actions presented in the GUI. Figure 10 below illustrates an example where there are two test actions available: A "Link Test" is the recommended action and a "Loopback Test" is an optional possible action that could also be selected by the operator.

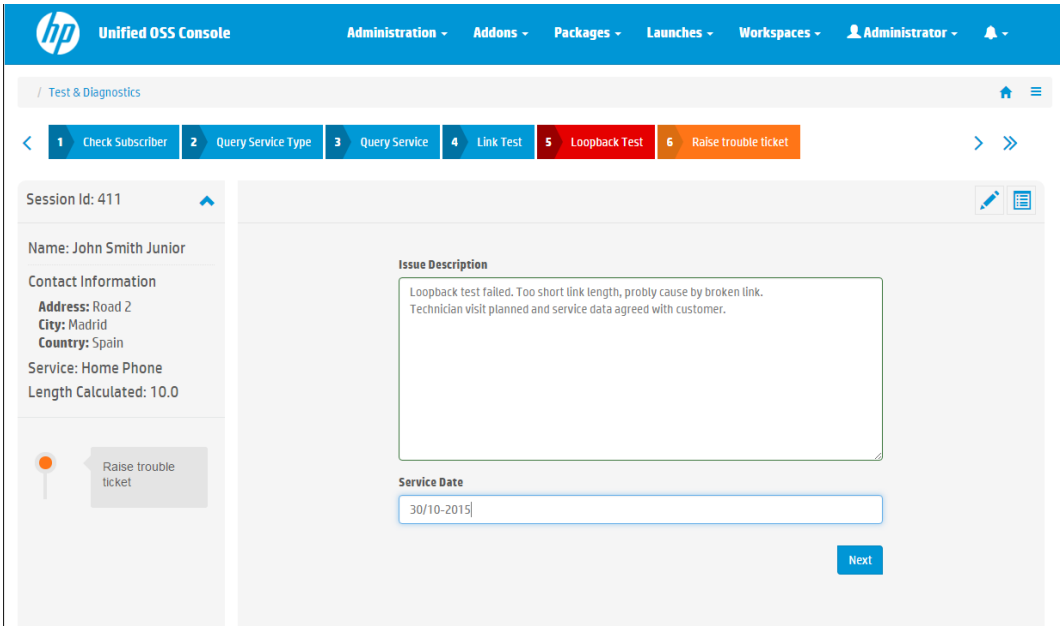
Figure 10 **Example operator Guidance provided as a recommended test and an optional test.**



In the figure above, the progress of the T&D Session is provided as the horizontal list of blue numbered steps: 1: Check Subscriber, 2: Query Service Type, 3: Query Service. This bar also allows the operator to re-run some earlier steps or resume the process at an earlier point of the session.

A case of a failed test action is illustrated in Figure 11 below. Here the progress bar represents the failed step by a red colored step: 5: Loopback Test. Similarly an orange colored step: 6: Raise Trouble Ticket represent the currently active action, in this case requesting the operator to provide input to the TT form.

Figure 11 Example of a failed test case which triggers raising a trouble ticket.

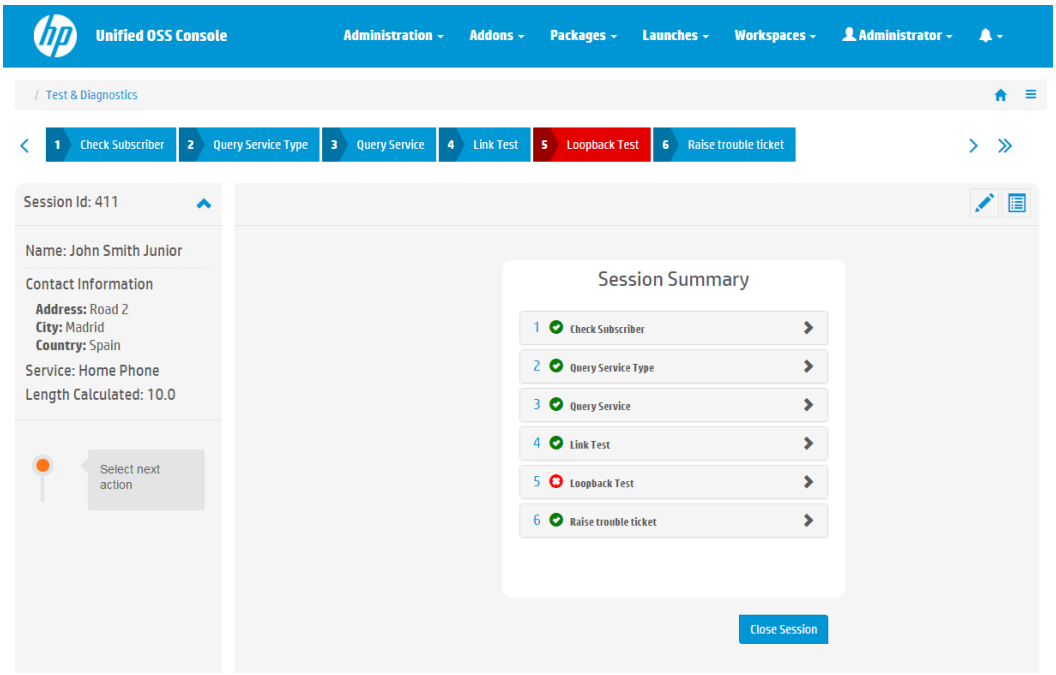


Depending upon the specific test cases and results, the diagnostic process may require more or fewer actions executed before a conclusion can be reached.

When a T&D Session concludes, a Session Summary that includes each step (action) performed and its main input and output parameters is presented. These data are also persisted as historical data and may be later searched and inspected via the Closed Session menu. Figure 12 below illustrates an example of such a Session Summary view.

More GUI examples and the T&D Console configuration guide is provided in Chapter 4 HPSA T&D Solution Configuration.

Figure 12 Example Session Summary view

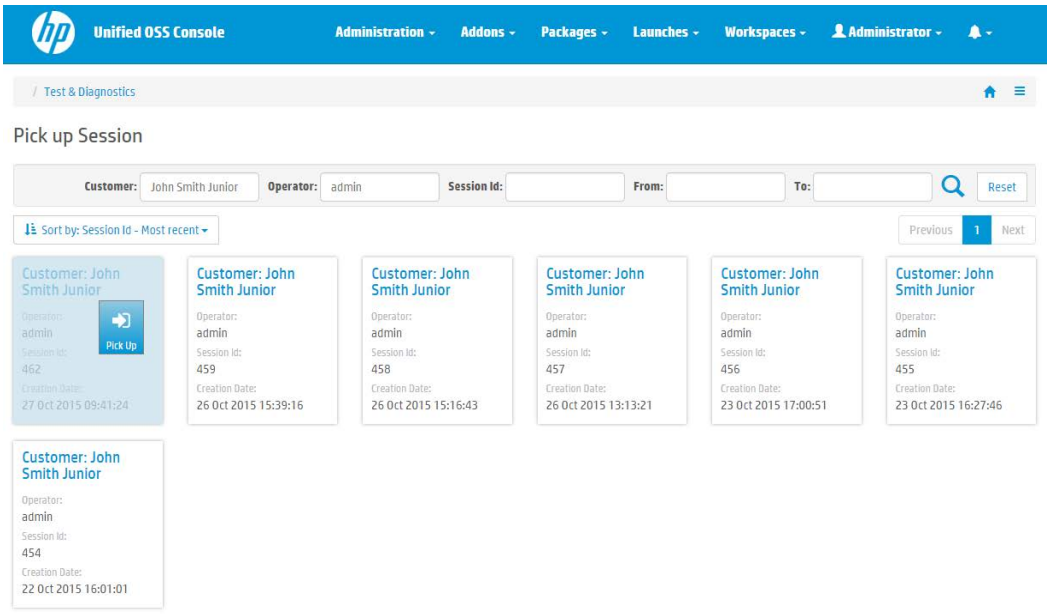


General structure of the T&D Console GUI

The previous example demonstrate some of the features of the T&D Console GUIs that are provided out-of-the-box with the HPSA T&D product. Tis section completes the description of the general features and controls available in the GUI.

The main GUI menu is already illustrated in Figure 8 above. Most of the T&D Session related entries, enter into a search view, e.g. illustrated by the “Pick Up Session” menu in Figure 13 below.

Figure 13 T&D Session search menu illustrated by Session Pick Up menu.



The page allows you to search using one or more of the following information: Customer, Operator, Session Id and Date range as filter.

Hovering the mouse over one of the returned session provide in this case a Pick UP option that allows you to proceed with this session if your associated role allows that.

In the top right corner of the main GUI, a “home” button and a “session menu” button are available to allow fast return or selection of another main menu entry.

The other T&D Session related menu a structured similarly.

When a Session is ongoing, Figure 9 illustrates a typical page view. Some of the features available are:

- T&D Session Progress bar.

    1. Execution history of already executed T&D Actions and the current active T&D Action
    2. Scroll options to navigate back and forth in the T&D Session execution history
    3. A Re-run option on previously executed Actions

- T&D Action execution status.

    4. Current Action Status

- T&D Session Sticky data display field.

    5. Customer data display
    6. T&D Sticky Data collapse and Expand controls

- T&D Action Main view top bar

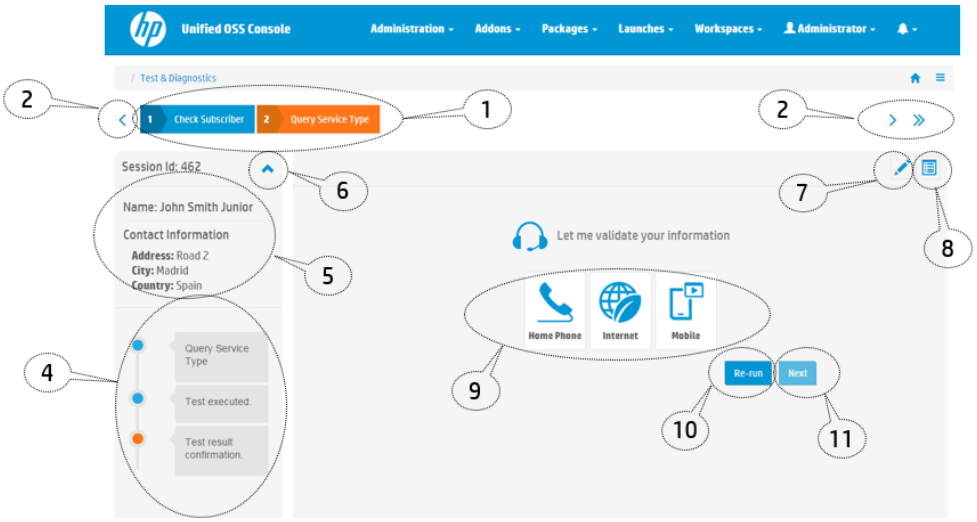
    7. Add/edit Notes option to T&D Session
    8. Historic Data Display option

- T&D Action main view.

    9. Display of input/output parameters of current T&D Action
    10. Re-run option of current T&D Action
    11. Next action to execute control
        If all input parameters are mandatory, the next button will be disabled until the fields have been provided

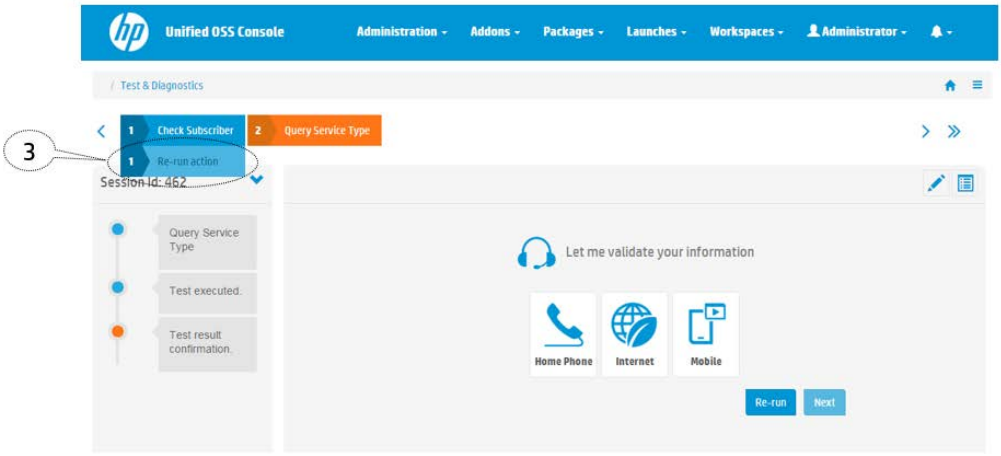
Figure 14 below identifies these features and control using the same numbers as above.

Figure 14 **Main T&D Console GUI page and common controls. Numbers refer to bullet list above**



The Re-run control of the T&D Session progress bar described as bullet 3 above appears only when you hover the associated T&D Action icon the progress bar. This is control is shown in Figure 15 below.

Figure 15 **Main T&D Console GUI page and the Re-run control in T&D Session Progress bar. Numbers refer to bullet list above**



3 HPSA T&D Installation

This chapter guides you through the steps required to install HPSA T&D. At a glance, installation consists of the following steps:

- Install HP Service Activator
- Deploy HP UCA Automation's HPSA Foundation Value pack using the HP Service Activator Deployment Manager
- Install HP Unified OSS Console
- Deploy the HPSA T&D Product using the HP Service Activator Deployment Manager
- Configure HPSA T&D Product components

For additional information about installing HP Service Activator, HP UCA Automation and HP Unified OSS Console, please refer to their respective documentation.

The distribution media of the HPSA T&D Product contains the following required packages:

- HP UCA Automation's HPSA Foundation Value pack
  This is the file: `/Binaries/TD/UCA_HPSA_FoundationVP-V20-1A.zip`
- HP Unified OSS Console
  This is the file: `/Binaries/UoC/UOCV2.1.3-MP.tar`
- HPSA T&D Product
  This is the file: `/Binaries/TD/TD.zip`

Install HP Service Activator

Follow the installation guide provided by the HP Service Activator distribution to install HPSA.

Install HP UCA Automation HPSA Foundation Value pack

If the HP UCA Automation HPSA Foundation solution is already installed/in-use, you may skip this step. Otherwise, the UCA HPSA Foundation solution is required by HPSA T&D and must be deployed before deployment of HPSA T&D:

- Use the utility: *`$ACTIVATOR_BIN`*`/deploymentmanager`
- Import `/Binaries/TD/UCA_HPSA_FoundationVP-V20-1A.zip`
- Deploy the local solution `UCA` (remember to check '`Create inventory tables`')

- Restart HPSA

Install HP Unified OSS Console

Follow the installation guide provided by the UOC product distribution (the uoc2_kit/README file provides a quick guide). This consists basically of the following steps:

- Untar the UOCV2.1.3-MP.tar package into e.g. `/tmp`
- Create users `uoc` and `couchdb`
- Execute as user root: **`/tmp/uoc2_kit/install.sh –s`**
- Execute as user root: **`/opt/uoc2/scripts/setup.sh`**
- Update user `uoc's` environment: `/home/uoc/.bash_profile`

Deploy HPSA T&D

The HPSA T&D product software is packaged as a zipped HP Service Activator solution pack named `TD.zip.`

The solution pack can be imported and 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*).

Follow these steps to deploy HPSA T&D product:

- 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 `/Binaries/TD/TD.zip`, and click the `[Import]` button.
- Click the "Deploy Local Solution" menu item, select the solution named "TD", and click the `[Deploy solution]` button.

    1. Two deployments modes are supported related to the two deployment descriptors available:

        - `deploy.xml`
          This is the standard deployment descriptor to be used in a normal scenario with a co-located OSS Console product (UOC).
        - deploy_no_uoc.xml
          This deploys the T&D backend only and can e.g. be used if the OSS Console is located on another server.

    2. Remember to check '`Create inventory tables`' (for the first cluster node installation in a cluster).

Configuring HPSA T&D

Before the HPSA T&D software can be used and before you may deploy and use a specific T&D Solution, you need to perform some configurations as described in this section.

AdvisorModule Configuration

NOTE — Configuration of the `AdvisorModule` is mandatory for using the user configurable state based operator guidance.

The configuration file that must be updated is `$ACTIVATOR_ETC/config/mwfm.xml.`

The `AdvisorModule` must be configured as:

```
<Module>
    <Name>advisor_module</Name>
    <Class-Name>
       com.hp.td.advisor.mwfm.engine.module.AdvisorModule
    </Class-Name>
    <Param name="database_module" value="db"/>
</Module>
```

Refer to HP Service Activator, Workflows and the Workflow Manager documentation for details of how to configure and update modules.

TD_log_manager Configuration

HPSA T&D requires that a T&D specific Log module is configured. This is used for collecting historical data.

The configuration file that must be updated is `$ACTIVATOR_ETC/config/mwfm.xml.`

The `SolutionXMLLogModule` must be configured as:

```
<Module>
    <Name>TD_log_manager</Name>
    <Class-Name>
      com.hp.ov.activator.mwfm.engine.module.SolutionXMLLogModule
    </Class-Name>
    <Param name="log_level" value="INFORMATIVE"/>
    <Param name="log_max_entries" value="1000"/>
    <Param name="log_allow_statistics" value="false"/>
    <Param name="log_directory"
           value="/var/opt/OV/ServiceActivator/log"/>
    <Param name="solution_name" value="TD"/>
    <Param name="index" value="false" />
</Module>
```

Refer to HP Service Activator, Workflows and the Workflow Manager for details of how to enable and configure this module.

IMPORTANT — The name must equal **`TD_log_manager`** and the parameter `solution_name` must equal **`TD`**

Authentication Manager Configuration

To support the authentication of T&D requests and associated roles, you must enable and configure an authentication manager module in $ACTIVATOR_ETC/config/mwfm.xml. HPSA supports 5 different ways. Please see chapter 7 in the HP Service Activator overview document.

In general, LDAP based authentication will be the most practical authentication mechanism as user and their credentials then only need to be defined in a single place and are used for run-time authentication by both the HPSA T&D Console and the HPSA T&D Platform.

If LDAP is not used, the administrator of the system must ensure the same users with same passwords are configured both in the HPSA T&D Console and in the HPSA T&D Platform at the respective places.

If you don't enable any authentication module, the T&D Solution may work but with limited functionalities around dispatching and resuming T&D session as only a single user will effectively exist.

Refer to HP Service Activator, Workflows and the Workflow Manager for details of how to enable and configure an authentication manager, including LDAP authentication.

Refer to section User Authentication and Roles for more specific information on configuring users and roles for a T&D Solution.

OSS Console Configuration

The T&D addon installed on the OSS Console require some adaptation to local HPSA configuration parameters. For this you must follow the below steps:

- **`cd /opt/uoc2/server/addons/plugins/td`**
- Use your favorite editor to edit `config.json`
- Modify/check the following name/value pairs:

  ```
  "sa.host" : "localhost",
  "sa.port" : "8080",
  ```

  to fit your local setup. E.g. the sa.port may have been specified during HPSA configuration to be different from the default 8080!

- Modify/check the following name/value pairs:

  ```
  "sa.rest.username" : "hpsa",
  "sa.rest.password" : "s5/HIIXk9NIZdrwnw6tSmA==",
  ```

  The username/password must be set according to the system user specified during HPSA configuration for the requests issued by the OSS Console towards the T&D Platform to succeed.
  Note, that the encrypted system user password must be generated using the crypt utility available in *`$ACTIVATOR_BIN`*. Use the following syntax:
  `$ACTIVATOR_BIN/crypt -encrypt <text>`
  where `<text>` is the clear text version of the system user's password. The utility will generate the encrypted version that must be entered as the `sa.rest.password` value.

Validation of Installation and Configuration

Having completed to above steps, you may now validate the correct behavior of the T&D product.

- Check HPSA installation

    1. You should start the HPSA service by executing:
       `# service activator start`

    2. You should be able, using a supported Web browser, to log in as the system user on HPSA's login form (the port may be different from 8080 in the URL below):
       `http://localhost:8080/activator/jsp/login.jsp`

    3. Inspect the installed workflows, by selecting the Workflows menu in HPSA's left panel. You should see 5 workflows related to CRModel, 5 workflows related the TD and 5 workflows related to UCA.

- Check UCA installation

    1. Select the inventory GUI by selecting the Inventory menu in HPSA's left panel. You should have 3 UCA instance views (Services, ActionFramework,, Parameters).

- Check OSS Console (UOC) installation

    1. Make sure the couchdb and uoc are started by executing:
       ```
       # sudo /etc/init.d/couchdb start
       # su – uoc
       $ uoc2 start
       ```

    2. Login, using a supported Web browser, to the OSS Console as admin:
       `http://localhost:3300/`

    3. Observe that a Test & Diagnostics menu appears on the GUI. If not, try to execute a restart cycle of UOC and HPSA Service Activator

HPSA T&D Localization

Localizing HPSA T&D 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.

This section describes the resource bundles that must be translated to localize the HP Service Activator Test and Diagnostic product and the corresponding HP Service Activator T&D solution using the product.

In general the Java resource bundles (in English) are files with names ending in `_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, like `_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. This translation must be done for strings defined for the backend and the data load file.

Localization of the T&D backend text strings

The default java resource bundle is found in *$ACTVATOR_OPT/solutions/TD/*etc/nls.

You must create a Java archive named tdnls.jar containing all the translated resource bundle files and save it in *$ACTVATOR_OPT/solutions/TD/*3rd-party/lib. Note this must be done after the import of the solution by the deployment manager but before the deployment of the solution. A tdnls.jar file already exist in the directory containing the default English text strings.

Localization of the T&D User Interface strings

The default English resource bundle is found in *$ACTVATOR_OPT/solutions/TD/*uoc/client/public/local. You can add resource bundles for the locales you need before making the deployment. The configuration here is done in json format as the Unified OSS Console is using json.

Localization of text strings used in the data load file

When configuring the dataload file for your specific solution, you provide a number of help text and labels as part of you configuration. To localize these strings you need to define for each string the default text, the key, and the resource bundle name in the dataload file. The syntax to configure this is as follow:

```
!<bundle name>!<key>!<default string>
```

If the <bundle name> is empty (reducing first token to !!), the default bundle defined in the file will be used. Below you see a help text configuration example causing the default bundle is used:

```
<text>!!delayHelpText!Specify the time delay required</text>
```

When the data loadfile is configured with the desired content, the java resource bundle can be created by using the dataload tool in the following way:

```
$ACTIVATOR_OPT/bin/TDData –generateBundle <data load file>
```

Pass the output of the command to a file that will then contain the properties. This file must then be translated to the appropriate locale in the same way as it is done for the T&D backend text strings above. You may generated multiple version for different locales and then pack these into a single <solution_name>nls.jar file and locate it in your solution's *$ACTVATOR_OPT/solutions/<solution name>*/3rd-party/lib.

Remember to update you deployment descriptor to include this file during deployment.

Clustered Environment

HPSA T&D product may be deployed into a HPSA cluster environment. The HPSA cluster functionality also supports a Disaster Recovery (DR) scenario, where a second cluster is cold standby and awaits an operator initiated start-up in case the primary cluster becomes unavailable.

The T&D product consists of the HPSA T&D Console and the HPSA T&D Platform. The HPSA T&D Console implements the operator GUI as a workspace in UOC. The HPSA T&D Platform is basically a HP Service Activator application using a JBoss application server.

The HPSA T&D Console and the HPSA T&D Platform could be deployed on different server platforms but it is recommend to deploy these two components co-located on the same platform to benefit from the virtual IP address switch-over feature of the HPSA T&D Platform's clustering feature which includes failover in case a server node fails and load balancing among the multiple nodes in a cluster.

Each cluster nodes will be associated a virtual IP address, say VIP1 and VIP2 in a two node scenario. In case of a fail-over, e.g. if node 1 fails, its VIP1 will be added to node 2's set of IP addresses. Hence the GUI display stations will continue operating towards VIP1 but actually use node 2 as the HPSA T&D Platform.

Additionally, the HPSA cluster feature support disaster recovery setup. Such a setup will contain a number of sites where the primary site is normally running. Other sites are called standby sites and multiple standby sites can exist. The standby are not running during normal operation.

Figure 16 Co-located HPSA T&D Console and the HPSA T&D Platform cluster scenario

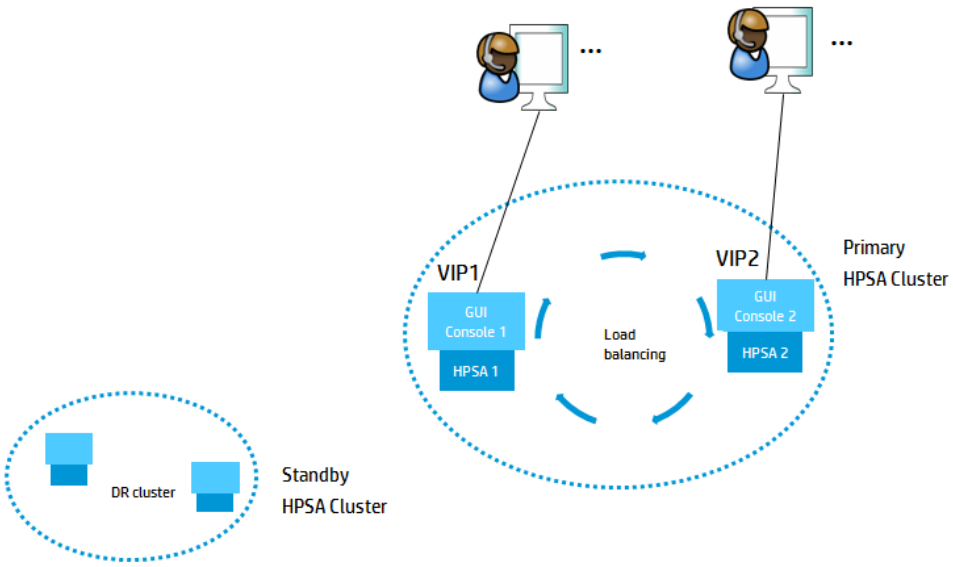


Figure 16 above illustrates a Primary and a Standby Disaster Recovery site. Each of these two sites consists of a two-node cluster configuration with the HPSA T&D Console and the HPSA T&D Platform co-located on the same server platforms. The DR standby site consists of a replication of the servers and their setup of the primary site and will, when started, provide VIP1 and VIP2 for the T&D Console clients to use and connect to.

There are some advantages when deploying the co-located scenario:

- The communication between the HPSA T&D Console and the HPSA T&D Platform will be internal, local to the node, which will provide improved response times.
- In case of failure of a cluster node, its VIP will move to a redundant node and access from the clients to the HPSA T&D Console will also move to the redundant node.

It is assumed, that the agents are distributed more or less statically between the HPSA T&D Console. If e.g. the agents are located in two different service centers to provide 24 hours availability, then half the agents of each site could be assigned the HPSA T&D Console 1 and the other half of each site could be assigned HPSA T&D Console 2. This would distribute the load over the service centers and over time more equally.

In case where the agents are not easily divided into static groups an external SLB could be deployed as a front-end of the HPSA T&D Consoles to provide more equal load distribution among the nodes.

For further information about configuring a Cluster and Disaster Recovery sites, please refer to the relevant HPSA documentation.

Example T&D Solution

NOTE	Deployment of the TD Example solution is optional; it is not a required part of HPSA T&D product.

In addition to the solution zip file containing the HPSA T&D product framework software, an example solution called “TD Ex” is also bundled with the installation kit.

The name of the “TD Ex” solution zip file is `TD_Ex.zip` and it is located in the directory `/Binaries/TD/TD_Ex on your install media`. It can be deployed with the HP Service Activator Deployment Manager using similar steps to the ones described in the Section “Deploying HPSA T&D Solution” on page 29.

The TD Example solution consists of sample workflows, some Customer and Services inventory beans and a XML-formatted configuration file (which can be found in the directory *`$ACTIVATOR_OPT`*`/solutions/TD_Ex/etc/data/dataloadAdvanced.xml`) that may be imported using the Dataload utility, please read the section “HP T&D Dataload tool” on page 59.

If you wish to use or experiment with the example solution, please also read the chapters “HPSA T&D Solution Configuration” and “Solution Delivery” which are based on the example scenarios.

4 HPSA T&D Solution Configuration

This chapter focuses on the configuration of a HPSA T&D Solution adapting it to a CSP's problem management scenarios and environment.

The HPSA T&D Product is a framework for implementing a CSP specific T&D solution. As mentioned in the section Installation, this framework is delivered as a HPSA solution located as a solution zip file that can be deployed with the HP Service Activator Deployment Manager, on your distribution media as `/Binaries/TD/TD.zip/TD.zip`.

An actual CSP specific T&D solution is made as yet another HPSA solution with its own deployment definitions, configuration files, workflow definitions, etc. that also can be deployed with the HP Service Activator Deployment Manager. The location of this CSP specific T&D solution will e.g. be: *$ACTIVATOR_OPT*`/solutions/<TD_name>` where `<TD_name>` is the chosen name for the `CSP's` specific T&D solution.

A HPSA T&D Solution is built from two main components, the HPSA T&D Console and the HPSA T&D Platform and this configuration guide describes the configuration aspects accordingly.

HPSA T&D Console

The HPSA T&D Console is based on the Unified Operations Console (UOC) as already described, and does not require further configuration than described in chapter 3: Installation.

For a specific T&D solution you may need to customize the GUI look-and-feel. This mainly consists of modifying the relevant CSS files and possibly add some font families and/or icons and images.

The customization of the HPSA T&D Console follow the guidelines of the UOC product and typically consists of:

- Creating a specific Theme representing the desired look-and-feel. This can be initiated by copying one of the existing theme directories to a new 'mytheme' directory:

  ```
  /opt/uoc2/client/addons/hp/themes/mytheme
  ```

  and update the theme descriptor file `themes.json`:

  ```
  [{
     "themeId": "mytheme",
     "name": "MyTheme",
     "description": "My Theme here",
     "author": "hp",
     "version": "1.0",
     "icon": "mytheme.png",
     "location": "/addons/hp/themes/mytheme"
  }]
  ```

- Under this directory you find the `bootstrap.css` which is the main target for customizations:
  `/opt/uoc2/client/addons/hp/themes/mytheme/bootstrap-3.2.0/dist/css/bootstrap.css`

- You may add theme specific wallpapers, logos, icons, etc. respectively under:

  `/opt/uoc2/client/public/images/wallpapers/mytheme-wallpaper.png`

  `/opt/uoc2/client/public/images/branding/mytheme-logo.png`

  `/opt/uoc2/client/public/images/themes/mytheme.png`

- Additionally, the T&D Product uses the css file:

  `/var/opt/uoc2/client/public/css/fftd.css`

  (installed from: `/opt/uoc2/client/public/css/fftd.css)`which also may be desired to adjust according to your local preferences.

You are kindly referred to pertinent UOC documentation, Bootstrap and general CSS documentation to further assist you in the GUI customization process.

HPSA T&D Platform

Configuring a specific T&D solution consists of several steps to be executed on the HPSA T&D Platform and which are described in the following sections. The main configuration item is the T&D dataload file that defines your specific T&D solution processes.

HP T&D Dataload Configuration File

The T&D Configuration file is in xml format and is located in the solutions data sub-directory: *$ACTIVATOR_OPT*`/solutions/<TD_name>/etc/data/<dataload>.xml`.

The schema definition file controlling the dataload configuration file structure is available as: *$ACTIVATOR_ETC/*`config/data.xsd.`

This dataload file is structured as 5 lists of the xml elements <group>, <action>, <serviceType>, <workflowTemplate> and <helpText> as illustrated below.

The XML notation used in this chapter, uses ‘usual’ regexp symbols for ease of reading: ‘+’ denotes 1 or more occurrences, ‘*’ denotes zero or more occurrences, ‘?’ denotes 0 or 1 occurrences, ‘|’ denotes choice.

```
<TD xmlns="http://www.xml.td.hp.com/Model">
   <solution/>?
   <groups>
         <group/>*
   </groups>
   <actions>
         <action/>*
```

```
    </actions>
    <serviceTypes>
          <serviceType/>*
    </serviceTypes>
    <workflowTemplates>
          <workflowTemplate/>*
    </workflowTemplates>
    <helpTexts>
          <helpText/>*
    </helpTexts>
</TD>
```

The meaning, structure and usage of these 5 element lists is explained in the sections below.

Groups

A **group** element represents a sub-section of the state-index lookup table (see section: The State-based Approach) that can be defined and managed independently from other sub-sections of the state-index lookup table. A **group** is associated an **order** attribute and a **name** that is used to identify the group.

Multiple groups may be used to segment the state-index table into logically separate sections, each section depending on a sub-set of states. The sections linked together by the **nextGroup** reference (see below) and a group of order N may only use a nextGroup refrence to an order N+1 group.

```
<group order=$nn>
   <name/>
   <states>
      <name/>*
   </states>
   <indexes>
      <index/>+
   </indexes>
</group>
```

The group element defines the list of **states** items **names (**unique among all groups) relevant for the **group**. The order of this list of state names must be the same order that their values must be specified in the **indexes** section (see below).

The **states** items list is followed by the state-**indexes** lookup table for the group. In the state-index lookup table entries, you define the state-vector **index** values to be matched at execution time to the actual value of the state-vector. Each **index** (i.e. state-vector value) is associated either a reference to a **nextGroup**, or up to three lists of actions relevant for this **group** of type **recommended**, **possible** or **excluded** or an **endState** element:

```
<index>+
   <cause/>?
   <states>
      <value/>+
   </states>
```

```
   <actions proceed=”false”>
      <recommended>?
         <action/>*
      </recommended>
      <possible>?
         <action/>*
      </possible>
      <excluded>?
         <action/>*
      </excluded>
   </actions> | <nextGroup/> | <endState/>
</index>
```

The **cause** element is not currently used. For possible future usage.
The sequences of **index states values** must correspond to the sequence of **states names** defined above.

The **states** values may be expressed as explicitly values such as Ok, SubscriberFound or by using the predefined wild-cards: ‘+‘, ‘-’ or ‘*’:

- **‘-’** (minus) matches an unknown value, i.e. a yet unassigned state-item.
- **‘+’** (plus) represents a known values, i.e. any assigned value of the state-item.
- **‘*’** (star) represents any value of the state-item, assigned or not.
- An empty value (e.g. </value>) will behave as a ‘-‘ wildcard.

A small example may better explain this. Consider e.g. the two actions: findSubscriber and pingTest. Assume, the findSubscriber action may return the Subscriber state-item as either ‘Known’ or ‘Unknown’ and the pingTest action may return the state-item Result as either ‘Ok’ or ‘Failed.

We may then define e.g. the following **states** and **indexes** values:

```
...
<states>
   <name>Subscriber</name>
   <name>Result</name>
</states>
<indexes>
   <index>
      <states>
         <value>+</value>
         <value>-</value>
      </states>
   <actions>
   ...
```

In this example the index will be matched if the state-vector item Subscriber has been assigned a value (‘+’) and that the state-item Result has not yet been assigned a value (‘-‘). Hence, that may represent a case when a Subscriber has been looked up (either ‘Known’ or ‘Unknown’) and the execution of the test action that updates Result has not yet been executed.

If the index values are configured as in the next example, the state-vector will match the index entry if the state-vector item Subscriber has been assigned the 'Known' value and that the state-item Result has been assigned the value 'Failed'. Hence, that may represent the case when a Subscriber has been validated successfully but the execution of a test action has failed.

```
...
<index>
   <states>
      <value>Known</value>
      <value>Failed</value>
   </states>
<actions>
...
```

When the dynamically calculated state-vector matches the **states value** of a specific index entry, the **actions** lists of this entry are looked up and provided to the operator as suggested lists of actions that may be requested for execution.

If only a single action is recommended and it requires no input parameters or the input parameters are hidden the operator (see below how to configure that), the recommended action will be requested for execution automatically by the T&D Console. Similarly, if the output parameters are hidden, the next lookup will be initiated without operator interaction. Hence, this can be used to automatically execute a chain of actions.

The **proceed** attribute on the actions list also allows automatic execution of a single suggested action with the slight difference, that it is the T&D Platform that executes the suggested action and not an (automatic) requests from the T&D Console.

When a **nextGroup** is looked up, it provides a linkage into the specified next group to continue the search for a matching index entry.

When an **endState** is looked up, it indicates the termination of the current T&D session.

Each action element in these lists, is a reference to one of the named actions specified in the <**actions**> element below and optionally also including the solution name:

```
<action role="some_role">
   <name/>
   <solution/>?
</action>
```

The role is an optional attribute that may be used to filter the selection and execution of an action to the specified role (see section User Authentication and Roles). An action request is executed as a user (e.g. the operator) and the roles associated this user must include the **role** attribute of the action.

Actions

An action is an activity that can be requested to be executed by the HPSA T&D Platform. The **action** xml element represents an executable action.

The **action** element is the most complex element as it defines a number of attributes of the action including its **name**, its **type**, its associated **solution**, etc., the lists and details of its **inputParameters** and its **outputParameters** as well as display format related information:

```
<action serviceValidation=”false”>
   <name/>
   <solution/>?
   <label/>?
   <description/>?
   <type/>?
   <actionMode/>?
   <outputParser/>?
   <dispatchType/>?
   <troubleTicketAction/>?
   <alarmAction/>?
   <cost/>?
   <throbber/>?
   <icon/>?
   <averageDurationTime/>?
   <includeMajorInStateVector/>?
   <majorStateVectorName/>?
   <includeMinorInStateVector/>?
   <minorStateVectorName/>?
   <inputParameters>?
      <inputParameter/>*
   </inputParameters>
   <inputGroups>?
      <inputGroup/>+
   </inputGroups>
   <outputParameters>?
      <outputParameter/>+
   </outputParameters>
</action>
```

The **action**’s child elements details are explained below:

name	string	Name of the action used e.g. to request its execution
solution	string	Name of the solution to which this action belongs.
label	string	Optional, a label to display on the user interface to ease the selection of the action.
description	string	A description that make it easier to interpret the purpose of the action on the use interface.
type	enum	Type of the action*: • test • audit • resolve • escalate • internal • read-only This is for future use.

actionMode	enum	Execution mode of the action*: • Open Loop: Requires local interaction to complete • Closed Loop: Runs without any interactions • None The actionMode is for future use.
outputParser	string	An optional parser that may filter the output data of the action*: • XPATH • Regex
dispatchType	string	HPSA. Actions are only dispatched to HPSA. This is for future use.*
troubleTicketAction	string	An optional specification of the type of trouble ticket action*: • create_tt • update_tt • check_tt • close_tt • dissociate_tt • associate_tt • dissociate_close_tt
alarmAction	string	An optional specification of the type of alarm action*: • update_alarm • terminate_alarm
cost	string	Optional used to indicate e.g. especially long duration action.*
throbber	string	The icon used on the GUI to represent the busy or execution state of the action. The icon file is located in: `/opt/uoc2/client/public/images/addons/widgets/fftd-dashboard/`
icon	string	The icon used on the GUI to represent the action and which makes it easier for the operator to select among actions. This option is for future use.
averageDurationTime	int	A GUI display value that indicate the expected duration of the action execution. May e.g. warn the operator about unusual long duration of the action to complete.
includeMajorInStateVector	boolean	If Major code return status parameter should be used as a state-vector item. Only some, more important parameters are influencing the state of the diagnostic process and should be part of the state-vector.
majorStateVectorName	string	The name of the Major code parameters to be used in the state-vector. Hence, provides a parameter name mapping feature.
includeMinorInStateVector	boolean	If Minor code return status parameter should be used as a state-vector item. Only some, more important parameters are influencing the state of the diagnostic process and should be part of the state-vector.

minorStateVectorName	string	The name of the Minor code parameters to be used in the state-vector. Hence, provides a parameter name mapping feature.
inputParameters		Optional list of **inputParamete**r. See below for further details.
inputGroups		One or more input parameters groups used to control the display of multiple input parameters. See below for further details.
outputParameters		Optional list of **outputParameter**. See below for further details.

*) The elements marked with a star are not used directly by the T&D Solution but included for compatibility with UCA product. You are re kindly referred to pertinent UCA documentation regarding the usage of these option elements.

The i**nputGroups element** is an optional list of one or more input parameter groups:

```
<inputGroups>?
   <inputGroup>+
      <parameter/>+
   </inputGroup>
</inputGroups>
```

Each **inputGroup** will be is associated a single form for the listed input parameters. Hence having more than one **inputGroup** will cause multiple forms to be provided in the GUI for parameter input, each form for each group. Not specifying i**nputGroups** will cause all input parameters to appear in a single form.

The **inputParameters** element defines the mandatory and optional input parameters the action requires, the mapping and transfer of parameters with the global state-vector and meta-data storage as well as different GUI related aspects.

Each parameter is configured by the following elements:

```
<inputParameter>*
   <name/>
   <type/>
   <label/>?
   <historyLabel/>?
   < description/>?
   < defaultValue/>?
   <editable/>?
   <hidden/>?
   <includeInStateVector/>?
   <stateVectorName/>?
   <includeInMetaData/>?
   <metaDataName/>?
   <extractFromMetaData/>?
   <includeInStickyData/?
   <stickyDataName/>?
```

```
   <displayFormatType/>?
   <displayFormat/>?
   <mandatory/>?
   < setAsCustomer/>?
   < setAsServiceType/>?
   < setAsProblem/>?
   < setAsResourceType/>? | <setAsResourceInstance/>?
   <helpTexts/>?
   <icon/>?
</inputParameter>
```

Each **inputParameter**'s child element details are explained below:

name	string	Name of the action used e.g. to request its execution
type	enum	The type of the parameter: • String • Integer • Float • Boolean
label	string	Optional, a label to display on the GUI to help the operator to identify the parameter.
historyLabel	string	To be shown (instead of label) on Closed Sessions.
description	string	A description that make it easier to interpret the purpose of the parameter.
defaultValue	string	Value to be used if it is not specified otherwise.
editable	boolean	If the value is allowed to be changed on the GUI by an operator.
hidden	boolean	The parameter is not to be shown on the GUI. E.g. if it is a parameter passed via internal parameters (e.g. state vector items or meta data items). This is e.g. used when chaining multiple actions to be executed without operator intervention.
includeInStateVector	boolean	If parameter should be used as a state-vector item. Only some, more important parameters are influencing the state of the diagnostic process and should be part of the state-vector.
stateVectorName	string	The name of the parameters to be used in the state-vector. Hence, provides a parameter name mapping feature.
includeInMetaData	boolean	The MetaData represents a temporary data store used to maintain global parameters across the execution of actions. Used for parameters that later must be extracted by other actions but which does not qualify as a state-vector parameter.
metaDataName	string	The name of the parameter used in the meta data store. Hence, provides a parameter name mapping feature.
extractFromMetaData	boolean	The parameter must be extracted from an already stored parameter in the MetaData store.

includeInStickyData	boolean	The stickyData represents a GUI specific display area used to main the view of sticky parameters across the execution of actions. The option is used to include the parameter in this sticky data set.
stickyDataName	string	The name of the parameter used in the stickyData store. Hence, provides a parameter name mapping feature.
displayFormatType	string	See below.
displayFormat	string	See below
mandatory	boolean	If the parameter is a mandatory parameter. Otherwise it is optional.
setAsCustomer	boolean	If this parameters should represent the global Customer object.
setAsServiceType	boolean	If this parameters should represent the global ServiceType object.
setAsProblem	boolean	If this parameters should represent the global Problem object.
setAsResourceType	boolean	This parameters is available for compatibility with UCA product.
setAsResourceInstance	boolean	This parameters is available for compatibility with UCA product.
helpTexts	string	One or more help text references (see helpText section below). The help texts may instruct and help the operator on e.g. some specific aspects of the parameter usage.
Icon	string	The icon used on the GUI to represent the parameter and which makes it easier for the operator to identify the parameter among multiple. The icon files are located relative to: `/opt/uoc2/client/public/images/addons/widgets/fftd-dashboard/`

The **outputParameters** element defines the list of output parameters that an action produces. The structure of an **outputParameter** is the same as the **inputParameter** strcuture described above.

The **displayFormat** elements allows configuration of different display related aspects. The details of the **displayFormat** and **displayFormatType** are explained below.

The **displayFormatType** allows specification of the general widget type or appearance of data. It includes the following types:

- Table
- Textarea
- Dropdown
- Icon
- Radiobutton
- Checkbox

Of these **displayFormatTypes** the Dropdown, Icon, Radiobutton and Checkbox assumes a ListDateType structure (see chapter 6: Test & Diagnostic Actions). The **Table** and the **Textarea** uses an additional **displayFormat** to further define the specifics of the format:

The **Textarea** type uses e.g. a <**displayFormat**>**5**</**displayFormat**> to indicate the input area's height in number of lines.

The most complex display formats are associated the **Table** type. This structure is defined as:

```
<fields>
   <field>+
      <name/>
      <label/>
      <type/> (String, Integer, Boolean, Float)
   </field>
<UI>?
  <field>+
</UI>
<Sticky>?
   <group>+
      <label/>?
      <field/>+
   </group>
</Sticky>
<StateVector>?
   <element>+
      <field/>
      <stateVectorName/>
   </element>
</StateVector>
<Meta>?
  <element>+
     <field/>
     <metaDataName/>
  </element>
</Meta>
```

The <**fields**> list of individual <**field**> elements describe the structure of the data field by field. Each field have a name a type and a display label that is used as column heading.

The <**UI**> element lists the (possible fewer) fields that are to be displayed in the table view on the GUI.

The <**Sticky**> element defines <**group**>s of fields to be displayed in the sticky data area in the left part of the GUI. Each group has a display label used as the heading for the fields associated the group. This allows handy availability on the GUI of important information throughout the T&D session.

The <**StateVector**> element allows mapping fields into state-vector items much like what is possible for simple parameters describe earlier.

Similarly, the <**Meta**> element allows assigning fields to persistent items in the MetaData store.

serviceTypes

A serviceType element allows the associating of a CSP's **type** of service with a set of problems specific to that **type** of service:

```
<serviceType>*
   <type/>
   <problems>?
      <problem>*
         <name/>
      </problem>
   </problems>
</serviceType>
```

workflowTemplates

A **workflowTemplate** element allows the association of a HPSA workflow to an action and service type combination:

```
<workflowTemplate>*
   <serviceType/>
   <actionName/>
   <actionSolution/>?
   <workflow/>
   <executionNode/>?
</workflowTemplate>
```

The structure allows the use of a single generic action name that depending on the type of service results in the execution of a service specific workflow.
The optional **executionNode** element allows specification of how the HPSA workflow should be invocated:

- StartJobAndWait — As a normal child workflow (using the StartJobAndWait node)
- ExecuteMacro — Default, as a macro workflow (using ExecuteMacro node).

helpTexts

The definition of **helpTexts** relevant for the T&D solution are all defined in this list. A **helpText** element is structured as:

```
<helpText>*
   <name/>
   <text/>
   <icon/>?
</helptext>
```

The icons are located in the OSS Console under:
`/opt/uoc2/client/public/images/addons/widgets/fftd-dashboard/`

Examples of Dataload Configuration files

This section includes some relative simple configuration files that may help in understanding how to configure a T&D solution. Both the configuration of decoration in the GUI and of the business flow of a T&D Session are described.

The examples maps all actions to just a single WF named `echo`, that takes an input parameter and sends it unchanged to a result parameter. See chapter “Test & Diagnostic Actions” for more details on how to construct Actions and associated workflows, including this `echo` workflow.

Minimum Dataload Configuration

An (almost) minimum dataload configuration file may look as shown in Figure 17 below:

Figure 17 Minimum (almost) TD Data-load file, Minimum.xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TD xmlns="http://www.xml.td.hp.com/Model">
   <solution>TD_logic</solution>
   <groups>
      <group order="0">
         <name>Example</name>
         <states>
           <name>Aresult</name>
         </states>
         <indexes>
            <index>
               <states>
                  <value>-</value>
               </states>
               <actions>
                  <recommended>
                     <action>
                        <name>SomeAction</name>
                     </action>
                  </recommended>
               </actions>
            </index>
            <index>
               <states>
                  <value>+</value>
               </states>
               <endState/>
            </index>
         </indexes>
      </group>
   </groups>

   <actions>
      <action>
         <name>SomeAction</name>
```

```
            <inputParameters>
               <inputParameter>
                  <name>input</name>
                  <type>String</type>
               </inputParameter>
            </inputParameters>
            <outputParameters>
               <outputParameter>
                  <name>result</name>
                  <type>String</type>
                  <editable>false</editable>
                  <includeInStateVector>true</includeInStateVector>
                  <stateVectorName>Aresult</stateVectorName>
               </outputParameter>
            </outputParameters>
         </action>
      </actions>

      <serviceTypes>
         <serviceType>
            <type>Logic</type>
         </serviceType>
      </serviceTypes>

      <workflowTemplates>
         <workflowTemplate>
            <serviceType>Logic</serviceType>
            <actionName>SomeAction</actionName>
            <actionSolution>TD_logic</actionSolution>
            <workflow>echo</workflow>
         </workflowTemplate>
      </workflowTemplates>

      <helpTexts/>
   </TD>
```

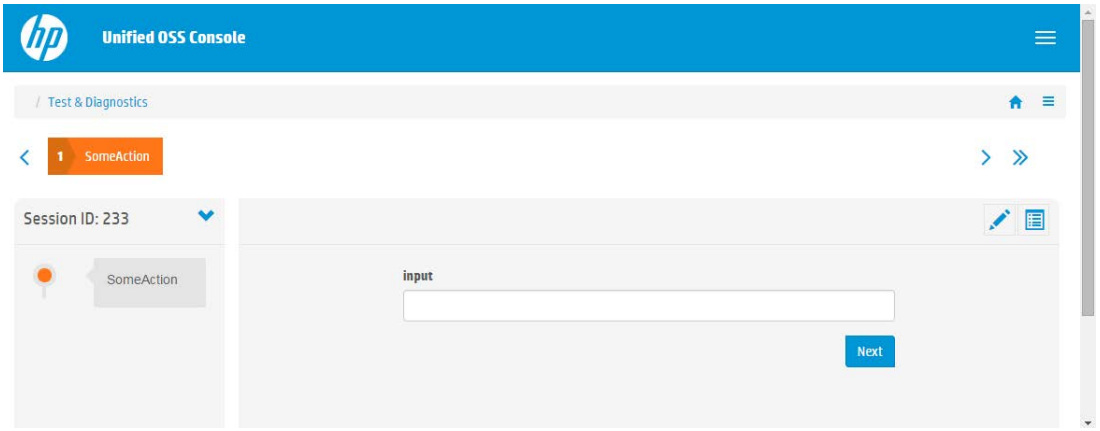
This configuration file defines a single state-item: `Aresult` and a single state index entry with and state-vector value of: `<value>-</value>.` Hence, this index represents the case when `Aresult` has not yet been assigned a value. When the actual state-vector matches this index, the recommended action: `SomeAction` is suggested.

The action `SomeAction` is the only action defined and its input and out parameters are defined in the `<actions>` element to be a single input string parameter ‘input’ and a single output string parameter ‘result’. The input parameter may be specified/modified by the operator and the output parameter is mapped to the state-item `Aresult`.

The action `SomeAction` is mapped to the workflow `echo` in the `<workflowTemplates>` element.

Starting a T&D Session with this configuration, will create to GUI as shown in Figure 18 below.

Figure 18 **Example of a single parameter input form for an action SomeAction.**

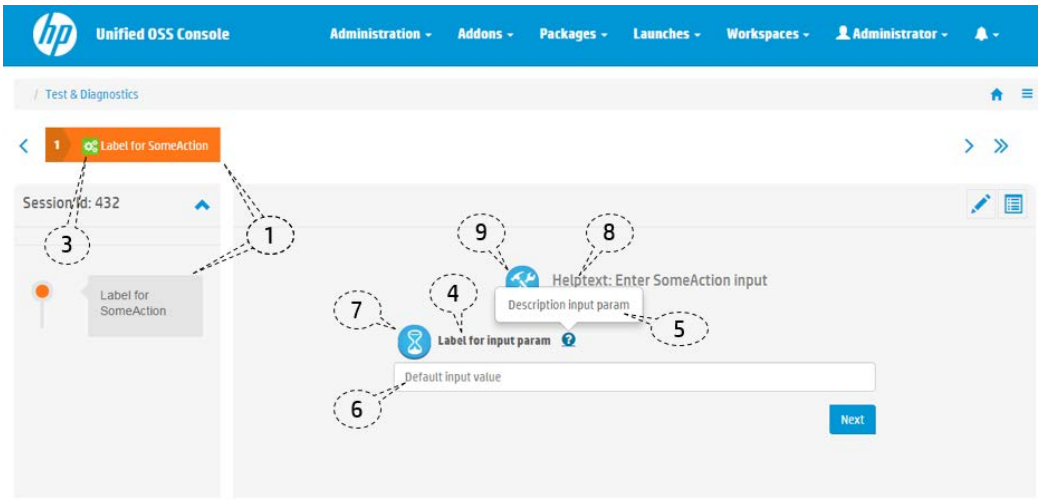


The labels and decorations displayed in Figure 18 corresponds to the default generated items. The solution provides a number of GUI customization options for the configuration of labels, description and help-text besides the usage of icons to make the GUI more user-friendly and/or to align the look-and-feel with the CSPs local preferences.

- Action related decorations
    1. Label: “Label for SomeAction”. The default label is the Action’s name
    2. Description. Not yet implemented.
    3. Icon: Displayed to the left of the action label.

- Parameter
    4. Label: “Label for input param”. The default is the input parameter’s name.
    5. Description: “Input param description”. Appears as tool tip for a “?” icon.
    6. Default value: “Default input value”. Prepopulated in input form.
    7. Icon: Is displayed to the right of input label and tool tip icon.
    8. Help text: “Helptext: Enter SomeAction input”
    9. Help text icon: A .png file in icon directory (see above)

Figure 19 **Example decoration points on action element. Numbers refer to the text above.**

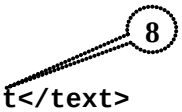


The GUI configuration options are defined on the action element as displayed below in Figure 20 which illustrates part of the data load file `Minimum_with_decorations.xml`:

Figure 20 **Example decoration from load file `Minimum_decorations.xml`. Numbers as above.**

```
...
<actions>
  <action>
     <name>SomeAction</name>
     <label>Label for SomeAction</label>
     <description>Description of SomeAction</description>
     <throbber>hp_throbber.gif</throbber>
     <icon>green_corner_049.png</icon>
     <inputParameters>
        <inputParameter>
           <name>input</name>
           <type>String</type>
           <label>Label for input param</label>
           <description>Description input param</description>
           <defaultValue>Default input value</defaultValue>
           <helpTexts>
              <helpText>input.A</helpText>
           </helpTexts>
           <icon>blue_circle_001.png</icon>
        </inputParameter>
     </inputParameters>
     <outputParameters>
        <outputParameter>
           <name>result</name>
           <type>String</type>
           <label>Label for output</label>
           <description>Description output param</description>
```

```
            <defaultValue>Default output value</defaultValue>
            <editable>false</editable>
            <includeInStateVector>true</includeInStateVector>
            <stateVectorName>Aresult</stateVectorName>
            <helpTexts>
               <helpText>output.A</helpText>
            </helpTexts>
            <icon>red_circle_001.png</icon>
         </outputParameter>
      </outputParameters>
   </action>
</actions>
 ...
<helpTexts>
   <helpText>
      <name>input.A</name>
      <text>Helptext: Enter SomeAction input</text>
      <icon>../../../workspaces/blue_circle_005.png</icon>
   </helpText>
   <helpText>
      <name>output.A</name>
      <text>Helptext: SomeAction output</text>
      <icon>../../../workspaces/green_circle_005.png</icon>
   </helpText>
</helpTexts>
```



The decorations of the result page displaying the output parameters is also configurable similarly to the input page illustrated above. See the example dataload file `Minimum_decorations.xml` for details.

Process Logic Example 1

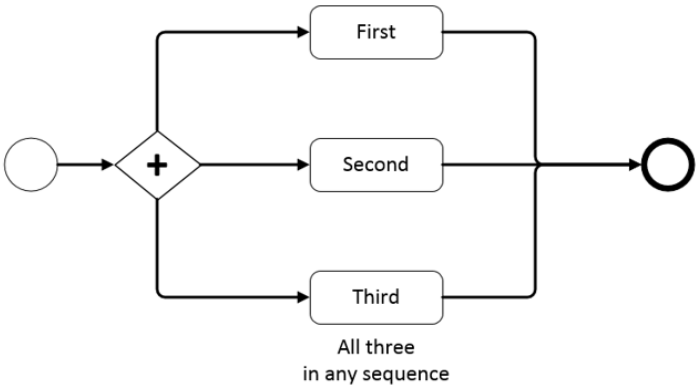
The dataload file `Logic1.xml` represents an example on how to specify the process logic using the state-index configurations.

Figure 21 below, illustrates the desired process as a flow diagram.

The process consists of three actions, FirstAction, SecondAction and ThirdAction and these must each be executed once, but in any sequence. All actions are mapped to a single simple example workflow `echo`.

Corresponding to the three actions, three State-items are defined, Aresult, Bresult and Cresult.

Figure 21 **First Process Flow example from dataload file** `Logic1.xml`



This type of process aligns nicely with the support of wild-cards in the state-index specification (described in chapter 4, section Groups) and the usage of recommended and excluded actions.

The require configuration consists of specifying the combinations of these state-items and the corresponding recommended and excluded actions as illustrated in the Table 1 below.

Table 1 **State-index values for the process flow logic illustrated in Figure 21 above**

Aresult	Bresult	Cresult	Recommended actions	Excluded actions
-	-	-	FirstAction, SecondAction, ThirdAction	
+	-	-	SecondAction, ThirdAction	FirstAction
-	+	-	FirstAction, ThirdAction	SecondAction
-	-	+	FirstAction, SecondAction	ThirdActiion
+	+	-	ThirdAction	FirstAction, SecondAction
...	...	...	...	
+	+	+	{end}	

This state-indexes of the above table is a condensed view of how the configuration appears in the dataload file. The excerpt below illustrates the configuration corresponding to the three first entries in the above table:

```
...
   <index>
      <states>
         <value>-</value>
         <value>-</value>
         <value>-</value>
```

```
      </states>
      <actions>
         <recommended>
            <action>
               <name>FirstAction</name>
            </action>
            <action>
               <name>SecondAction</name>
            </action>
            <action>
               <name>ThirdAction</name>
            </action>
         </recommended>
      </actions>
   </index>
   <index>
      <states>
         <value>+</value>
         <value>-</value>
         <value>-</value>
      </states>
      <actions>
         <recommended>
            <action>
               <name>SecondAction</name>
            </action>
            <action>
               <name>ThirdAction</name>
            </action>
         </recommended>
         <excluded>
            <action>
               <name>FirstAction</name>
            </action>
         </excluded>
      </actions>
   </index>
   <index>
      <states>
         <value>-</value>
         <value>+</value>
         <value>-</value>
      </states>
      <actions>
         <recommended>
            <action>
               <name>FirstAction</name>
            </action>
            <action>
               <name>ThirdAction</name>
```

```
            </action>
         </recommended>
         <excluded>
            <action>
               <name>SecondAction</name>
            </action>
         </excluded>
      </actions>
   </index>
...
```

Details are available in the `Logic1.xml` dataload file which may be inspected and loaded to experiment with this example (see section HP T&D Dataload tool).

Process Logic Example 2

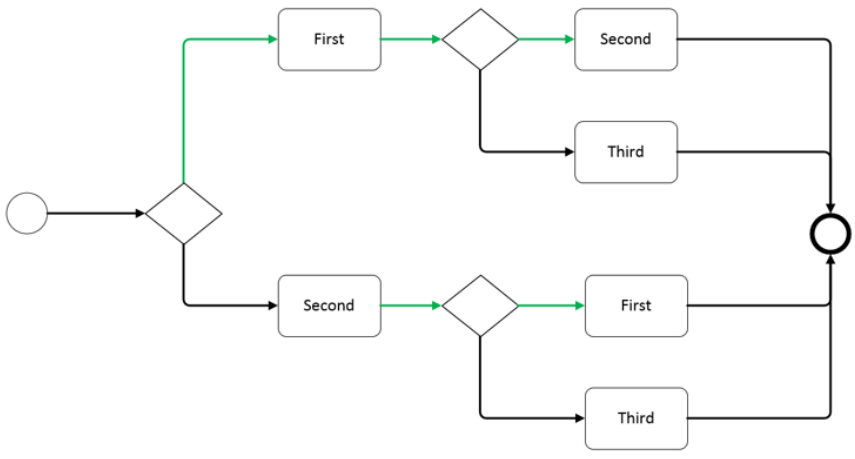
The dataload file `Logic2.xml` represents an additional example on how to specify the process logic using the state-index configurations.

Figure 22 below, illustrates the desired process as a flow diagram. As the previous example, the process consists of three actions, FirstAction, SecondAction a ThirdAction and these must each be executed once, but in any sequence. All actions are mapped to a single simple example workflow `echo`.

The green arrows represent the recommend patch and the black arrows the optional paths.

Corresponding to the three actions, three State-items are defined, Aresult, Bresult and Cresult.

Figure 22 **Second Process Flow example from dataload file `Logic2.xml`. Two out of three actions must be executed.**



Again, this type of process aligns nicely with the support of wild-cards in the state-index specification (described in chapter 4, section Groups) and the usage of recommended and possible actions.

The require configuration consists of specifying the combinations of these state-items and the corresponding recommended and possible actions as illustrated in the Table 2 below.

Table 2 State-index values for the process flow logic illustrated in Figure 22 above

Aresult	Bresult	Cresult	Recommended actions	Possible actions
–	–	–	FirstAction	SecondAction
+	–	–	SecondAction	ThirdAction
–	+	–	FirstAction	ThirdAction
+	+	–	{endState}	
–	+	+	{endState}	
+	–	+	{endState}	

Details are available in the `Logic2.xml` dataload file which may be loaded to experiment with this example.

Process Logic Example 3

The final logic example are based on the dataload file `Logic3.xml`. This example also illustrates how to specify the process logic using the state-index configurations but adds the usage of explicit state-item values to influence the process flow..

Figure 23 below, illustrates the desired process as a flow diagram. As the previous example, the process consists of three actions, FirstAction, SecondAction and ThirdAction. All these three actions are mapped to a single simple example workflow `echo`.

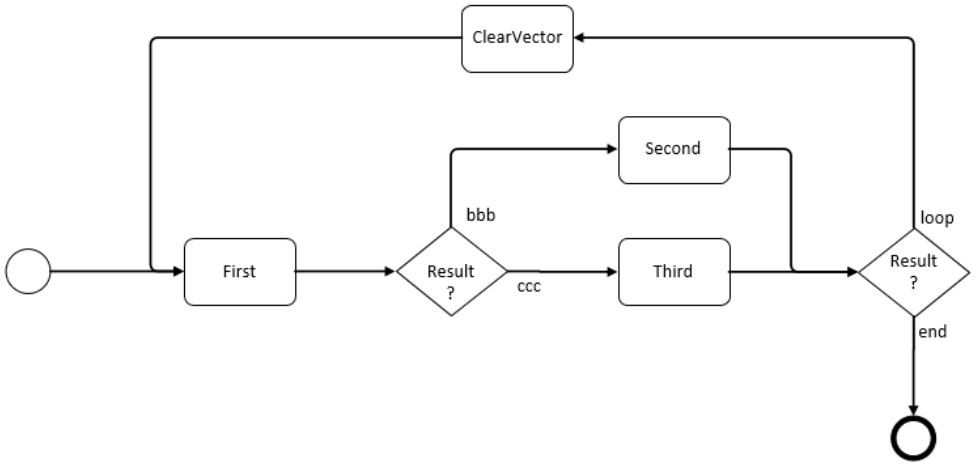
Corresponding to the three actions, three State-items are defined, Aresult, Bresult and Cresult.

This process logic enters FirstAction initially. Its input parameter is simply echoed to its result parameter by the `echo` workflow. By entering the input string as “bbb” or “ccc” either the Second or the ThirdAction is the recommended next action.

Again, using the `echo` workflow for also Second and ThirdAction, the input parameter is echoed as the result parameter, so providing either “loop” or “end” as the input string, either restart the process or terminates it, respectively.

To restart the process properly, the state-vector needs to be cleared. As we cannot ‘unassign’ the state-items clearing is done in this example by reassigning the state-items (Aresult, Bresult, Cresult) with values (o,o,o). This is done by the ClearVector action (mapped to the workflow `reset`).

Figure 23 **Second Process Flow example from dataload file `Logic3.xml`.**



The require configuration consists of specifying the combinations of these state-items and the corresponding recommended and possible actions as illustrated in Table 3 below.

The example also uses the '*' wild-card to match any assigned or unassigned value. This reduces the number of state-index entries that have to be configured, but also requires an understanding of the precedence of how these are matched.

The precedence of the state-index values are:

- Explicit values (e.g. "loop") and '-' the unassigned value has highest precedence
- '+' matching any assigned value comes next
- '*' matching any assigned or unassigned values has lowest precedence

Table 3 **State-index values for the process flow logic illustrated in Figure 23 above**

Aresult	Bresult	Cresult	Recommended actions
-	-	-	FirstAction
o	o	o	FirstAction
bbb	*	*	SecondAction
ccc	*	*	ThirdAction
+	loop	*	ClearVector
+	*	loop	ClearVector
+	end	*	{endState}
+	*	end	{endState}

Details are available in the `Logic3.xml` dataload file which may be inspected and loaded to experiment with this example (see section HP T&D Dataload tool).

HP T&D Dataload tool

NOTE HP Service Activator service should be stopped in order to use the **`TDData`** data-load utility.

The HPSA T&D Solution includes a utility, called `TDData`, to import the configuration of the T&D solution from an XML-formatted file.

The utility may also be used to export the configuration content (e.g. from a development/test system) to a file and to import it again from that file (e.g. on the target system), typically as part of solution installation and deployment on a target system. The utility is found in *`$ACTIVATOR_OPT`*`/bin/`.

The utility needs credentials to establish a session with the database system.

`TDData` must be called with a number of options, from the following:

`-export`	to export the configuration contents from the database
`-import`	to import the configuration contents to the database
`-generateBundle`	to generate a resource bundle file with localizable strings
`-deleteData`	to delete the configuration data from the database. Note it also deletes historical data.
`-dbUser`	user name for session with database
`-dbPassword`	password for session with database
`-dbHost`	Database host (omit if local)
`-dbName`	Database instance name.
`-dbPort`	Database port number (omit if the default port 1521 is used)
`-verbose`	generates verbose output
`-help`	outputs usage information

Exactly one of the `export, import, generateBundle` or `deleteData` options must be specified.

The `deleteData` option may use an optional dataload `filename` argument which deletes only the configuration data in the database which matches the supplied file content. Without a `filename` argument, the `deleteData` option deletes all T&D related configuration data from the database.

The database related options may not (al) need to be supplied and will then be attempted to be obtained from local system configuration. E.g. the `dbUser` and `dbPassword` may be read from a local `dbAccess.cfg` file.

In the development/delivery phase of a T&D project, a dataload file will typically go through a number of iterations. Note, that HP Service Activator should be stopped before an existing already loaded configuration file is deleted (-deleteData), then a new is loaded (-import) after which HP Service Activator must be started again. This cycle of commands could look like the following:

- **`service activator stop`**

- **`TDData –deleteData`**
- **`TDData –import new_dataload.xml`**
- **`service activator start`**

In cases not involving updates of the Action workflows, it may not be necessary to stop/start Service Activator but sufficient to use the reload configuration option in HPSA's menu after deleteData/import new dataload file.

User Authentication and Roles

The HPSA T&D Console (based on UOC) and the HPSA T&D Platform (based on HPSA) both support authentication using LDAP. Hence, LDAP is the recommended authentication mechanism as it requires only configuration of users and credential at the LDAP server.

The required configuration of users and roles is separately described for the HPSA T&D Console and the HPSA T&D Platform below.

HPSA T&D Console

The HPSA T&D Console supports two different authentication methods: SAML and Local. Please see section 1.2 in the UOC User Guide for more information about how to configure these. When using LDAP as a common authentication mechanism, HPSA T&D Console must use SAML.

Authorization is only used in the HPSA T&D Platform, i.e. no special configuration is required in the HPSA T&D Console regarding roles.

HPSA T&D Platform

The HPSA T&D Platform supports 5 different authentication mechanisms. Please see chapter 7 in the HP Service Activator overview document. When using LDAP as a common authentication mechanism the LDAP Authentication Module must be configured in the `mwfm.xml`file.

Two different roles are predefined in the T&D product: TD_ADMIN and TD_VALIDATION. User accounts in HPSA T&D Platform must have assigned the TD_ADMIN role to work as a T&D administrator, to e.g. be able to transfer sessions between other users.

The TD_VALIDATION role must be assigned to user accounts that are required to execute service validation tests.

HPSA Workflow Manager T&D Modules and Nodes

The T&D product includes some components that extend the functionalities of the HPSA platform for the Test and Diagnostic use-cases. These components consist of modules and workflow nodes and are described in the following sections.

HPSA Workflow Manager T&D Modules

The T&D product includes a HPSA Advisor module that implement the central execution logic of the T&D Advisor which is described in section Operator Guidance. The module must be configured in the mwfm.xml configuration file as described in the section: AdvisorModule Configuration.

HPSA Workflow T&D Nodes

The T&D product includes some T&D specific HPSA workflow nodes to ease the construction of the complex XML data structures that controls the appearance on the GUI of lists and of tables.

ConvertBeansToXMLList

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

Creates the XML structure representing a list of beans to be displayed in the T&D Console..

Table 4 ConvertBeansToXMLList Parameters

Name	Required	Description	Default	Type
bean_list	Yes	Array of bean objects	None	Object
value_field	No	The name of the value field in the bean	None	String
label_field	No	The name of the label field in the bean	None	String
icon_field	No	The name of the icon field in the bean	None	String
selected_field	No	The name of selected field in the bean	None	String
ret_variable	No	The variable where the xml structure is returned	None	String

Example `ConvertBeansToXMLList - use in the workflow`

This example creates the XML structure, `productsXMLList,` which represents a list of product names based on the list of products beans. The source of the field is the bean attribute specified by the `value_field` and the destination XML field is the `label_field` name (i.e. `Productname`).

```
<Process-Node disablePersistence="true">
   <Name>Make XML list from bean</Name>
   <Action>
      <Class-Name>
         com.hp.ov.activator.mwfm.component.builtin.ConvertBeansToXMLList
      </Class-Name>
      <Param name="bean_list" value="products"/>
      <Param name="value_field" value="constant:Product_name"/>
      <Param name="label_field" value="constant:ProductName"/>
      <Param name="ret_variable" value="productsXMLList"/>
   </Action>
</Process-Node>
```

ConvertBeansToXMLTable

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

Creates the XML structure representing a table in the T&D Console.

Table 5 ConvertBeansToXMLTable Parameters

Name	Required	Description	Default	Type

bean_list	Yes	Array of bean objects	None	Object
field0 *field1* ... *fieldN*	No	The name of the fields in the bean that should go into the xml structure.	None	String
ret_variable	No	The variable where the xml structure is returned	None	String

Example **ConvertBeansToXMLTable - use in the workflow**

This example creates the XML structure, `customersXMLTable,` which represents a three column table consisting of customer ids, names and addresses, extracted from a list of `customers` beans. The source of the fields are the bean attribute names specified by the `field0, field1` and `field2` parameters.

```
<Process-Node disablePersistence="true">
   <Name>Make XML table from bean</Name>
   <Action>
      <Class-Name>
         com.hp.ov.activator.mwfm.component.builtin.ConvertBeansToXMLTable
      </Class-Name>
      <Param name="bean_list" value="customers"/>
      <Param name="field0" value="constant:Customerid"/>
      <Param name="field1" value="constant:Name"/>
      <Param name="field2" value="constant:Address"/>
      <Param name="ret_variable" value="customersXMLTable"/>
   </Action>
</Process-Node>
```

CreateResponseString

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

Creates the xml response string to send back to the T&D controller workflow from an action workflow.

Table 6 CreateResponseString Parameters

Name	Required	Description	Default	Type
attribute0 *attribute1* ... *attributeN*	Yes	The attribute element to add to the xml structure.	None	Object
value0 *value1* ... *valueN*	Yes	The corresponding values to be assigned to the corresponding attribute elements	None	String
type0 *type1* ... *typeN*	Yes	The corresponding types of the attribute elements.	None	String
ret_variable	No	The variable where the xml structure is returned	None	String

Example **`CreateResponseString - use in the workflow`**

This example creates the XML structure, `customerXML,` which represents a two parameters response structure, consisting of customer id, names and addresses, extracted from a list of `customers` beans. The source of the fields are the bean attribute names specified by the `field0, field1` and `field2` parameters.

```
<Process-Node disablePersistence="true">
   <Name>Make XML table from bean</Name>
   <Action>
      <Class-Name>
         com.hp.ov.activator.mwfm.component.builtin.CreateResponseString
      </Class-Name>
      <Param name="attribute0" value="constant:Customerid"/>
      <Param name="attribute1" value="constant:Name"/>
      <Param name="value0" value="constant:Customerid"/>
      <Param name="value1" value="constant:Name"/>
      <Param name="type0" value="constant:String"/>
      <Param name="type1" value="constant:String"/>
      <Param name="ret_variable" value="customerXML"/>
   </Action>
</Process-Node>
```

HPSA Action Workflows and Parameter Contracts

The result of an action workflows must adhere the workflow contract with the TD Controller workflows. The contract is specified as:

```
<Contract>
   <Input mandatory="7">
      <Var>parent_job_id</Var>
      <Var>message_data</Var>
      <Var>problem_name</Var>
      <Var>action_name</Var>
      <Var>log_manager</Var>
      <Var>log_level</Var>
      <Var>sync</Var>
   </Input>
   <Output>
      <Var>major_code</Var>
      <Var>minor_code</Var>
      <Var>major_description</Var>
      <Var>minor_description</Var>
      <Var>diagnostics</Var>
      <Var>response_string</Var>
   </Output>
</Contract>
```

This contract is also required by the UCA product and must be adhere to for maintaining compatibility with UCA product.

The response_string includes the actual parameters in a structure like:

```
<parameters>*
   <Parameter>+
      <attribute/>
```

```
        <value/>
        <type/>
      </Parameter>
    </parameters>
```

The valid parameter types include:

- `String`
- `Integer`
- `Float`
- `Boolean`

Complex data types are supported that allow e.g. an XML table or an XML list to be assigned to a parameter of type string. See Chapter 6: Test & Diagnostic Actions for more details in complex data types.

5 HP T&D Solution Delivery

The chapter focuses on the delivery and deployment process of a HPSA T&D solution in a specific CSP environment.

The analysis of the CSP's T&D requirements, infrastructure and existing processes must be well understood before the T&D Actions, the T&D Processes and the configuration of the T&D Solution can be made. An up-front full and final analysis is not required as the T&D solution supports a step -wise refinement and extension approach.

Finally, HPSA T&D provides some solution components available out-of-the-box that can be reused and/or used as a template in the delivery process.

The example used in this chapter illustrates a number of configurations of different data types and display formats as well as customized GUI components. It is recommended that the reader installs and experiments with this TD_Example scenario to gain experience and learn how a T&D solution may be configured.

Hence, with no further reference, please remember to use the `dataloadAdvanced.xml` configuration file of the TD_Ex example throughout this chapter.

Customer Process Analysis

A T&D Session maintains the context of a single test and diagnostics process and consists of a sequence of steps (implemented as T&D Actions) that possibly changes the state of the T&D Session. The state of the T&D Session is constructed from individual State-items that each represent a specific aspect of the session and which typically assigned values by the T&D Actions.

The CSP's T&D requirements and existing processes must be analyzed and understood and the corresponding T&D Actions and State-items must be identified.

It is not mandatory that the full process analysis is completed at once; rather it is recommended that the analysis is divided into some steps focusing on groups of activities which may be logically inter-related but somewhat separate from other groups.

TD_Ex Example Scenario Overview

As an example (which is implemented by the TD_Ex example solution, see the section "Example T&D Solution"), consider a customer care center use-case. This is an example and other sequences and steps may be relevant for other CSPs. For this example we assume that the process analysis may have identified that the call agent goes through a common sequence of steps for most support calls:

- Validate the subscriber.
  A call may origin from a person different from the subscriber; it may be made from a phone line not part of the subscriber's services, etc. Hence, there will often be a need for an operator assisted identification of the actual subscriber that the reported problems concerns.

- Identify the Services purchased by the subscriber.
  Having identified the subscriber, the services purchased by this subscriber can be identified from the CSPs inventory or billing systems.

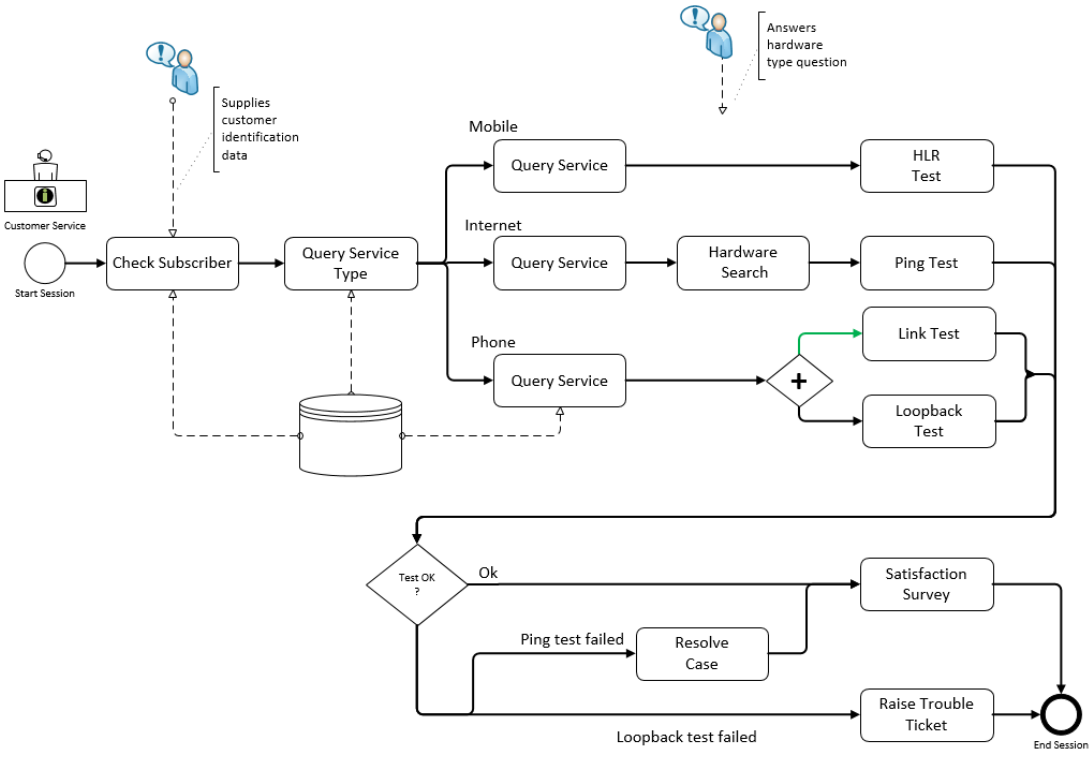
- Select the faulty service to be tested among these services.
  The service for which the reported problem applies must be selected from the set of subscriber services. This will often be based on the call agent's dialog with the subscriber.

- For the selected service select among possible problem related test actions to identify the cause of the issue.
  For a specific service a number of test actions may be defined among which the best to diagnose the problem is selected based on the call agent's dialog with the subscriber and from the recommendations provided by the T&D advisor that will guide the operator to the best test actions to execute.

- Finally, the result of the T&D session may e.g. suggest that all is working fine, or it may resolve (i.e. repair) the issue by some available actions, or there could be some more laborious cause identified that requires further investigation requested by e.g. raising a trouble ticket.

Figure 24 below illustrates the general business process flow that the TD_Ex scenario implements.

Figure 24 Business Process implemented by the TD_Ex example scenario.



As described above, the example process consists of some main phases and the concept of groups of state-indexes described in xxx lends itself to reflect such aspects. In the Example the following state-index groups are defined:

- Common Info
  This group defines the state-index values related to identification of the subscriber, her service types and the selection of the SuT.

- Internet
  This group defines the state-indexes related to the tests actions and finalization action related to the Internet SuT

- Mobile
  This group defines the state-indexes related to the tests actions and finalization actions related to the Mobile SuT

- Home Phone
  This group defines the state-indexes related to the tests actions and finalization actions related to the fixed line telephony SuT

Let's look a little closer on the analysis of the **Common Info** group.

The first step in the analysis is to identify the important State-items, the T&D Actions and their parameters mappings that are pertinent for this group of actions. Table 7 below provides an example of a table to be used when analyzing a CSP's scenarios and an example outcome of this analysis. In a real scenario a more extensive table will be required.

The parameters of the T&D Actions are listed in the Parameter column. These may depend on the already existing actions available or they may be identified as part of this analysis.

The column **Type** lists for each parameter its type. E.g. the *customerFound* parameter is a Boolean that represents the important step in the process when the subscriber in the call has been validated.

The identified State-item mappings are listed in the **State-item** column. The State-items represent the key aspect of the steps analyzed and which controls the sequence of steps in the T&D Session. The State-items values are assigned from a parameter and this allows also a mapping of names, so e.g. the State-item *subscriberId* is mapped from the output parameter *customerFound*.

Table 7 Example Parameter, State-item and T&D Action identification from process analysis

Parameter	Type	Created/updated by Action	State-item	Metadata-item	Required by Action(s)
customerFound	Boolean	checkSubscriber	*subscriberId*	-	queryService, queryServiceType
phoneNumber	Boolean	checkSubscriber	*phoneNumber*	-	-
serviceList	String	queryServiceType	*serviceType*	sType	queryService
serviceList	String	queryService	*services*	serviceList	hardwareSearch

These State-items also represents persistent information that the T&D process may use or refer to in other steps/actions. But additional information not related to the state of the process may also be necessary to pass between actions and such parameters re typically mapped to Metadata-items.

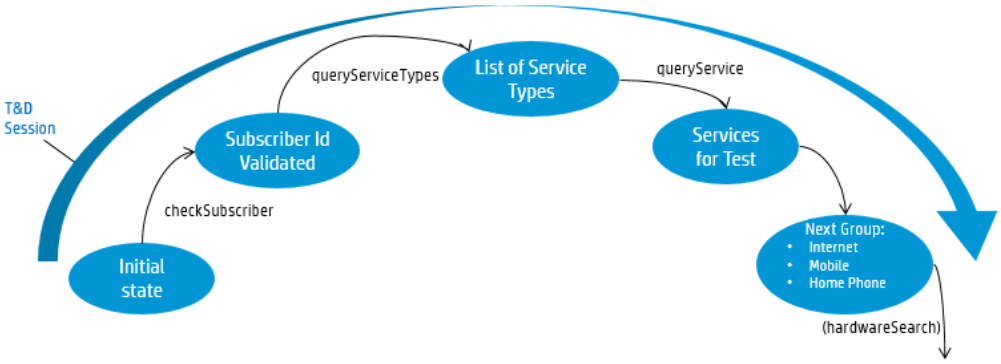
The column **Required by Action(s)** list the T&D Action that is going to use the state-item or metadata-item e.g. for information look-ups.

The table doesn't show all details/attributes of the parameters and state-items. E.g. a Subscriber item containing the full details of the subscriber may be of general use throughout the T&D Session. To avoid repeated fetches of this kind of information it is assigned to Stickydata-items.

These state-items and T&D Actions (grouped into the “Common Info”) are required to identify the subscriber, the subscriber’s services before selecting, from possibly multiple service instances, the Service under Test (SUT) which is suffering some reported problem. For the selected SuT the recommended and possible actions are selected for further test and problem cause identification.

The T&D Session is represented as a sequence of state-transitions involving these State-items and T&D Actions as displayed in the state-transition diagram in Figure 25.

Figure 25 State-transition diagram of “Common Info” group corresponding to Table 7 above



The state of the T&D Session is maintained as a State-vector that consists of the dynamic list of State-items, which represent the key aspects of the T&D Session and which are compared with the configured State-indexes to fetch the next actions that are recommend or possible to execute.

Figure 26 Configuration of the state-indexes corresponding to the above State-transition example.

subscriberId	phoneNumber	serviceType	services		Recommended actions
-	-	-	-	...	checkSubscriber
+	+	-	-	...	queryServiceType
+	+	+	-	...	queryService
+	+	Internet	+	...	-> Internet
+	+	Mobile	+	...	-> Mobile
+	+	Home Phone	+	...	-> Home Phone

Figure 26 illustrates to configuration of the state-indexes corresponding to the above Common Info group example.

At any point of the T&D Session, the State-vector will have a specific value (i.e. a specific sequence of State-items). The value of the State-vector is used to identify a matching entry in the State-index table. Each State-index entry contains the configured operator guidance (Recommended actions) which the T&D Advisor may provide to a human operator.

Figure 26 illustrates the State-index table and as may be observed, the values in the indexes, may contain actual explicit values like 'Internet' or wildcards as '+', '-' or '*'.

The wildcards have the following interpretation when searching for a match and are listed in their order of precedence, i.e. a '-' match is preferred to a '+' match (an explicit value has same precedence as a '-'):

- '-' (minus) matches only unknown values (i.e. yest unassigned values).
- '+' (plus) matches any known value.
- '*' (star) matches any value of the item including unknown values. So basically a "don't care".

TD Example Scenario

Datails around the configuration and impementaion of a T&D solution are illsuatrated via the TD_Ex example scenario.

TD_Ex Example Scenario State-index Configuration Details

This section describes the configuration details of the State-index table used by the T&D Advisor to match the current State-vector with a State-index to identify the proper entry of guidance information.

The `dataloadAdvanced.xml` dataload file is used throughout and may be inspected by the reader.

Figure 27 below shows the detailed XML configuration of the state-index table corresponding to the Common Info group.

Figure 27 Configuration details of the State-index table of the Common Info group.

```
  ...
<group order="0">
    <name>Common Info</name>
    <states>
        <name>subscriberId</name>
        <name>phoneNumber</name>
        <name>serviceType</name>
        <name>services</name>
    </states>
    <indexes>
        <index>
            <states>
                <value></value>
                <value></value>
                <value></value>
                <value></value>
            </states>
            <actions>
```

```
            <recommended>
                <action>
                    <name>checkSubscriber</name>
                </action>
            </recommended>
        </actions>
    </index>
    <index>
        <states>
            <value>+</value>
            <value>+</value>
            <value></value>
            <value></value>
        </states>
        <actions>
            <recommended>
                <action>
                    <name>queryServiceType</name>
                </action>
            </recommended>
        </actions>
    </index>
    <index>
        <states>
            <value>+</value>
            <value>+</value>
            <value>+</value>
            <value></value>
        </states>
        <actions>
            <recommended>
                <action>
                    <name>queryService</name>
                </action>
            </recommended>
        </actions>
    </index>
    <index>
        <states>
            <value>+</value>
            <value>+</value>
            <value>Internet</value>
            <value>+</value>
        </states>
        <nextGroup>Internet</nextGroup>
    </index>
    <index>
        <states>
            <value>+</value>
```

```
                        <value>+</value>
                        <value>Mobile</value>
                        <value>+</value>
                    </states>
                    <nextGroup>Mobile</nextGroup>
                </index>
                <index>
                    <states>
                        <value>+</value>
                        <value>+</value>
                        <value>Home Phone</value>
                        <value>+</value>
                    </states>
                    <nextGroup>Home Phone</nextGroup>
                </index>
            </indexes>
        </group>
          ...
```

The usage of groups to partition your solution and in particular your analysis and implementation work is highly recommended.

TD_Ex Example Scenario Actions Configuration Details

This section describes the configuration details of the T&D Actions to be used in a T&D Solution.

The dataloadAdvanced.xml dataload file is used throughout and may be inspected by the reader.

The T&D Actions are configured according to the following steps:

- Define the Name and Type of the Action.
  Currently the Type is for future use and 'Internal' is used in the examples.

- Define the Input and Output parameters.
  This includes their type and mapping to/from StateVector, MetaData and/or StickyData store parameters.

- Define the display decorations and formats.
  This includes labels, icons, default values, help texts and advanced display formats.

- Define the associated HPSA workflow.
  Each Action is mapped to a HPSA workflow that implement the actual functionalities of an Action.

The Figure 28 below displays as an example, the details of the **`queryService`** action configuration in the `dataloadAdvanced.xml` file related to the above first three bullets.

Figure 28 Configuration details of the `queryService` Action of the Common Info group.

```
<action>
   <name>queryService</name>
   <label>!!queryService!Query Service</label>
   <description>!!queryServiceDesc!Find service</description>
   <type>internal</type>
```

```
      <actionMode>Open Loop</actionMode>
      <outputParser>None</outputParser>
      <dispatchType>HPSA</dispatchType>
      <cost>1 seconds</cost>
      <inputParameters>
         <inputParameter>
            <name>subscriberId</name>
            <type>String</type>
            <editable>false</editable>
            <hidden>true</hidden>
            <extractFromMetaData>CustomerId</extractFromMetaData>
         </inputParameter>
         <inputParameter>
            <name>serviceType</name>
            <type>String</type>
            <editable>true</editable>
            <hidden>true</hidden>
            <extractFromMetaData>sType#Value</extractFromMetaData>
         </inputParameter>
      </inputParameters>
      <outputParameters>
         <outputParameter>
            <name>serviceList</name>
            <type>String</type>
            <label>!!Service!Service</label>
            <description>!!ServiceDesc!Service List</description>
            <editable>true</editable>
            <includeInStateVector>true</includeInStateVector>
            <stateVectorName>services</stateVectorName>
            <includeInMetaData>true</includeInMetaData>
            <metaDataName>serviceList</metaDataName>
            <displayFormatType>Dropdown</displayFormatType>
            <mandatory>true</mandatory>
            <helpTexts>
               <helpText>queryServiceHelpText1</helpText>
            </helpTexts>
         </outputParameter>
      </outputParameters>
   </action>
```

As mentioned above, an action's relation with a HPSA workflow must also be configured. Typically an Action maps to a single workflow, but different Actions may map to the same workflow. So an example of the final step in the Action configuration is provided in Figure 29 below.

Figure 29 Configuration details of the `queryService` Action's mapping to a HPSA workflow.

```
   <workflowTemplate>
      <serviceType>any</serviceType>
      <actionName>queryService</actionName>
```

```
    <workflow>queryServiceAdvanced</workflow>
</workflowTemplate>
```

The **serviceType** element is for future use to possibly allow action mapping of the same action to different workflows base on different serviceTypes.

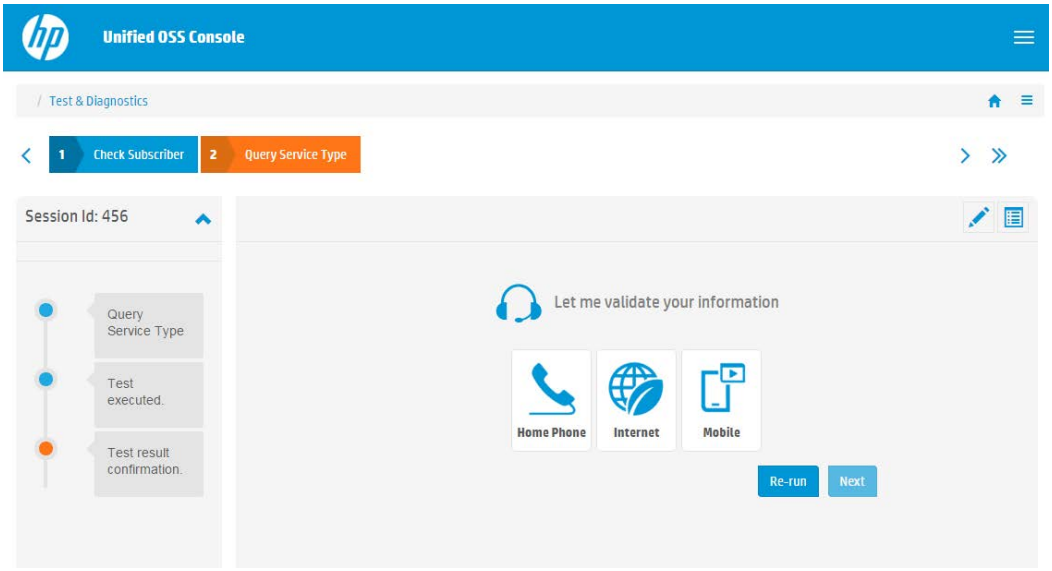
TD_Ex Example Scenario Display Format Configuration Details

This section describes the configuration details of GUI display related aspects that allow adaptation of the GUI to local preferences.

The `dataloadAdvanced.xml` dataload file is used throughout and may be inspected by the reader

As an example of some of the GUI configuration capabilities, we use the page related to result of the `queryServiceType` action. The page is illustrated in Figure 30 below.

Figure 30 Result page of queryServiceType action.



The display configuration related to this page is listed in Figure 31 below.

Figure 31 Outputparameters configurations of queryServiceType action.

```
<outputParameters>
   <outputParameter>
      <name>serviceList</name>
      <type>String</type>
      <label>!!Service!Service</label>
      <description>!!ServiceDesc!Service List</description>
      <editable>true</editable>
      <includeInStateVector>true</includeInStateVector>
```

```
            <stateVectorName>serviceType</stateVectorName>
            <includeInMetaData>true</includeInMetaData>
            <metaDataName>sType</metaDataName>
            <includeInStickyData>true</includeInStickyData>
            <displayFormatType>Icon</displayFormatType>
            <mandatory>true</mandatory>
            <helpTexts>
               <helpText>queryServiceTypeHelpText1</helpText>
            </helpTexts>
         </outputParameter>
      </outputParameters>
        ...
      <helpText>
         <name>queryServiceTypeHelpText1</name>
         <text>!!queryServiceTypeHelpText1!Let me validate your
information</text>
         <icon>headphones.png</icon>
      </helpText>
```

The **displayFormatType** is specified as **Icon** which means, that the list of serviceTypes returned by the action, is displayed as the row of associated Icon as Figure 30 shows.

The association of icons to serviceTypes must be done by the action workflow (`queryServiceAdvanced`) that 'knows' which serviceTypes are being populated in the list. The XMLList structure that the workflow returns as an output parameter contains the assigned icon names which the T&D Console GUI will use for display.

The way an actual action workflow obtains the icon names depends on the specific solution environment. In this example case the icons are defined as part of the database schema of the different service types and therefore available in the bean structure directly.
In other cases, the workflow may need to be programmed to include the icons names directly or these could possibly be fetched from some different external system and possibly added to an in-memory bean before being converted by e.g. the ConvertBeansToXMLList workflow node.

The parameter exchange conventions are described further in the chapter Test & Diagnostic Actions.

As an example of a Table type display format, we may use the page related to result of the `checkSubscriber` action. The page is illustrated in Figure 32 below.

The response includes the output parameter `anotherResult` that is associate a **Table** display format as illustrated in Figure 33 below.

Figure 32 **Table type display format example from `checkSubscriber result`.**

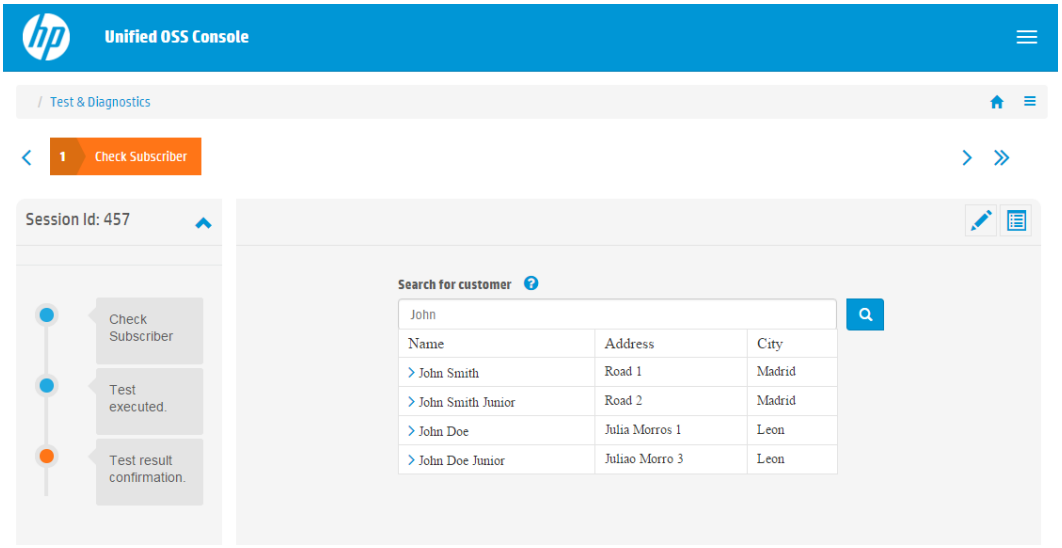


Figure 33 **`Dataload configuration snippet of the checkSubscriber result.`**

```
<outputParameter>
   <name>anotherResult</name>
   <type>String</type>
   <label>!!Customer!Customer</label>
   <editable>true</editable>
   <includeInMetaData>true</includeInMetaData>
   <includeInStickyData>true</includeInStickyData>
   <displayFormatType>Table</displayFormatType>
   <displayFormat><![CDATA[
      <table xmlns=
             "http://www.hp.com/td/xml/complextypes/table">
         <fields>
            <field>
               <name>Index</name>
               <label>Index</label>
               <type>Integer</type>
            </field>
            <field>
               <name>Name</name>
               <label>Name</label>
               <type>String</type>
            </field>
            <field>
               <name>Address</name>
               <label>Address</label>
               <type>String</type>
            </field>
            <field>
               <name>City</name>
```

```
            <label>City</label>
            <type>String</type>
         </field>
         <field>
            <name>Country</name>
            <label>Country</label>
            <type>String</type>
         </field>
      </fields>
      <UI>
         <field>Name</field>
         <field>Address</field>
         <field>City</field>
      </UI>
      <Sticky>
         <group><field>Name</field></group>
         <group>
            <label>Contact Information</label>
            <field>Address</field>
            <field>City</field>
            <field>Country</field>
         </group>
      </Sticky>
      <Meta>
         <element>
            <field>Index</field>
            <metaDataName>CustomerId</metaDataName>
         </element>
      </Meta>
   </table>]]>
</displayFormat>
```

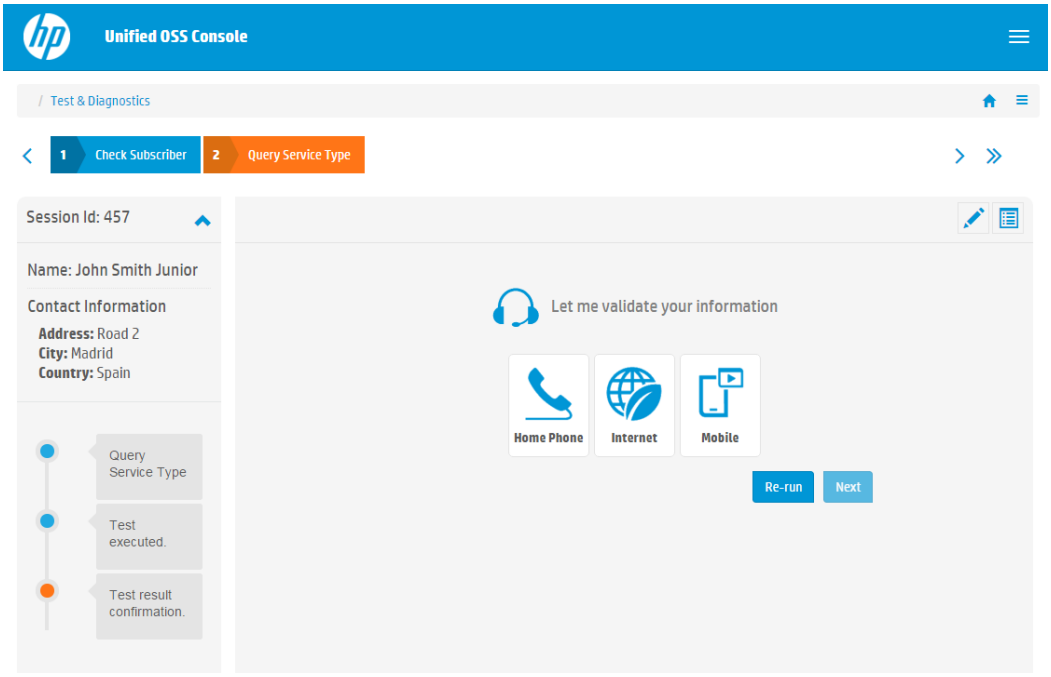
The **fields** element defines the structure of the data, field for field as it is returned in the response. The **UI** element defines the fields to be displayed using their associated **label** element as heading names as may be seen in Figure 32 above.

When one of the entries in the table is selected as the actual subscriber, the **Meta** element makes the Index of that table entry available as a `CustomerId` parameter in the meta-data store. Similarly, the **Sticky** element makes the details of the selected subscriber available in the sticky-data store. These sticky data are displayed in the left pane of the GUI window as illustrated in Figure 34 below as details of “John Smith Junior”.

This sticky data is made available through-out the T&D Session and useful to provide handy and fast access to important data of the T&D Session.

The parameter exchange conventions are described further in the chapter Test & Diagnostic Actions.

Figure 34 **Example of display of sticky data related to the selected subscriber in left pane.**



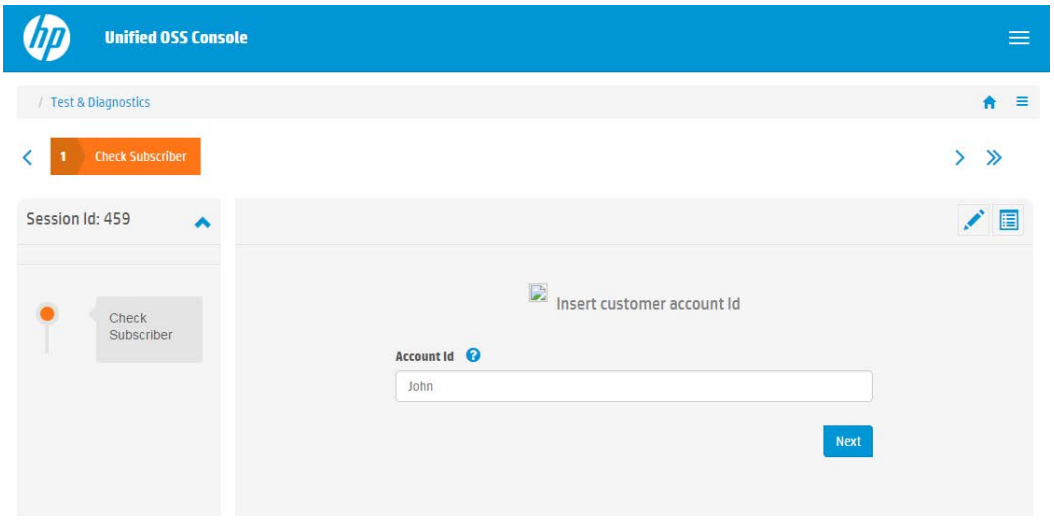
TD_Ex Example Scenario Customized T&D Console GUI forms

Although quite many display aspects are configurable, in real deployments scenarios it is most likely that additional and more elaborate GUI pages are desired.

This section provides an example of a small customization made for the `checkSubscriber` pages.

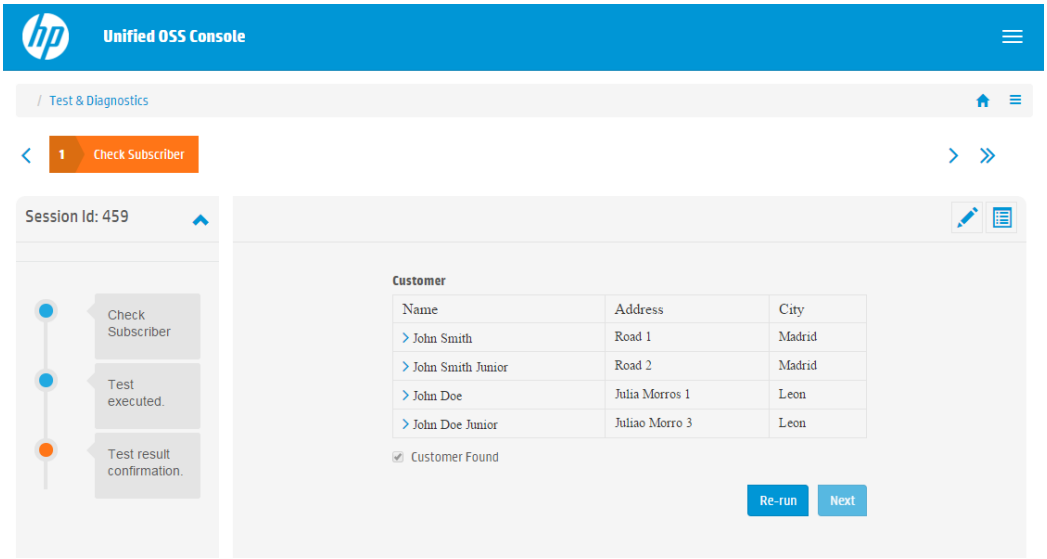
First let's see how the default generated pages would look like using just the dataload configurations. The first page is the input page where the subscriber information may be entered in a search form, illustrated in Figure 35 below.

Figure 35 **Default generated input page using just the dataload configurations options**



When the search is made using the Next button, the next page displayed is the result page illustrated in Figure 36 below.

Figure 36 **Default generated output page using just the dataload configurations options**



Finally, when the correct subscriber is selected in the table and the Next button submitted we continue with the QueryServiceType page illustrated already in Figure 30 above.

The two pages illustrated in Figure 35 and Figure 36 are customized by the files located in the TD_Ex solution directory in:

`TD_Ex/etc/console/server/addons/plugins/td/forms`

and consist of the files:

- `checkSubscriber_WAITING_INPUT.json`
  Customizes the input page illustrated in Figure 35

- `checkSubscriber_WAITOUT_INPUT.json`
  Customizes the output view displayed in Figure 36 e.g. by adding an input form for repeated searches

- `checkSubscriber_WAITOUT_OUTPUT.json`
  Customizes the table of subscribers listed in the output view displayed in Figure 36.

When deploying the TD_Ex solution, these files will be copied to the T&D Console into location: `/opt/uoc2/server/addons/plugins/td/forms.` This illustrates the naming convention of customizations using the action name followed by one of the three extensions.

The below Figure 37 and Figure 38 illustrates the corresponding customized versions of the above two pages that are deployed by the TD_Ex solution.

Figure 37 TD_Ex customized version of Figure 35

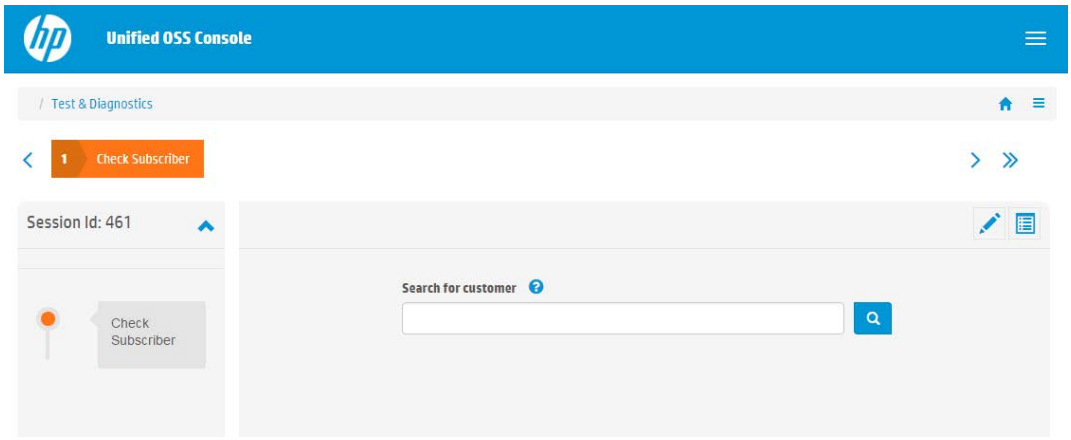
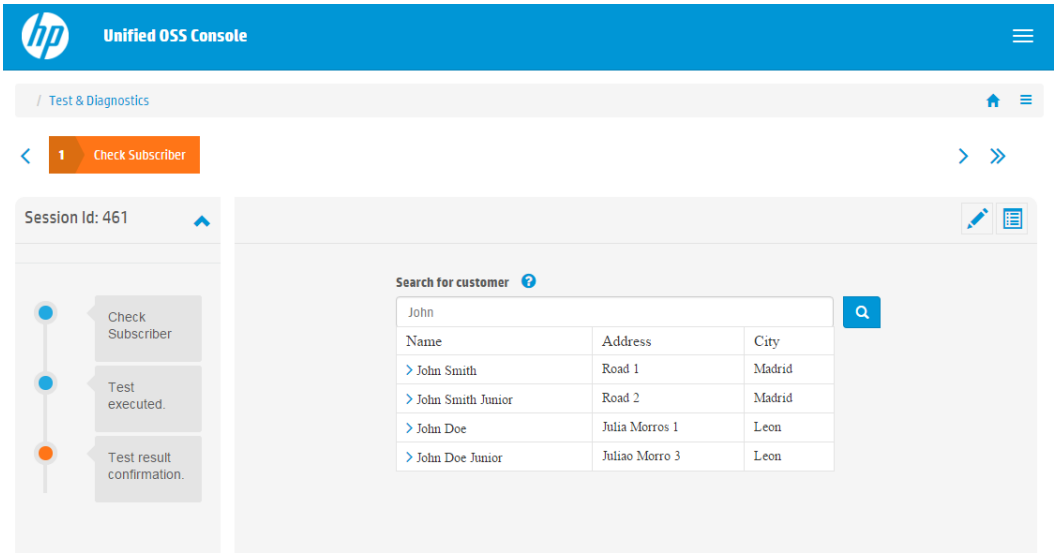


Figure 38 TD_Ex customized version of Figure 36



External System Integration

The HPSA T&D Platform supports integration with external requestor systems via its North-bond interface that is used by the HPSA T&D Console, with external systems like inventory and trouble ticketing system via native HPSA module interfaces as well with test resources and network elements via the HPSA native plugin interfaces.

6 Test & Diagnostic Actions

This chapter is primarily of interest for the system integrator who will implement the T&D processes, the T&D Actions and their associated HPSA workflows.

This chapter uses the example scenario to illustrate a number of configurations of different data types and their exchange between Actions workflows and the T&D Console. It is recommended that the reader installs and experiments with this TD_Example scenario to gain experience and learn how a T&D solution may be implemented.

The configuration of simple display aspects and decoration in the GUI is described in section: HP T&D Dataload Configuration File.

In this section a closer look on the parameter exchange formats are described based on the TD_Ex example scenario.

The sample scenario actions and associated workflows of the TD_Ex scenario are listed in Table 8 below.

Table 8 Actions and associated HPSA workflows in TD_Ex example scenario

Action	Workflow
checkSubscriber	`checkSubscriberAdvanced`
queryService	`queryServiceAdvanced`
queryServiceType	`queryServiceTypeAdvanced`
hardwareSearch	`hardwareSearch`
linkTest	`linkTest`
loopbackTest	`loopbackTest`
coreServiceTest	`coreServiceTest`
pingTest	`pingTest`
HLRTest	`HLRTest`
resolveCase	`resolveCase`
troubleTicket	`troubleTicket`

satisfactionSurvey	`satisfactionSurvey`

The exchange of parameters most adhere the conventions imposed by the HPSA TD product which includes the workflow contract between the action workflows and the controller workflows. It also includes the parameters exchange formats used on the REST interface that allows the specification of displays formats in the T&D Console.

The result of an action workflows must adhere the workflow contract with the TD/Controller workflow. The contract is e.g. specified in the hardwareSearch workflow as:

```
<Contract>
   <Input mandatory="7">
      <Var>parent_job_id</Var>
      <Var>message_data</Var>
      <Var>problem_name</Var>
      <Var>action_name</Var>
      <Var>log_manager</Var>
      <Var>log_level</Var>
      <Var>sync</Var>
   </Input>
   <Output>
      <Var>major_code</Var>
      <Var>minor_code</Var>
      <Var>major_description</Var>
      <Var>minor_description</Var>
      <Var>diagnostics</Var>
      <Var>response_string</Var>
   </Output>
</Contract>
```

This contract is required by the UCA product and must be adhere to maintaining compatibility with UCA product.

The response string as an XML structure containing the more action specific result parameters. The basic structure is a list of (attribute, value, type) tuples per parameter, but the value element may also be of complex type to allow more advanced display format specifications.

In the hardwareSearch workflow the example illustrated in Figure 39 may be inspected.

Figure 39 Example of the XML structure of the response_string parameter from `hardwareSearch` example workflow.

```
<Parameters>
   <Parameter>
      <attribute>result</attribute>
      <value>ok</value>
      <type>String</type>
   </Parameter>
   <Parameter>
      <attribute>hardwareList</attribute>
```

```
      <value><![CDATA[<list
xmlns="http://www.hp.com/td/xml/complextypes/list/data"><value
icon="verizon.png" label="Verizon Quantomm">27</value><value
icon="verizon9100.png" label="Verizon
9100">28</value></list>]]></value>
      <type>String</type>
   </Parameter>
</Parameters>
```

This advanced list structure of a value element is specified by the schema defined in *$ACTIVATOR_ETC* /config/TypeListData.xsd and illustrated in Figure 40 below.

Figure 40 Advanced parameter value `ListDate` structure definition from `TypeListData.xsd`

```
<element name="list">
  <complexType>
    <sequence>
      <element name="value" maxOccurs="unbounded" minOccurs="0">
        <complexType>
          <simpleContent>
            <extension base="string">
              <attribute name="icon" type="string" />
              <attribute name="label" type="string" />
              <attribute name="selected" type="boolean" />
            </extension>
          </simpleContent>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
```

The TypeDataList structure is used by a number of parameter displayFormatTypes as described in the section: HP T&D Dataload Configuration File.

- Dropdown
- Icon
- Radiobutton
- Checkbox

The parameter displayFormatType Table assumed a value structured as a DataTableType which is specified by the *$ACTIVATOR_ETC* /config/TypeTableData.xsd. This is illustrated in Figure 41 below.

Figure 41 **Advanced parameter value `TableDate` structure definition from `TypeTableData.xsd`**

```
<element name="table">
    <complexType>
      <sequence>
        <element name="row" maxOccurs="unbounded" minOccurs="0">
          <complexType>
            <sequence>
              <element name="cell" type="tns:CellType"
                       maxOccurs="unbounded" minOccurs="1" />
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
  <complexType name="CellType">
    <sequence>
      <choice>
        <element name="value" type="string"
                 maxOccurs="1" minOccurs="0" />
        <element name="list" maxOccurs="1" minOccurs="0">
          <complexType>
            <sequence>
              <element name="value" type="string"
                       maxOccurs="unbounded" minOccurs="1" />
            </sequence>
          </complexType>
        </element>
      </choice>
    </sequence>
  </complexType>
```

The somewhat daunting task of specifying the parameters values is eased by using the available workflow nodes described in section HPSA Workflow T&D Nodes. E.g. the generation of a DataList type is conveniently done using the **`ConvertBeansToXMLList`** workflow node (described in section HPSA Workflow T&D Nodes).

Similarly the generation of a DataTable type is conveniently done using the **`ConvertBeansToXMLTable`** workflow node (also described in section HPSA Workflow T&D Nodes).

A-1 North-bund ReST API

The purpose of this appebdix is to specify the architecture of the T&D REST Interface for T&D process guidance to support the operator in progressing correctly in the T&D process.

The REST interface is the layer that connects the backend of the solution (HPSA) with the OSS console or other north-bound systems.

A brief description of each REST method is provided in the next sections.

Overview

REST is a collection of principles rather than a set of standards. This T&D REST interface consist of several methods that complies with REST best practices. A REST interface is web based and uses the HTTP verbs GET, PUT, POST and DELETE for the CRUD operations and a url as the object.

The WFManager needs to be provided in every single method as a parameter, because it is the way T&D links with HPSA. So it is going to be intentionally omitted in each method parameter list explanations below.

The JSON response descriptors contain the HTTP status of the request, returning the same HTTP Status code returned by the server.

T&D REST Methods

users

HTTP Verb: GET, Path: /users

Provides a list of the HPSA´s users. No special role is requires to be invoked, so it is used to get a list of available users before transferring a session.

Return Type: UserListDescriptor

Contains an array with the names of the users.

Parameters:

- None.

userIsInRole

HTTP Verb: GET - Path: / userIsInRole

Returns 'true' if the operator is included in the specified the role name.

Return Type: IsInRoleDescriptor

Returns true if the operator is in the role, false if not.

Parameters:

- operatorName and roleName. Cannot be empty.

startSession

HTTP Verb: POST - Path: /sessions

Starts a new session. That means, creating a new job based on the TD_CONTROLLER workflow.

Return Type: SessionIdDescriptor

See Appendix to get more details

Parameters:

- language: mandatory, cannot be empty.

searchSessions

HTTP Verb: GET - Path: /sessions

Returns a list of the TD sessions, it means that looks for the jobs associated with the TD_Controller, and also allows a filtering on customer, operator, and date. It also supports pagination.

The underlying logic is based in a private method which is the one that search the sessions that fulfill the filter, the pagination, and besides, accept as a parameter the corresponding workflow. (In this case, TD_Controller.)

Return Type: SessionsDescriptor

results: an array of the SessionDescriptor* retrieved. (See Appendix to get more details)

count: a field that indicates the number of sessions retrieved.

totalCount: Not in use yet, it is intended to help in pagination.

Parameters:

- operatorName: String. Name of the operator.
- customerName: String. Name of the customer.
- sessionId: String. Retrieves only the session indicated.
- fromDate and toDate: Filter start time. Valid formats are:
  - yyyy-MM-dd HH:mm:ss.SSS
  - yyyy-MM-dd HH:mm:ss.SSS Z
  - yyyy-MM-dd HH:mm:ss
  - yyyy-MM-dd HH:mm:ss Z
- numberOfResults: String. Results per page.

searchHistoricalSessions

HTTP Verb: GET - Path: /sessions/searchHistory

Returns an array containing the terminated sessions. Filters, sorting and pagination are allowed.

Return Type: HistoricalSessionDescriptor

totalCount: Number of sessions retrieved

historicalSessions: Array of terminated sessions. Each terminated session consist of another array of ActionInstanceDescriptor returning a description of each actions executed for a single test. (See Appendix to get more details)

Parameters:

- Those needed to filter, sort and paginate.
    - Filter and sort: operatorName, customerName, sessionId, terminationState, fromDate, toDate
    - Paginate: numberOfResults, startingIndex, sort_by, asc (Boolean)

transferSession

HTTP Verb: PUT - Path: /session/{sessionId}

Transfers the session to a new operator.

Return Type: SessionDescriptor

Description of the session that has been transferred. (See Appendix to get more details)

Parameters:

- sessionId to be transferred.
- operatorName: new operator.

terminateSession

HTTP Verb: DELETE - Path: /session/{sessionId}

Ends the job, and stores relevant information into the database.

Return Type: SessionIdDescriptor

Returns the status code 200 if the action has been successfully terminated, and the sessionId.

Parameters:

- closeCode: Stored in the session History.

getSessionState

HTTP Verb: GET - Path: /session/{sessionId}/sessionState

The Session state (WAITING, WAITOUT or RUNNING).

Return Type: SessionStateDescriptor

The session state and the SessionDescriptor, relevant information about the session. (See Appendix to get more details)

Parameters:

- sessionId.

getCurrentAction

HTTP Verb: GET - Path: /session/{sessionId}/action/output

Description about a session waiting for Output parameters, and the parameter´s descriptor and values.

Return Type: ActionInstanceDescriptor

See Appendix to get more details about ActionInstanceDescriptor.

Parameters:

- sessionId.

getAdvisedActions

HTTP Verb: GET - Path: /session/{sessionId}/ advisedActions

Recommended, Allowed and Excluded actions, for present and passed tests. It is possible to retrieve the advised actions for the actual action either for a specified executed action.

Return Type: NextActionsDescriptor

Return three different lists for the recommended, allowed and excluded actions. Each of them are lists of RawActionDescriptor (See Appendix to get more details).

Parameters:

- sessionId.
- history and index: Set ‘history’ to true, and get a certain advisedActions from the history or the next one (depending on the index).

getHistoryAdvisedActions

HTTP Verb: GET - Path: /session/{sessionId}/ historyAdvisedActions

The complete history of advised action for each action in a session.

Return Type: List<NextActionsDescriptor>

All the NextActionDescriptors (see getAdvisedActions), for each action in the T&D test, presented in a list.

Parameters:

- sessionId.

getHistoryActions

HTTP Verb: GET - Path: /session/{sessionId}/historyActions

Get information about executed actions in the current session.

Return Type: ActionHistoryDescriptor

Count: Number of different actions executed.

Actions: List of all the executed actions in the same order returned as ActionInstanceDescriptor.

condensedActions: List of the actions that have been executed at least once, as ActionInstanceDescriptor.

See Appendix to get more details about ActionInstaceDescriptor

Parameters:

- sessionId.

getStickyData

HTTP Verb: GET - Path: /session/{sessionId}/ stickyData

Information associated to tables (complex types).

Return Type: Map<String, Object>[]

The sticky information formatted in a way that can be read by the final user. The descriptor contains a list of pair name-value for each element that has been defined as sticky-data.

Parameters:

- sessionId.

instantiateAction

HTTP Verb: GET - Path: /sessions{sessionId}/ action/input

Instantiates an action. This step is previous to the executeAction procedure. (See Appendix to get more details)

Return Type: ActionInstanceDescriptor

The Session state and relevant information about the session and the action.

Parameters:

- sessionId.
- actionName and solutionName: Identify the action to instantiate.
- backtrack and index: Set backtrack to true will create a new Action that will override another action in a certain position (index).

executeAction

HTTP Verb: POST - Path: /session{sessionId}/executeAction

Executes the action, and passes the input parameters (if they existed).

Return Type: SessionIdDescriptor

Returns the status code 200 if the action has been successfully executed.

Parameters:

- sessionId.
- ExecuteActionRequest: It includes :
  - input: Parameter´s name and value listed as Map.
  - actionInstanceDescriptor: Returned by the previous instantiateAction step. (See appendix to get more details)
  - backtrack and index: Set backtrack to true will create a new Action that will override another action in a certain position (index).

retryAction

HTTP Verb: PUT - Path: /session/{sessionId}/action

Used to notify to a session waiting that the user has decided to rerun the last run action to obtain a different result, instead of populating the output parameters for it.

Return Type: SessionIdDescriptor

Returns the status code 200 if the action has been successfully retried, and also the sessionId

Parameters:

- sessionId.

endAction

HTTP Verb: DELETE - Path: /session/{sessionId}/action

Populates the output parameters and notifies it to the action waiting in the second queue.

Return Type: SessionIdDescriptor

Returns the status code 200 if the action has been successfully ended.

Parameters:

- request: It is a JSON request that includes:
  - outpuParameters: Parameter´s name and value listed as Map.
  - sessionId.

setSessionNote

HTTP Verb: POST - Path: /session/{sessionId}/note

Sets a note to the session.

Return Type: SessionIdDescriptor

Returns the status code 200 if the session´s note has been successfully set.

Parameters:

- sessionId.
- NoteRequest: It includes the note content.

getSessionNote

HTTP Verb: GET - Path: /session/{sessionId}/note

Sets a note to the session.

Return Type: NotesDescriptor

Returns a descriptor with the note´s contents.

Parameters:

- sessionId.

getServiceValidationActions

HTTP Verb: GET - Path: /serviceValidationActions/list

Return a list of available Service Validation actions.

Return Type: ActionDescriptor[]

Returns a list containing the actions marked as Service Validation.

Parameters:

- None.

searchServiceValidationActions

HTTP Verb: GET - Path: /serviceValidationActions

Returns a list of the TD sessions, it means that looks for the jobs associated with the TD_ExecuteServiceValidationAction, and also allows a filtering on customer, operator, and date. It also supports pagination.

The underlying logic is based in a private method which is the one that search the sessions that fulfill the filter, the pagination, and besides, accept as a parameter the corresponding workflow. (In this case, TD_ExecuteServiceValidationAction.)

Return Type: SessionsDescriptor

results: an array of the SessionDescriptor retrieved. (See Appendix to get more details)

count: a field that indicates the number of sessions retrieved.

Parameters:

- operatorName: String. Name of the operator.
- sessionId: String. Retrieves only the session indicated.
- fromDate and toDate: Filter start time. Valid formats are:
  - yyyy-MM-dd HH:mm:ss.SSS
  - yyyy-MM-dd HH:mm:ss.SSS Z
  - yyyy-MM-dd HH:mm:ss
  - yyyy-MM-dd HH:mm:ss Z
- numberOfResults: String. Results per page.

instantiateServiceValidationActions

HTTP Verb: GET - Path: /serviceValidation/input

Instantiates an action. This step is previous to the executeServiceValidationAction procedure

Return Type: ActionInstanceDescriptor

The Session state and relevant information about the session and the action.

Parameters:

- actionName.
- solutionName.

executeServiceValidationActions

HTTP Verb: POST - Path: /serviceValidationAction

Executes the service validation action, and passes the input parameters (if they existed).

Return Type: SessionIdDescriptor

Returns the status code 200 if the action has been successfully retried, and the sessionId

Parameters:

- request: : It is a JSON request that includes:
  - inpuParameters: Parameter´s name and value listed as Map .
  - actionName and solutionName: The action to be executed.
  - language and operator.

Frequently used JSON descriptors

SessionIdDescriptor:

Returns the status code 200 if the operation has been successful, and also the sessionId.

- Status.
- sessionId.

SessionDescriptor:

Details regarding the session, and a brief description of the action running.

- operatorName
- customerName
- language
- creationDate
- serviceId
- serviceName
- updateDate
- majorCode
- minorCode
- majorCodeDescription
- minorCodeDescription
- action: see RawActionDescriptor

ActionInstanceDescriptor:

Description of the instance running, and an extended description of the running action.

- state
- taskStatus
- note
- diagnostics
- majorCode
- majorDescription

- minorCode
- minorDescription
- action: see ActionDescriptor

RawActionDescriptor:

Brief description of the action running, no references to input/output parameters.

- solution
- name
- description
- label

ActionDescriptor:

Extended description of the action. Takes the rawActionDescriptor as starting point and add more information, including the input and output parameters.

- actionMode
- averageDurationTime
- icon
- throbber
- inputGroups
- helpText
- isServiceValidation
- cost
- type
- executionNode
- inputTitle
- outputTitle
- inputParameters: List of ParameterDescriptor. (See below)
- outputParameters: List of ParameterDescriptor. (See below)

ParameterDescriptor:

- name
- type
- label
- historyLabel
- description
- value: Object
- defaultValue
- editable
- required
- displayType
- displayFormat
- mandatory
- helpTexts: HelpTextDescriptor[]
- icon
- setAsCustomer
- setAsCause
- setAsProblem
- setAsServiceType