OPENCM USER GUIDE

WEBMETHODS CONFIGURATION MANAGEMENT

Version 1.8.5 | 21.09.2018

VERSION

Version	Date	Author	Description
1.8.2	30.08.2018	Håkan Hansson	First public release, relating to implementation v1.8.2
1.8.3	03.09.2018	Håkan Hansson	Updated third party jars (latest versions)
1.8.4	05.09.2018	Håkan Hansson	Integrated with Keepass
1.8.5	21.09.2018	Håkan Hansson	Introducing 2-way synchronization, Command Central nodes management

TABLE OF CONTENTS

1.0 Introduction

1.1. Overview

The objective of this document is to describe the implementation of the OpenCM IS package and its context in which it operates. The OpenCM implementation is an open source project, located on https://github.com/SoftwareAG/OpenCM. Contributions to the code base are highly encouraged and welcomed.

The practice of Configuration Management is in this document covering the notion of storing, managing and operating on top of an off-line repository, which is the storage repository of how an environment, server or product component should be, and currently is, configured. For that reason, the CM practice is optimally involved during the whole life-cycle of the DBP platform components: blueprinting, provisioning and configuration, quality assurance, change management, operation and maintenance, upgrades, decommissioning, etc.

1.2. Audience

This document is primarily intended for stakeholders responsible of managing the OpenCM component itself (developer, user) but also others who have an interest in the overall governance procedures of configuration properties management (of webMethods Digital Business Platform components).

The first part of the document provides a high-level overview of the CM strategy to provide a context for where OpenCM fits in. The latter half of the document provides step by step instructions on how to install, configure and use OpenCM.

1.3. Terms & Abbreviations

- **Configuration Management**. Abbreviated in this document as “CM” and refers to the overall practise of managing configurable information.
- **Source of Truth.** Abbreviated as “SoT” throughout the document and refers to a repository of information which defines configuration items and their corresponding values as they are meant to be. This is also referred to as the baseline repository within OpenCM.
- **Baseline Repository.** See above “Source of Truth”.
- **Runtime Repository.** This is the storage area within OpenCM that reflects the runtime installation information. This storage area is meant to be continuously refreshed by extracted data.
- **Configuration Item.** Abbreviated as “CI” and not be confused with “Continuous Integration”. CI’s refer to individual properties that are defined within the CM repository.

1.4. Summary Current Capabilities

The OpenCM implementation currently supports the following capabilities:

- **Runtime Extraction** – Connect and retrieve information from runtime installations based on all the information that is provided by the locally installed Platform Manager. In addition: Integration Server package information, JCE Policy Information, JDBC Adapter connection and SAP Adapter connection information.
- **2-Node Auditing** – Ability to compare any configuration property between different installations (comparing node 01 against node 02).
- **Baseline vs Runtime Auditing** – ability to compare differences between the source of truth and the current runtime extraction data for a single node.

- **Default Auditing** – ability to compare differences between extracted data and their corresponding default values (as defined by the webMethods product when first installed).
- **Layered Auditing** – Ability to generate a report (Excel) that outlines all the values in all environments based on a single property. This is used to get a more comprehensive view on the differences across environments.
- **Command Central Node Definitions** – can generate a CLI script that defines all the installation nodes within the different environments (for easier and quicker CCE setup).

Having all the configuration properties and their associated value in a single place also provides the ability to quickly and easily inspect/review settings via the OpenCM user interface.

2.0 Introducing a CM Strategy

The primary reason for defining and implementing a CM strategy comes from the fact that business service quality is degraded without one. If the configuration properties of the runtime environments are not adequately documented, nor ensured, the unavoidable consequence is (undeliberate) differences or incorrectness in the setup of the platform within the various installations.

The objective of a CM strategy is to define an implementation approach that is avoiding the following (common) pitfalls:

- There are often/always discrepancies in terms of configuration settings between different runtime nodes and between different environments (where they actually should have been equal)
- There are often discrepancies between runtime environments and documentation
- The configuration information (what should be) is stale or completely absent
- Manual environment audits are time-consuming and cumbersome
- The same information is stored in multiple locations, thus needs to be updated in multiple places
- No clear understanding of wrong vs. right configuration property: i.e. there is no single source of truth
- Activities around installations, configurations and deployments are time consuming and error prone
- Quality Assurance is invalidated due to misconfigured environments

1.5. Configuration Repository

Central to the CM strategy is governance of configuration information and the level of control is then dictated by the ability to manage CI items in a separate location (from the runtime itself). What type of configuration repository to use is discussed here:

1.5.1. None

Not all organisations document the properties or settings that have been defined, updated and applied to their DBP installations. The obvious consequence is that it is not possible to know what configurations are correct or incorrect. Whatever is defined in the runtime environment is the “source of truth”. A loss of server information (deliberate or undeliberate) will be hard to reproduce and the ability to provision new environments is equally difficult.

Inspection of a configuration setting in an environment and ensuring that it is correctly defined can only be done by comparing with another environment and by applying some level of subjective sanity check. Even then, it is not possible to know what the value of the property should be.

2.1.1 Static Repositories

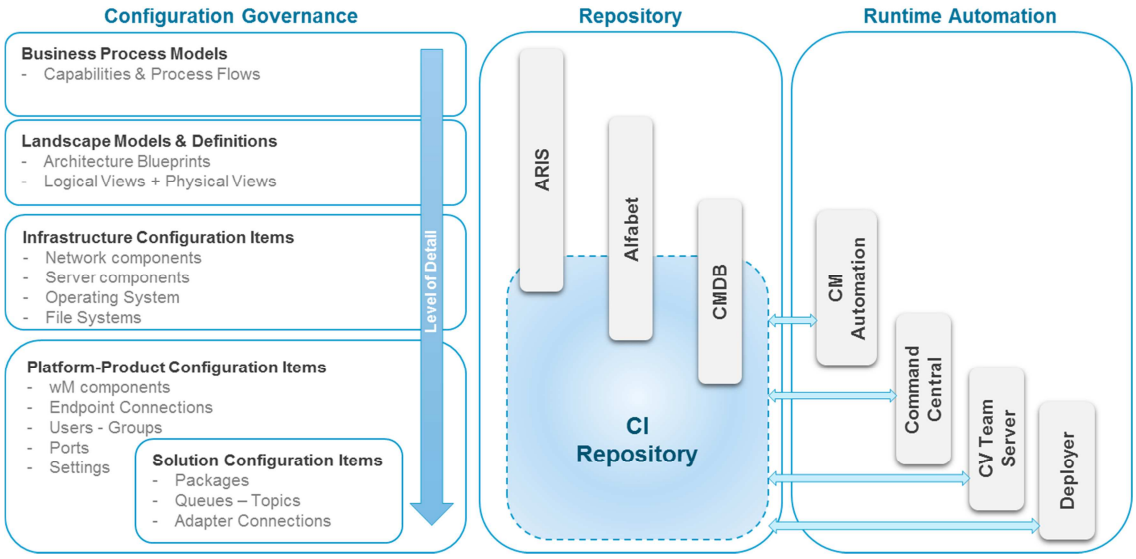
Somewhat more mature organisations will document the settings, the configurations and the properties of the DBP environments and will most likely use a more static repository such as Word documents, Visio diagrams, Excel spreadsheets or a Wiki-type location. The problems with static repositories are:

- The repository will inevitably become out of synch with the environment over time. Even the most rigorous governance process will still prompt a manual, time-consuming environment audit to ensure that there are no misconfigurations.
- It is difficult to represent the environment using static repositories for different purposes. Architects, operations team, maintenance team, developers, change management, etc. are all “users” of the repository and each team wants to view the information from different angles. This often leads to having the same configuration properties/items defined in

multiple locations, after having been extracted out and copied to other places. A change of configuration then leads to having to perform updates in multiple locations (which again leads to discrepancies between multiple repositories).

2.1.2 Repositories vs Automation

Within the CM practice, it is important to distinguish between runtime automation tools (which are often classified as “Configuration Management tools”) and the support systems that can operate on top of a store that holds configuration information.



- Illustrated on the left side of above diagram, shows the level of granularity for “configuration governance”. On a very high level, the business functionality is defined as a set of process models, thus capturing the underlying IT requirements. The implementation of the business requirements are then supported by various architectural views and prompts an infrastructure setup with webMethods installations and solution deployments.

- On the right side of the diagram, there are a number of different automation tools:
  - CM Automation is a collective group of commercial/open source tools such as Puppet, Chef, SaltStack, Ansible, etc.
  - Command Central is the SoftwareAG tool for perfomring installations, applying fixes and managing of various configuration settings in the runtime environment.
  - Cross Vista team Server and SoftwareAG Deployer are tools for automating the deployment efforts of individual solution implementations.

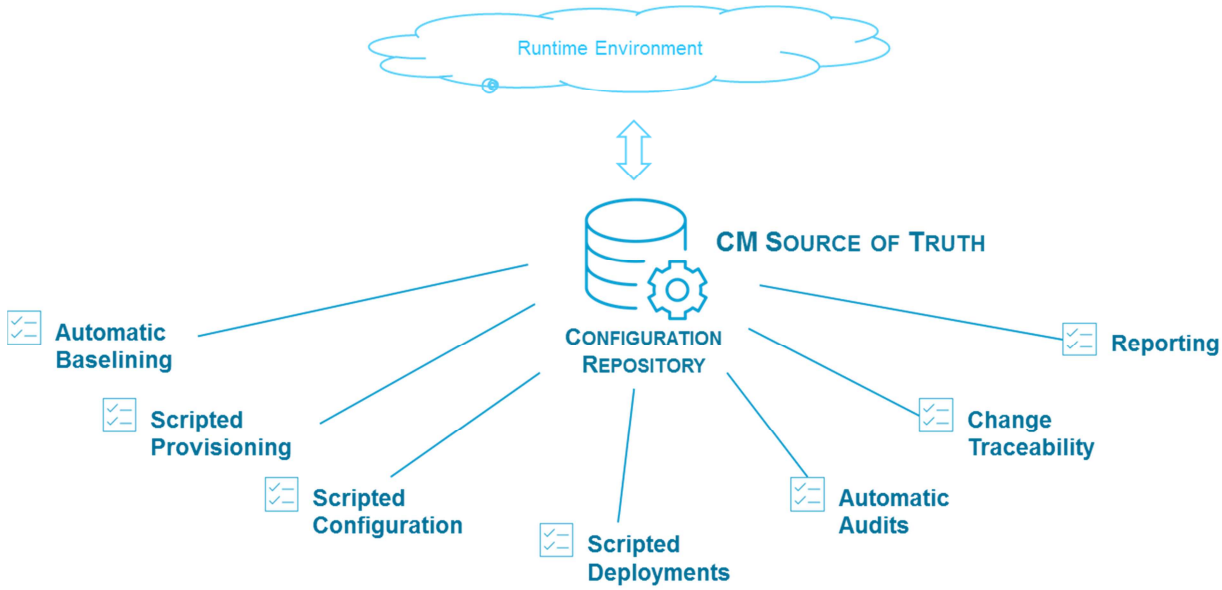
- In the middle of the diagram, are identified (potential) respositories of configuration information:
  - SoftwareAG ARIS provides a repository for the DBP blueprints and the physical architectural views can be extended to cover finer granularity of the assets. However, the volume of data that is stored in a CM implementation is far too great. It is estimated that between 500-1000 properties per server installation would need to be captured. The ability to model and define configuration items is inherent but not adequate for storing vast amount of configuration information.

- SoftwareAG Alfabet is also a repository can potentially could serve as a configuration management repository but again is not fit for purpose when it comes to more granular (and vast amount of) configuration items.
- Commercial CMDB implementations (e.g. BMC Atrium) are used to manage large landscapes and the level of details required for a CM implementation is again not fit for purpose. The granularity of CI items is often on the service/application level, and not on an individual product configuration property level.

At the bottom of the middle section of the diagram, is the required space to fill for the purpose of the CM Strategy and no commercial/open source utility has been identified. This is the space which OpenCM intends to fill.

2.2 CM Requirements

In line with the need to provide a CM repository, other requirements haven been identified to provide a complete CM implementation to serve the overall strategy:



It should be noted that the current version of OpenCM fulfils the automated baselining of configuration properties and the continuous auditing requirements.

2.2.1 Automatic Baselining

The initial setup or configuration information into the SoT must not be dependent on a manual definition of a single configuration property; it would take a huge amount of time. The population of the repository should both be an automatic activity (snapshot from runtime environment) as well as an option to “model” the properties for further provisioning/configuration activities.

2.2.2 Support Scripted Provisioning

Installing a new environment or setting up a new server can be highly automated by scripting (with additional Command Central) support. However, the configuration information used by the provisioning process must come from the SoT and not be duplicated.

2.2.3 Support Scripted Configurations

Post installation activities that cover changing component configurations, adding users, ports, etc. must be supported by the CM strategy and subsequently be stored in the SoT. As with Provisioning activities, the scripts and helper properties files must not be duplicating the information from the SoT and stored in multiple locations.

2.2.4 Support Scripted Deployments

The development process that incorporates automated build and deployment (in a continuous integration context) must continue to be supported. However, the configuration items included in today's change process must be covered by the SoT and thus integrated with the CM strategy.

2.2.5 Support Automated Audits

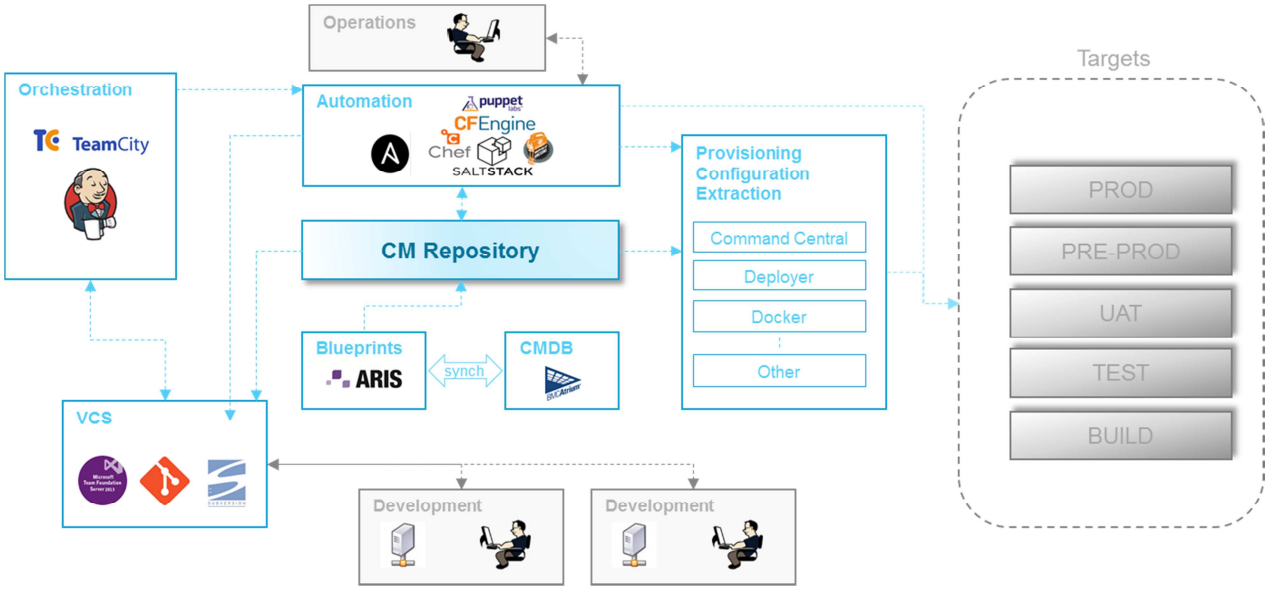
The CM strategy must support the ability to perform regular, automated environment audits as per content of the SoT. There could be various use cases and scope for the auditing, but the requirement is that the implementation should manage to perform retrieval of runtime information (in various environments) and compare settings with SoT. In addition, comparison between various entities within the SoT should also be possible (e.g. comparing two cluster nodes with each other in the SoT).

2.2.6 Support Change Traceability and Reporting

The CM strategy must incorporate a mechanism to find out or understand who changed what and when. I.e. there must be some sort of traceability of changes performed on individual CI's. In addition, the SoT must be reportable, meaning it should be possible to extract information about collections of CI's (based on criteria) or a single CI property.

2.3 CM in a Change Context

The CM repository should be looked at from a change context and perhaps best represented by the following diagram:



- From a Continuous Integration perspective, the CM Repository (and its functions) sits in between the orchestration component (e.g. Jenkins) and the automation tools that provide the ability to deploy solutions to the target environments. Key is that required configuration information required to deploy is extracted from the CM repository and not separately managed by the orchestration scripts.
- Similarly, operational tasks that involve provisioning or configuration changes are also based on the information located within the CM repository.

Change activities may first initiate an update to the CM repository before applying the same change to the actual runtime environment, - if and when a change to the source of truth is required.

Although OpenCM implementation is here placed in a context, its functionality is not dependent on surrounding components except for Command Central SPM's (which are used for extraction of configuration items and the ability to perform continuous audits).

3.0 Installation and Prerequisites

OpenCM comes as an Integration Server package and should preferably be deployed to an "Administration" type server, separate/different from a business runtime IS.

3.1 Summary

The following steps are required to set up OpenCM and perform auditing:

1. Install OpenCM package on the Integration Server (this section)
2. Define the nodes (see section 4.1.3)
3. Define what to extract (see section 4.2.2)
4. Define what to audit (see section 4.4 and 4.5)

3.2 Compatibility

OpenCM has been developed and tested on webMethods version 9.9, 9.12, 10.0 and 10.1.

The ability to extract and compare target runtime versions is based on the respective SPM/SPM plugin capabilities. For example, certain information extracted from a v9.0 installation is not available compared to a 10.1 installation.

3.3 Download

OpenCM package is available to download from SoftwareAG GitHub Repository

3.4 Dependencies & Requirements

- ✓ OpenCM is reliant on webMethods Command Central Platform Managers (SPM's) to extract and update runtime configuration properties.
- ✓ OpenCM also utilizes file system storage for property configuration information: allocate a separate folder/directory on a file system that can be accessed by OpenCM. Estimate 3-4 Mb of disk space for each managed server in the landscape.

3.5 Installation

1. Install OpenCM on an Integration Server using the regular package installation procedures.
2. Create a separate "root" directory on the file system for wxcm specific configuration files and storage of extracted property files. E.g.:
    - OpenCM directory = F:\SoftwareAG\opencm
3. Under the above OpenCM directory, extract the template zip file from the OpenCM/templates/opencm-template-config.zip.
4. Set up a soft link from the /pub folder of the OpenCM package, pointing to the above cmdata and output directories. This is needed to ensure that the OpenCM UI can access the configuration data for visualization and report output.
    - Change current directory to <is_instance>/packages/OpenCM/pub

      Windows:

      >> mklink /D cmdata F:\SoftwareAG\opencm\cmdata

      >> mklink /D output F:\SoftwareAG\opencm\output

      Linux:

      >> ln -s F:\SoftwareAG\ opencm\cmdata .

>> ln -s F:\SoftwareAG\ opencm\output .

3.6 Configuration Tasks

1. Add the following Global Variable:
   - Name: **WXCM_MASTER_PASSWORD**
     - Purpose: The master password used to encrypt and decrypt access information
     - Example Values (type = password):
       - manage
2. Update the **default.properties** located in the package config folder with appropriate target directory locations. The default.properties simply points to a “main” opencm.properties file, located under the central config directory (away from the package directory).

   The purpose is to be able to deploy new versions of the OpenCM package without overwriting existing configuration files.
3. Update the **opencm.properties** located in the main opencm config folder with appropriate target directory locations. Also:
   - *endpoint_config* refers to the type of endpoint definitions that are used. Refer to section 4.1.3 for more information.
   - *cce_mgmt_node* refers to the Command Central node used, and applies to Command Central management functions for defining environments and installation nodes directly in Command Central UI. *cce_mgmt_enforce_extraction_node* is also used for above functionality. Refer to section 4.6 for additiona information.
   - *local_opencm_node*, *target_opencm_node* and *ftps_timeout_ms* elements refer to synchronization functionality. Refer to section 4.7 for more information.

1.5.2. Adjust as per local Environment

To set up OpenCM to reflect the current environment, perform the following tasks:

1. Update the nodes.properties as per section 4.1.3
2. For Extraction of runtime data: update extract.properties as per 4.2.2
3. For Comparing nodes (two-node audit): update two-node properties as per 4.4
4. For Comparing environments (layered audit): update layered audit properties as per 4.5

4.0 Implementation

The following section outlines the key parts of (current version of) the OpenCM implementation.

4.1 Visualization

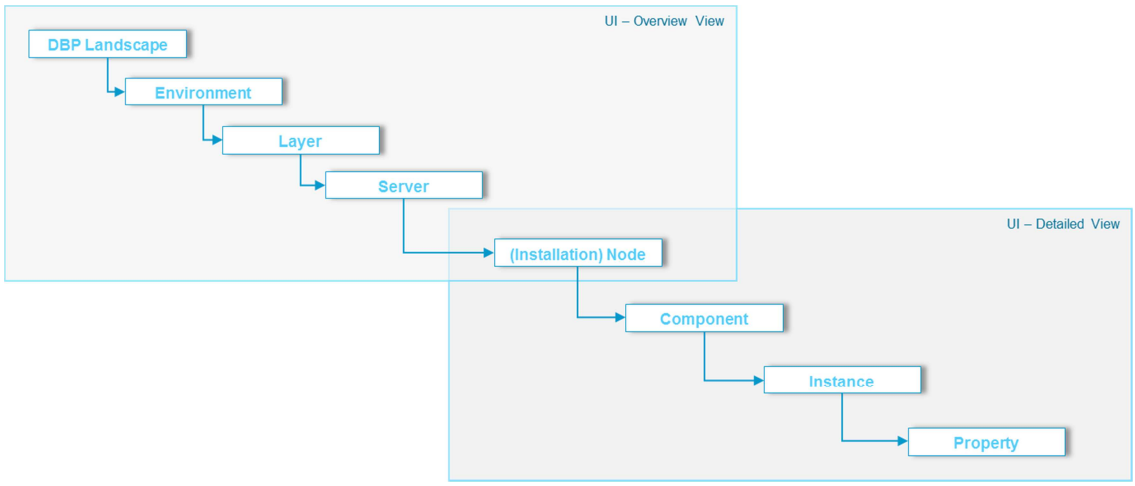
Visualization covers the OpenCM home page and the ability to view the baseline and runtime snaphot information. It also covers the ability to invoke certain functions from the home page via menues.

The dynamic tree of environments, servers, nodes, etc. that are displayed on the main center section is based on d3 tree visualizations (https://d3js.org).

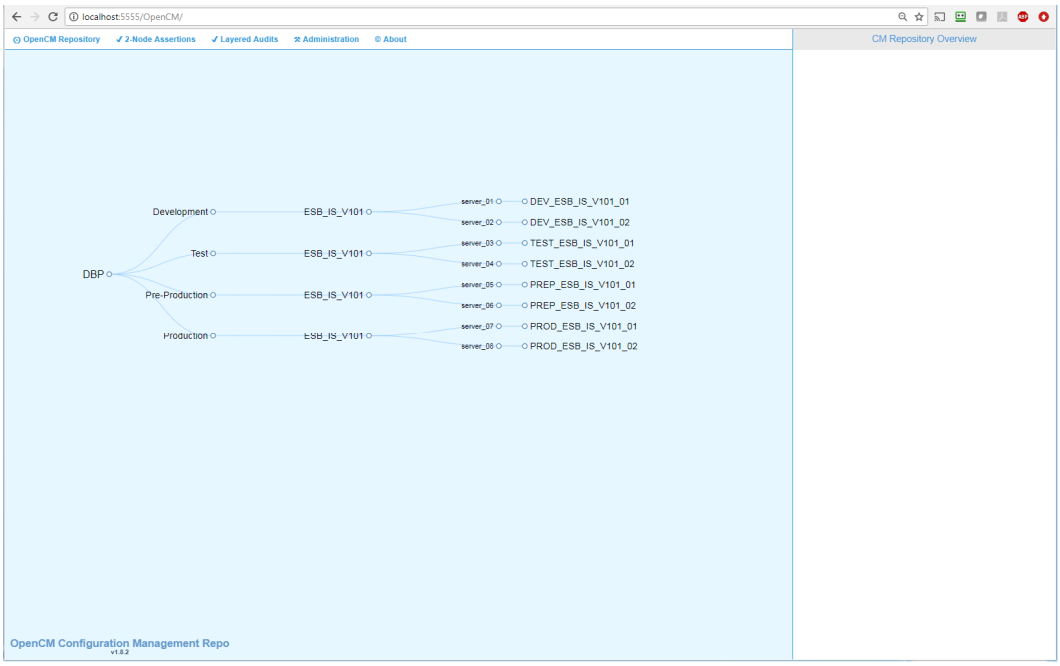
There are two separate views:

- **Overview** shows the complete tree structure of installations within the integration platform.
- **Detailed** view shows the individual installation (node).

The structure is as follows:



Overview: showing the landscape and relationships between environments, servers and installations:

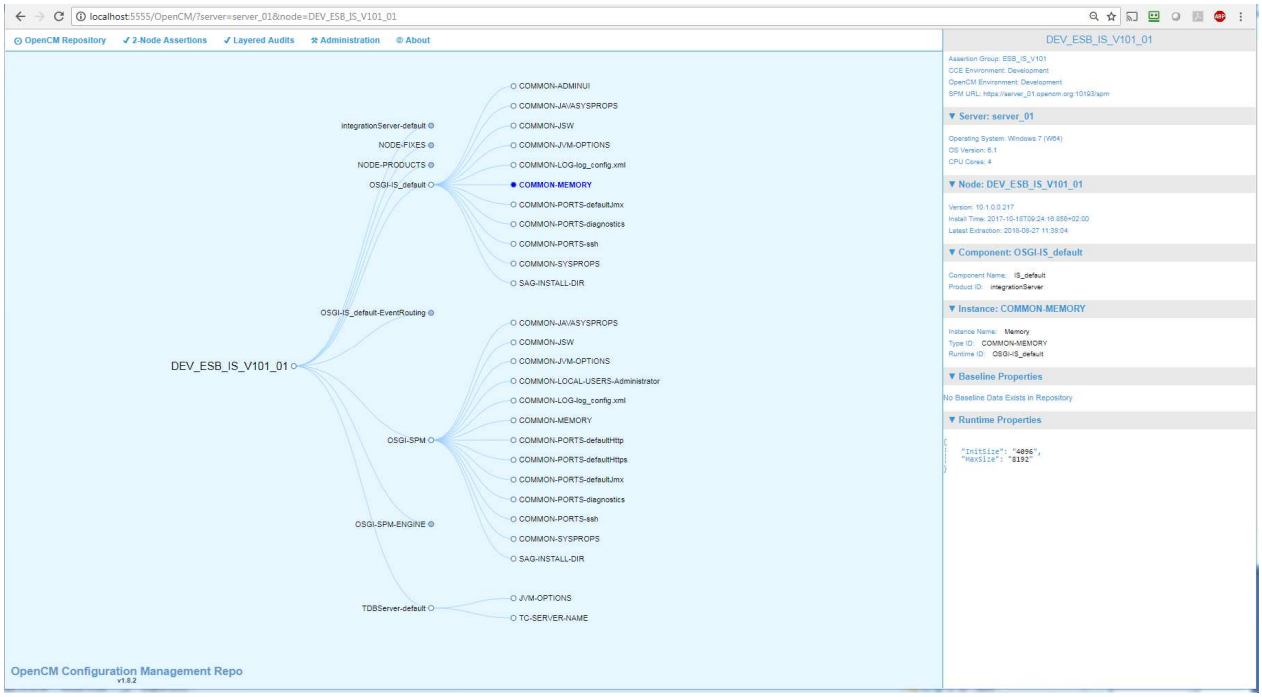


The structure of the overview is based solely on the nodes.properties, - a central configuration file describing the different servers and installations in the various environments. I.e. it is not required

to perform runtime extractions in order to view this page.The nodes.properties will be described in a later section.

Clicking on the node name will show the individual installation details as below.

Installation: showing the detailed information about node properties:



Above screenshot shows an example view of a single installation, with the node – component – instance representation on the main center canvas.

The right-side section contains detailed information of a single instance, with multiple properties shown on the lower right-side corner. The property information is displayed in json format, same as the actual file storage format.

There are two separate sections displaying property information:

- **Baseline** properties indicate that the node has been “baselined” and would constitute the “source of truth”.
- **Runtime** properties indicate that there is a snapshot extraction taken from the actual runtime installation.

4.1.1 UI Folder structure

- **/pub**

  Root directory of the home page. The main entry to the home page is within the index.dsp file.

- **/pub/css**

  Contains css style sheets. Key style sheet is the “wxcm.css” file.

- **/pub/dndTree**

  Contains d3 json files for visualization.

- **/pub/js**

Holds the java script files. The main, custom scripts reside in the wxcm.js file, whilst the other ones are part of the component that is used to visualize:

- d3.js (for the d3 tree visualization)
- jquery*.js (for jquery functionality)

- **/pub/img**

  Contains the few icons used on the OpenCM home page

- **/pub/output**

  This is the directory (symbolic link) where html and excel report files are stored during processing and picked up for display.

- **/pub/cmdata**

  The cmdata directory (symbolic link) is the area where retrieved extraction data is stored.

1.5.2.1. Main Service

To generate the dndTree json files for visualization, the main service is OpenCM.priv.visualization:generate and can be triggered from the frontend page, under the „Configuration“ – „Refresh Tree“ menu.

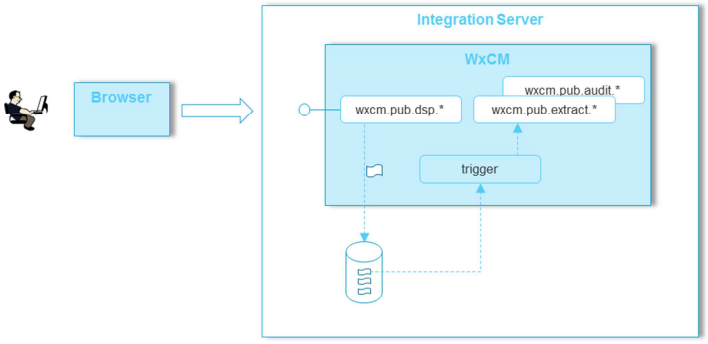
The service then performs two separate steps:

1. Based on “nodes.properties”, generate the overview tree structure
2. For each node, create a separate tree structure for baseline (if exists) and runtime (if exists).

Each tree structure will be stored under the pub/dndTree folder within the package folder.

4.1.2 Integration to Backend

In order to submit requests into the local Integration Server, the browser submits a URL „get“ from within the main java script wxcm.js, targetting the pub/dsp/* package services. The services are simple flow services, receiving the http request and performs a subsequent local message invocation via JMS (send), to allow for a asynhronous browser request.



4.1.3 Endpoints

In order to perform extractions, visualize the DBP integration landscape in the UI, perform audits, etc., information about available nodes and endpoints is needed. There are currently two ways to define the landscape nodes and the endpoint information:

1. OpenCM internal – using a “nodes.properties” file describing all the installation nodes
2. Keepass

Nodes.properties

This is the main, central property file for identifying the nodes used for both extractions and auditing activities. A node “runtime component” is a runtime entity (e.g. a Platform Manager or an Integration Server) and identifies the attributes required (such as server name and port). There might be multiple instances per installation node.

```
nodes:
  - node_name: DEV_ESB_IS_V912_01
    environment: Development
    hostname: server_01.opencm.org
    assertion_group: ESB_IS_V912
    runtimeComponents:
      - name: OSGI-SPM
        protocol: http
        port: 8092
        username: Administrator
        password: manage
      - name: integrationServer-default
        protocol: http
        port: 5555
        username: Administrator
        password: manage
      - name: integrationServer-instance_02
        protocol: https
        port: 5443
        username: Administrator
        password: manage
```

The above example shows a single node (installation) with 3 separate runtime components:

- An SPM
- An Integration Server instance named “default”
- Another Integration Server instance named “instance_02”

The idea is that each node contains relevant information for not only extracting the information (therefore the endpoint information) but also to define its properties as to where it resides and to what assertion group it belongs to. Some clarifications:

- **cce_environment:** defines the Command Central environment in which the node is located. This property is not used during extractions, nore for auditing purposes but used as a helper function for generating CCE CLI commands to define installations.
- **opencm_environment:** defines what environment the node belongs to within OpenCM. For example, it is possible that the node is grouped under the „DEV_TEAM_01“ environment within Command Central but is really just a „Development“ node grouping that can be used within visualization (UI) or layered audits.
- **assertion_group:** defines the layered group of nodes that it belongs to for the purpose of detecting differences amongst a set of installations. This is also used during „layered audits“.

Passwords

Storing passwords in clear text on the file system is not desirable however, and OpenCM will encrypt the password parameter values before extraction (if needed). It is therefore possible to edit the endpoint configuration file prior to running the extract service and specifying a clear text password anywhere in the file. After decryption is performed, the endpoint property file will look as follows (e.g.):

```
nodes:
  - node_name: DEV_ESB_IS_V912_01
    environment: Development
    hostname: server_01.opencm.org
    assertion_group: ESB_IS_V912
    runtimeComponents:
      - name: OSGI-SPM
        protocol: http
```



```
    port: 8092
    username: Administrator
    password: "[ENCRYPTED::zXiy/HYOa2iG2fieiGtweA==:elu/Z8d3+8CS+6y6McjrXA==]"
  - name: integrationServer-default
    protocol: http
    port: 5555
    username: Administrator
    password: "[ENCRYPTED::JQ8tlrZU/ECgwDDo8VVNSQ==:VdONpTwZTbbBKhuHYaE0Dw==]"
```

The passwords are encypted based on the javax.crypto libraries and driven by a master password. The master password is defined in the Global Variables section of the Integration Server (refer to the installation section).

To support encyption and decryption, there are two services defined under the priv.security folder of the OpenCM package:

- OpenCM.priv.security:decryptPropertyFile
- OpenCM.priv.security:encryptPropertyFile

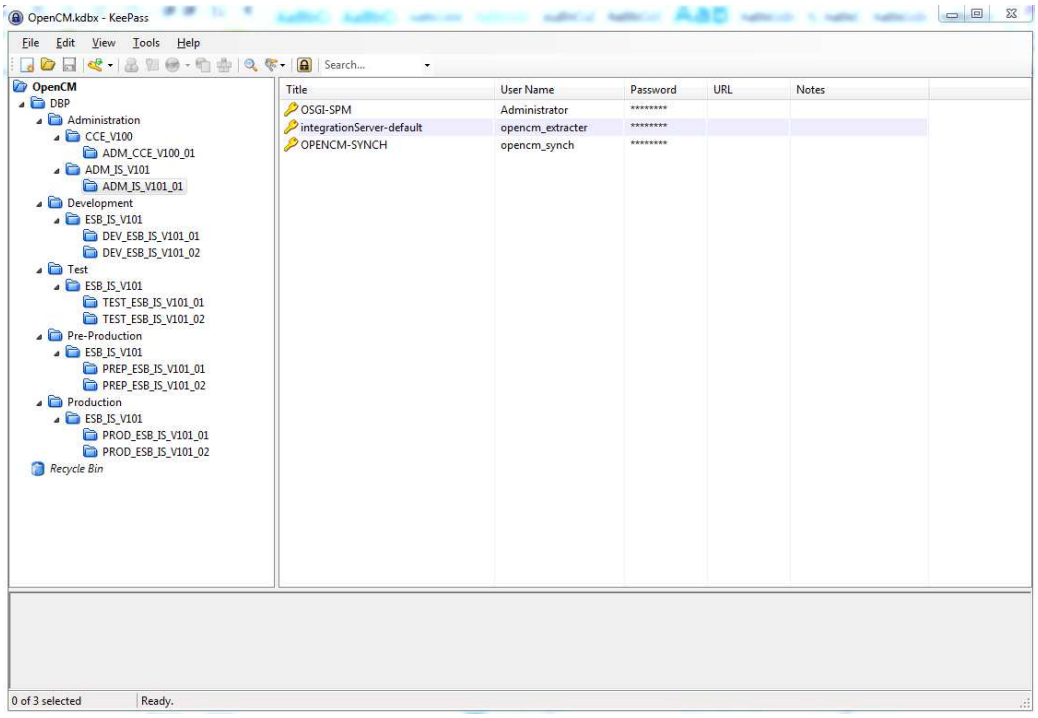
These services can also be invoked from the OpenCM home page, under the “Configuration” menu.

Keepass

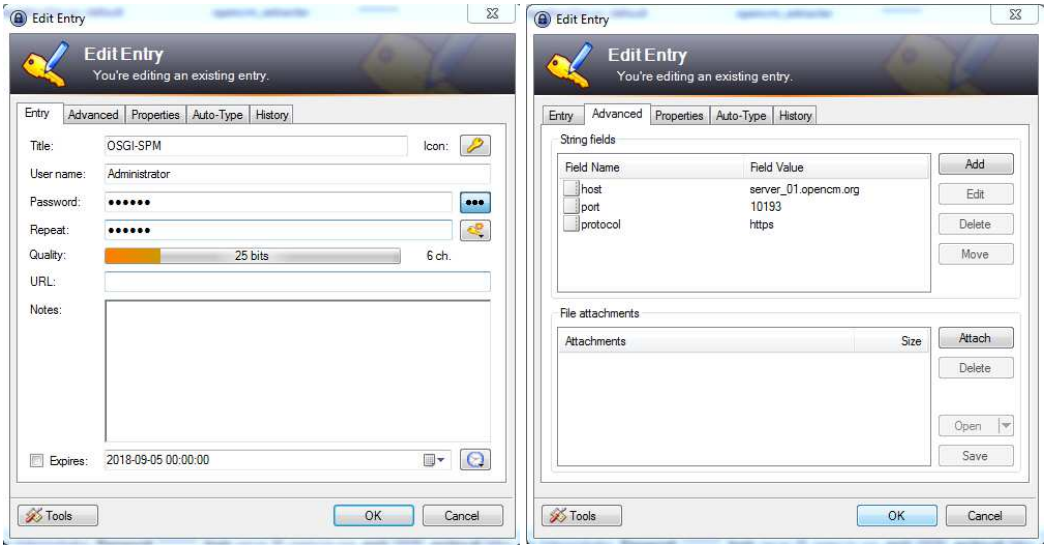
Another way to define the installation nodes is by using Keepass as the holder (or database) of the information. There are some important notes about this approach:

1. The Keepass database file must be accessible by the OpenCM
2. The Keepass groups define the meta-model structure of the property information. I.e. the first level after the main, top, group are refered to as environment names. The second level is the layer and the third level is the installation node itself.
3. The Keepass entries under the node level, are considered runtime components. Each entry includes username and password, and in addition the following key-value pairs:
    a. host
    b. transport
    c. port

    In essence, the Keepass structure (groups + entries) contain the same information as the above nodes.properties.



The following shows the Keepass entry for the SPM component:



It also means that all groups under the top group will be parsed by OpenCM to find information. The top group can be any group within the Keepass database, and is references by the “top_group” property in opencm.properties.

4.2 Extraction

Extraction involves retrieving runtime properties from the various nodes in the environments.

4.2.1 Main Service

Main extraction service is OpenCM.pub.runtime:extract and can be triggered from the frontend page, under the „CM Repository“ – „Extract using Properties“ menu.

The process starts by invoking the following main service:

⇨ OpenCM.pub.runtime:extract

The main service then performs two steps:

3. Retrieve the information and store on to the local disk based on “extract.properties”.
4. Update the d3 structure with the new folder structures

Extraction will store runtime information under the “runtime” file structure and if the node information is already present, it will first remove the installation node directory. This means that the runtime storage area will contain the latest extraction information (only).

4.2.2 Extract Properties

Extraction is driven by a properties file that describes what to extract. The file is located under the “extract_config” value from the main wxcm.properties file.

The property file is made up of the follwowing:

```
# ------------------------------------------------------------------------
# OpenCM Property file for driving the snapshot extraction process (yaml notation).
#
# All extractions are based on the definitions in the nodes.properties
#
# ------------------------------------------------------------------------
# Whether to extract all nodes defined or not.
#  true:  Attempt to connect to all defined nodes and their associated SPM instance
#  false: Assumes other criteria below
extractAll: false

# Extract all nodes from a particular OpenCM environment
# The list of environments are inclusive of defined nodes (if there are nodes listed below, they will also be included)
opencm_environments:
#  - Development
#  - Test
#  - Pre-Production
#  - Production

# Extract specific nodes
# The list of nodes are inclusive of defined environments (if there are environments listed above, they will also be included)
nodes:
  - DEV_ESB_IS_V101_01
```

All information retrieved from the runtime environment is based on predefined nodes (in nodes.properties) and the purpose of the above configuration is to be able to determine which nodes to collect information from and how to group the information (based on the OpenCM storage meta-model).

extractAll:

- Takes [true | false] as value
- “true” means that OpenCM will extract information from all nodes defined in the nodes.properties (one by one).
- “false” means that OpenCM will use other criteria below.

opencm_environments:

- Only applicable if the “extractAll” flag is set to false.
- Takes a list of environment names, as per the nodes.properties

nodes:

- Only applicable if the “extractAll” flag is set to false.
- Takes a list of node names, as per the nodes.properties

4.2.3 Retrieving Information

At this point in time, we have information about which nodes to collect information from (based on the extract and the nodes properties) and we also know how to connect to the remote components. Command Center is not involved at all during extractions, instead we collection information from each installation using the remote Platform Manager directly.

There currently two main extraction activities supported in OpenCM:

1. Collect all the information available using the remote SPM as the data collector
2. Collect information directly from a remote Integration Server

Collection via SPM

For each SPM to collect information from, the SPM managed “components” are retrieved using a Platform Manager API request through http. For each of the components, all “instances” are retrieved, again with a Platform Manager API call. And finally for each instance, the “configuration data” is retrieved.

The retrieved data is in the form of json, and stored directly on to disk in line with the OpenCM meta-model in to a directory named:

environment/server/node/component/instance

For each of the component, instance and configuration data enitities retrieved, there is some meta-data information stored on to disk, - separate from the actual property information. This information is used to populate the right-side section of the OpenCM home page, describing information about the properties.

Products & Fixes

In addition to above components/instance/configuration data information retrieved, there is also property information about product components installed and fix levels. This information is separately retrieved from the SPM and stored under the following two component names:

a. NODE-PRODUCTS
b. NODE-FIXES

Data Collection via Integration Server

Information that cannot be extracted using the SPM must be retrieved in other ways. At the moment, the OpenCM package also retrieves the following information:

- **IS - Package Information** - This covers the names of the installed Intregration Server packages, its version and whether the package is enabled or not.
- **IS - JDBC Adapter Connection Information** – If available, then all defined JDBC connection information will be retrieved.
- **IS - SAP Adapter Connection Information** – If available, then all defined SAP connection information will be retrieved.
- **IS - JCE Policy Information** – Whether the JCE unlimited strength policy files have been updated or not.

Other data collection extensions may be developed in the future.

The mechanism to extract IS Package information is to make use of a screen scraping approach, where the OpenCM acts as a browserless client to the IS Administration pages. The supporting libraries are the Selenium HtmlUnitDriver (licensed under the Apache 2)

4.2.4 Retrieving Information from Non-Managed Nodes

It is also possible to extract information in a stand-alone fashion without an IntegrationServer and the OpenCM package. The approach is as follows:

- Export the class files under the code/classes directory into a jar file named “opencm.jar”

- Along with the above opencm.jar file, also copy the existing jar files under the code/jars directory to the target server where the extraction will take place
- Copy the following configuration files to the target server:
    - opencm.properties
    - extract.properties
    - nodes.properties
- Modify the above property files to fit the target server directory structure (depending on where the files are located)
    - The extract properties should then be defined to only access a single installation (or multiple if located on the same server). The node names defined must be represented in the nodes.properties file.
- Save and run a Windows batch script file to invoke the extraction process:

```
@echo off
setLocal EnableDelayedExpansion

SET WXCM_LIB_DIR=F:\openCM\lib
SET WXCM_CONFIG_DIR=F:\openCM\config
SET SAG_INSTALL_DIR=F:\SoftwareAG\webMethods

SET /p OPENCM_MASTER_PASSWORD="Enter OpenCM Master Password: "

set CLASSPATH="
for /R %WXCM_LIB_DIR% %%a in (*.jar) do (
  set CLASSPATH=!CLASSPATH!;%%a
)
for /R %SAG_INSTALL_DIR%\common\lib\ext %%a in (*.jar) do (
  set CLASSPATH=!CLASSPATH!;%%a
)
set CLASSPATH=!CLASSPATH!"
set CLASSPATH=%CLASSPATH%;%SAG_INSTALL_DIR%\common\lib\wm-isclient.jar

call %SAG_INSTALL_DIR%\jvm\jvm\bin\java.exe -Xms128m -Xmx256m -Xnoclassgc
com.softwareag.wxcm.extract.spm.StandAlone %WXCM_CONFIG_DIR%\wxcm.properties %OPENCM_MASTE
R_PASSWORD%

endlocal & set EL=%EL%
exit /b %EL%
```

- The prompted OpenCM Master password is the master password to encrypt/decrypt password values in the nodes.properties file

4.3 Baselining

Baselining in a OpenCM context means that a complete runtime installation node is promoted from being a “runtime extract” to a “source of truth”.

In practice, this means that the folder structures of the cmdata runtime directories are copied over to the cmdata baseline directories and the d3 visualization tree properties are refreshed.

4.3.1 Main Service

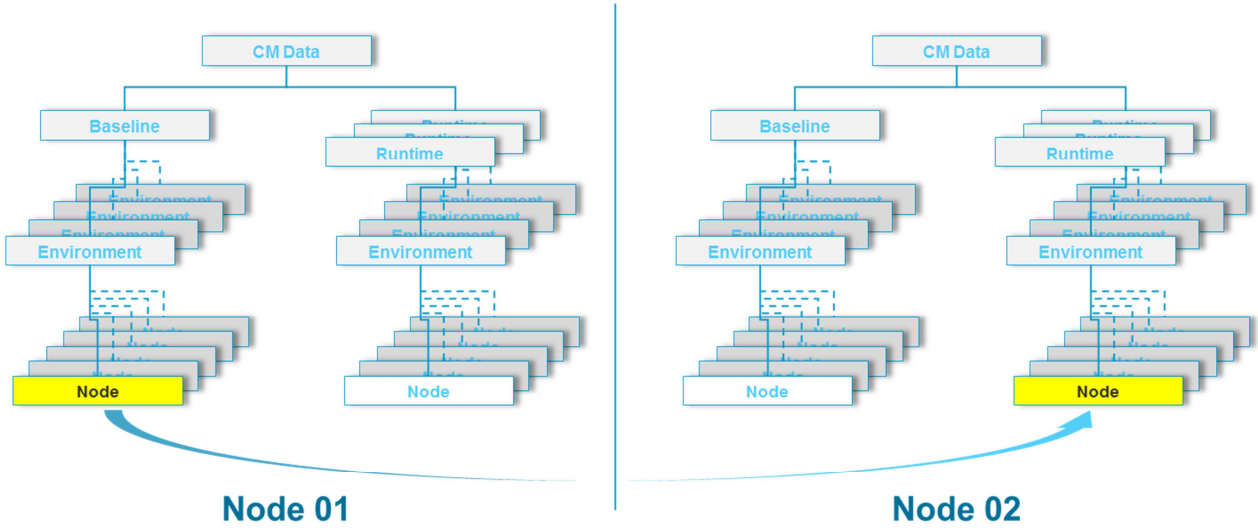
Main baselining service is OpenCM.pub.runtime:promote and can be triggered from the frontend page, under the „CM Repository“ – „Promote Node to Baseline“ menu. This menu item will only be present when accessing a particular node.

It takes one parameter:

- node (what node to promote)

4.4 2-Node Assertions

Two-Node auditing involves comparing extracted data for two separate nodes with one another. I.e. it gives an opportunity to compare any node (snapshot or baseline) with any other node (snapshot or baseline).



The assertions are driven by two configuration property files that defines:

a. What nodes to compare (i.e. node 01 with node 02)

   Note: this property file is common to all 2-node audits

b. What to compare:

   i. Property Key-Values – Compares individual property information with one another.

E.g.:

a. config/audit_two_node/audit_nodes.properties:

```
nodes:
  - node_01:
      repoType: runtime
      nodeAlias: DEV_ESB_IS_V912_01
    node_02:
      repoType: runtime
      nodeAlias: DEV_ESB_IS_V912_02

  - node_01:
      repoType: runtime
      nodeAlias: TEST_ESB_IS_V912_01
    node_02:
      repoType: runtime
      nodeAlias: TEST_ESB_IS_V912_02
```

b. config/audit_two_node/IS-Resources.properties:

```
# ----------------------------------------------------------------------
# Comparing IS Resources
# ----------------------------------------------
testName: "OpenCM 2-Node Audit - IS Resources"
# ----------------------------------------------------------------------
# Properties Assertion Configuration
# ----------------------------------------------------------------------
assertProperties: true
```



```
properties:
  # ------------------------------------------------
  # - Asserting IS Resources
  # ------------------------------------------------
  - component: integrationServer-*
    instance: IS-RESOURCES
    key: /
# ------------------------
# Ignoring Properties
# ------------------------
propertyFilters:
```

All audit property files are located under the <opencm_root>/config/two_node_audit directory.

Description of auditNodes Properties:

- **nodes:**
  - A list of node01 and node02 pairs, that identify which nodes to compare with one another. Each node is outlining the location of the node (according to the OpenCM storage meta-model)

Description of Audit Properties:

- **testName:**
  - This is the name of the test, showing up on the HTML result view (on OpenCM UI)

- **assertProperties:**
  - takes [true | false] value
  - If true, then the defined properties in each node will be compared
- **properties:**
  - This is the definition of what properties to compare. Each property definition is made up of the following three parameters:
    - ⇨ component
    - ⇨ instance
    - ⇨ key

Example - Extended Settings:

```
properties:
  - component: integrationServer-default
    instance: COMMON-SYSPROPS
    key: /
```

The above example is the definition for asserting/verifying that all the Integration Server extended settings are identical, - denoted by the instance name “COMMON-SYSPROPS”.

The key parameter value is “/”, which indicates that all properties under the root json path are examined.

To broaden the search for extended settings for multiple IS instances (not only “default”), it is possible to have a list of component names, e.g.:

```
properties:
  - component: integrationServer-default, integrationServer-instance-01, integrationServer-instance-02
    instance: COMMON-SYSPROPS
    key: /
```


To further broaden the search for extended settings for any component name that holds an instance name of “COMMON-SYSPROPS”, the keyword “ALL” can be used, e.g.:

```
properties:
  - component: ALL
    instance: COMMON-SYSPROPS
    key: /
```

Example - Fix Levels:

Another example of property comparison is when there is a fixed component name, such as fix levels. The component is named “NODE-FIXES” and underneath the component directory is a mutlitude of instance directories, each holding information about a particular fix installation. To define this, the keyword “ANY” weould be used for instance name, e.g.:

```
properties:
  - component: NODE-FIXES
    instance: ANY
    key: /version
```

In above example, the key value is “/version” which indicates that only the version property of the json file should be examined. There are other properties that are thus ignored.

Example – IS Packages:

A property comparison of Integration Server package information is done by not only specifying the key parameter but the associated value that should be examined. The purpose is to be able to define what value constitutes the key and what value constitutes the key value. E.g.:

```
properties:
  - component: ANY
    instance: IS-PACKAGES
    key: /[packageName,version]
```

Component = ANY indicates that any component (IS instance) will be searched. Other components such as a Terracotta Server or an OSGI-default instance do not have an instance underneath with the name “IS-PACKAGES” so it is safe to use the “ALL” keyword.

The Key parameter = /[packageName,version], indicates that the “key value” should be derived from the json file (in this case the name of the IS package) and the associate version key should be used to identify the version number of the IS package. The json file looks like the following, e.g.:

```
[{
        "packageName":"Default",
        "enabled":"true",
        "version":"9.9.0.0.102"
},
{
        "packageName":"CustomerPkg01",
        "enabled":"true",
        "version":"2.5.0"
}]
```

The above definition then results in gathering information such as:

- Property key: Default

  Property Value: 9.9.0.0.102
- Property key: CustomerPackageX

  Property Value: 2.5.0

Property Filters

Similar to above Component and Instance filters, it might make sense to exclude certain properties that are known to be different and should not end up in the report as assertion failures.

A single filter is composed of the following four parameters:

- ⇨ nodeAlias
- ⇨ component
- ⇨ instance
- ⇨ key

E.g.:

```
propertyFilters:
  - nodeAlias: ANY
    component: integrationServer-*
    instance: COMMON-SYSPROPS
    key: /watt.server.log.maxEntries, /watt.server.auth.samlResolver
```

The above example exludes comparing properties “watt.server.log.maxEntries” and “watt.server.auth.samlResolver” (because they would normally be different)

All 2-Node assertion property files are located under the “audit_two_node” directory. The name of the property file will then be displayed in the OpenCM UI, under the 2-Node Assertion menu.

4.4.1 Main Service

In order to initiate a 2-Node assertion, the following main service is invoked:

- ⇨ OpenCM.pub.audit:run

This service can also be initiated from the OpenCM UI, specifying the property file to use.

4.4.2 Implementation

The node-node comparisons are based on a Junit approach, and more specifically the TestNG libraries (http://testng.org), also licensed under Apache 2.

The first part of the assertion process is to collect all the property information for node 01 and node 02, and thereafter run the assertion.

The tests are grouped by test suites, which are in our case equal to the single test type:

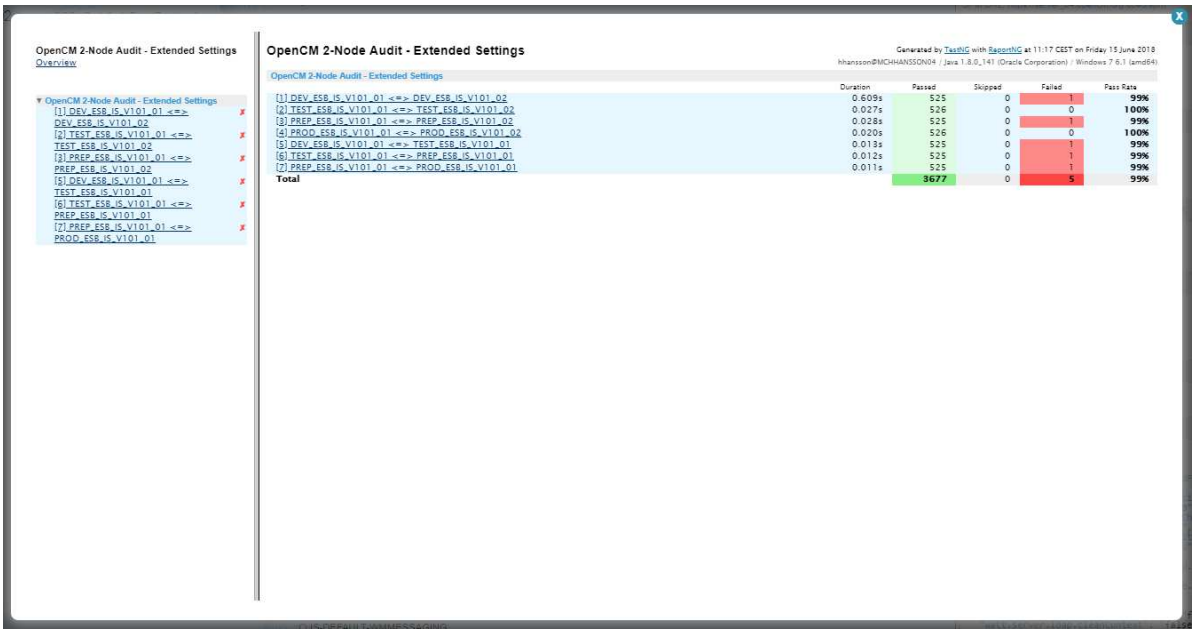
- Properties

Each test suite is made up of multiple test cases, and equals the list of node pairs we are comparing.

Each test case is then made up of individual tests, relating to the property – property comparison that are completed.

4.4.3 Result

Once the assertion process is completed, the output (result) can be viewed via the OpenCM UI, under the 2-Node Assertion – View Report menu item. E.g.:

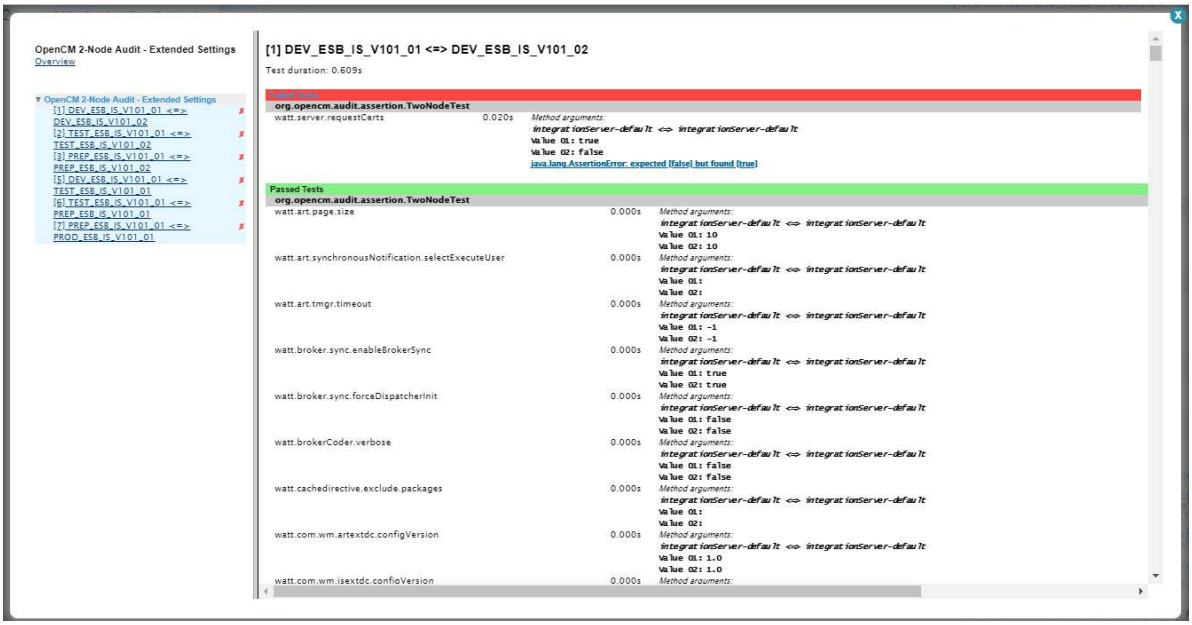


The above screenshot visualizes the result from an audit of Extended Settings, with the test name of „Audit - Extended Settings“.

All in all, there are 7 test cases (7 node-node comparisons), comparing differences between the clusters as well as in between environments. On the left-hand side, the overview is shown (red cross indicates discrepancies) and the main central part of the screen are more details.

All in all, 3677 tests were successful (equal settings) and 5 failed (differences in settings).

To inspect further details as to what differs, one would enter into a particular test case:



The top part of the main section shows the failed assertions. To interpret the failures, one can see the following:

- There is an extended setting named „watt.server.requestCerts“
- On node 01 (DEV_ESB_IS_V101_01) the value is „true”
- On node 02 (DEV_ESB_IS_V101_02) the value is „false”

4.4.4 Baseline vs. Runtime Node Auditing

An additional form of 2-node auditing exists for a single installation, where baseline data and runtime data exists. This means that one can compare differences between what exists in the baseline (source of truth) and the latest runtime extract.

Main node audit service is OpenCM.pub.audit:nodeAudit and can be triggered from the frontend page, under the „2-Node Audit“ – „Baseline-Runtime Node Audit“ menu. This menu item will only be present when accessing a particular node.

It takes one parameter:

- node (what node to audit)

The audit process does not use a separate configuration propery, instead it compares all available configuration properties stored in the baseline and runtime directories.

4.4.5 Default Properties Auditing

Another type of auditing relates to the ability to find out what has changed in a runtime or baseline repository compared to default values (values configured in a plain vanilla installation). This would show all the configuration settings that have been changed after installation.

The way to perform this type of auditing is based on default values being stored within the OpenCM package (under OpenCM/resources/default) and currently the following default values have been generated:

- Version 9.9 (Integration Server, My webMethods Server, UM and Terracotta)
- Version 9.12 (Integration Server, My webMethods Server, UM and Terracotta)

Main node audit service is OpenCM.pub.audit:nodeAudit and can be triggered from the frontend page, under the „2-Node Audit“:

- „Default vs. Baseline Node Audit“
- „Default vs. Runtime Node Audit“

The above menu items will only be present when accessing a particular node.

It takes one parameter:

- node (what node to audit)

The auditing process makes use of a single property file that defines the filters used on the comparisons (defining properties that should be excluded from the audit). The file is located in the package config directory named „default_auditing.properties“.

4.5 Layered Auditing

One of the limitations of above 2-Node Assertions is the difficulty in determining what needs to be done to harmonize the environments, i.e. to make them look the same. Taking Fix levels as an example, we know that we have differences between two nodes in Production and Pre-Prod environment, but what about all the other environments? Maybe for the NUMClient.CommonLibraries fix, there are further discrepancies in the test and maintenance environments.

In order to decide what to do, one must have a more comprehensive view on the differences for a particular property across all environments. The end goal would look as follows (e.g.):

Property	DEV	TEST	UAT	PreProd	PROD	MAINT
IS Core Fix	Node 01 = Fix11 Node 02 = Fix11	Node 01 = Fix7 Node 02 = Fix7	Node 01 = Fix7 Node 02 = Fix7	Node 01 = Fix5 Node 02 = Fix5	Node 01 = Fix5 Node 02 = Fix5	Node 01 = Fix5 Node 02 = Fix5
JDBC Fix	Node 01 = Fix11 Node 02 = Fix11	Node 01 = Fix4 Node 02 = Fix4	Node 01 = Fix11 Node 02 = Fix11	Node 01 = Fix11 Node 02 = Fix11	Node 01 = Fix11 Node 02 = Fix11	Node 01 = Fix11 Node 02 = Fix11
SPM Fix	Node 01 = Fix14 Node 02 = Fix8	Node 01 = Fix7 Node 02 = ” ”	Node 01 = Fix7 Node 02 = Fix7	Node 01 = Fix5 Node 02 = Fix5	Node 01 = Fix7 Node 02 = Fix5	Node 01 = ” ” Node 02 = ” ”
…						

The above table illustrates that for a single property, there are values examined for all environments and will then provide “decision material” on what to do. For example, the IS Core fix is different and to harmonize, we must apply Fix11 to all environments (except DEV). This then assuming the strategy is to take the highest found fix level and upgrading the ones that don’t have it.

The approach is called “Layered Auditing”, due to the fact that comparisons are done layer by layer (as opposed to node by node). A table such as above would only be applicable if comparions are performed on installations with similar purpose. E.g. comparing fix levels between ESB Integration Servers and Universal Messaging Server does not make sense. Therefore, all the nodes that are related to the “ESB” layer would form one group and all the Universal Messaging installations would form another group. These two groups are referred to as two separate “assertion groups”.

A successful audit (all ok) would then be the result of comparing all the nodes within an assertion group and all nodes possess the same value for a particular property.

4.5.1 Main Service

The layered audit service that performs a more „environment-wide“ audit of all properties is:

⇨ OpenCM.pub.audit:runAuditEnv

Again, this service can be invoked from the OpenCM UI.

To support this service, a property file is used to define what property will be audited:

<layered_audit>.properties - definition of what to audit, similar to the earlier described 2-node assertion audit.properties. <layered_audit> name refers to the custom name one choses for the audit.

```
# ----------------------------------------------------------------------
# OpenCM Environment Audit Property file (yaml notation).
# ----------------------------------------------------------------------
# ----------------------------------------------
reportName: "OpenCM Layered Audit - IS Extended Settings"
# ----------------------------------------------------------------------
# Assertion Groups (if empty, then all)
# ----------------------------------------------------------------------
assertionGroups:
  - ESB_IS_V101
#  - MFT_V912
#  - MED_V912

# ----------------------------------------------------------------------
# OpenCM Environments (if empty, then all per nodes definitions)
# ----------------------------------------------------------------------
environments:
  - Development
  - Test
  - Pre-Production
  - Production

# ----------------------------------------------------------------------
```



```
# Properties Assertion Configuration
#   - includeDefaultValues = to include in the report the default values for the particular property (if
found)
# ------------------------------------------------------------------------------
includeDefaultValues: true
properties:
  # ------------------------------------------------
  # - Asserting IS extended settings
  # ------------------------------------------------
  - component: integrationServer-*
    instance: COMMON-SYSPROPS
    key: /

# -------------------------
# Ignoring Properties
# -------------------------
propertyFilters:
  - nodeAlias: ANY
    component: integrationServer-*
    instance: COMMON-SYSPROPS
    key:
/watt.server.compile,/watt.server.log.maxEntries,/watt.server.auth.samlResolver,/watt.sap.SLDSetting,/
watt.security.sign.keyAlias,/watt.security.sign.keyStoreAlias,/watt.security.decrypt.keyAlias,/watt.securi
ty.decrypt.keyStoreAlias,/watt.security.ssl.keyAlias,/watt.security.ssl.keyStoreAlias

# -------------------------
# Show/Hide equals (i.e. when all property values are equal)
# -------------------------
hideEquals: false

# -------------------------
# Conditional Formatting
#  - Apply cell coloring using rules for every property row:
#    * EQUALS => if all property values are equal
#    * DIFFS => if any property value is different
#    * <custom> => e.g. ODD(ROW())=ROW()
# -------------------------
formatting:
  - rule: EQUALS
    bgcolor: "#E6FFE6"
  - rule: DIFFS
    bgcolor: "#FACFC1"
```

Some clarifications:

- **reportName:** a String value that will end up in the Excel Spreadsheet heading
- **assertionGroups:** a list of assertion groups to generate the report. If the list is empty, then it is assumed that all assertion groups defined (in the above nodes.properties) will be used.
- **environments:** a list of opencm environment names to include in the audit.
- **properties/propertyFilters:** same meaning as above 2-node assertions.
- **hideEquals:** show or hide the property rows that are analyzed as equals
- **formatting:** this is a section that affects the coloring/highlighting of the property rows.

  The formatting section comes with three separate values:

  a. rule
  b. range
  c. bgcolor

  For rule, there are two pre-defined *rule* values:

  1. EQUALS (what to do with the property rows that are equal)

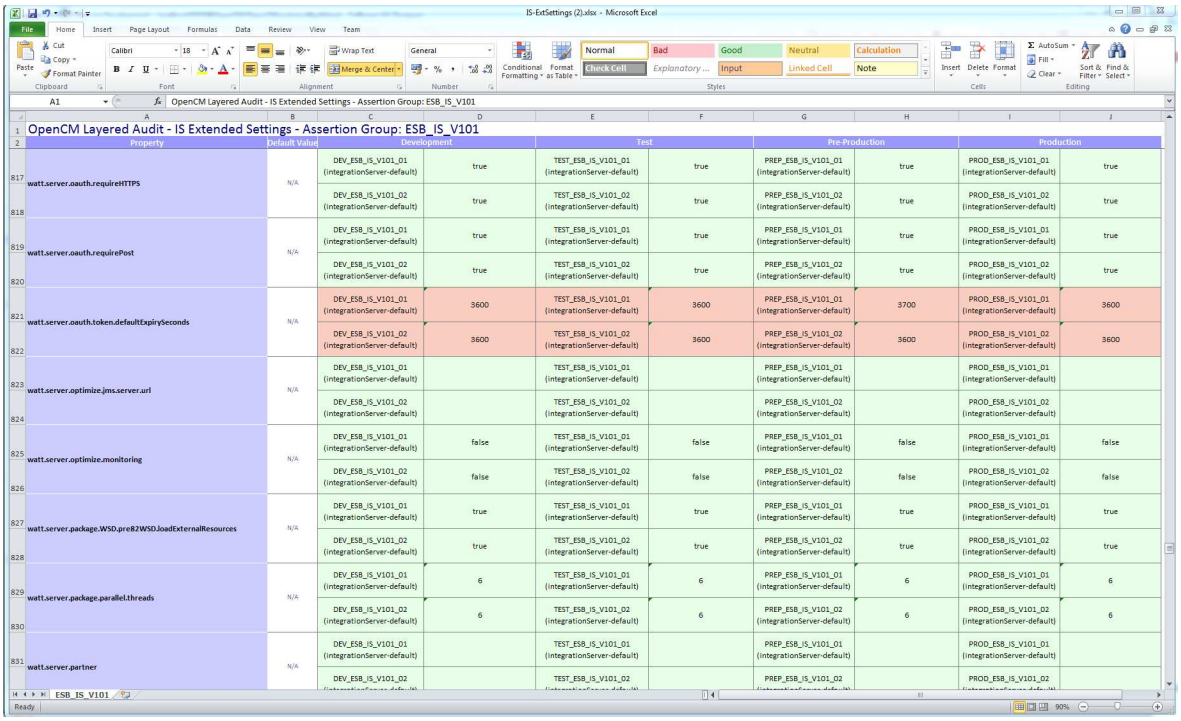
2. DIFFS (what to do with the property rows that are equal)

When above values are used, the range value is not applicable since it highlights all the property values across all rows.

4.5.2 Result

After processing has completed, an Excel file has been generated and stored under the <opencm>/output/excel folder. The excel file can also be accessed via the OpenCM UI.

A sample output:



OpenCM Layered Audit - IS Extended Settings - Assertion Group: ESB_IS_V101

Property	Default Value	Development		Test		Pre-Production		Production	
watt.server.oauth.requireHTTPS	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	true	TEST_ESB_IS_V101_01 (integrationServer-default)	true	PREP_ESB_IS_V101_01 (integrationServer-default)	true	PROD_ESB_IS_V101_01 (integrationServer-default)	true
		DEV_ESB_IS_V101_02 (integrationServer-default)	true	TEST_ESB_IS_V101_02 (integrationServer-default)	true	PREP_ESB_IS_V101_02 (integrationServer-default)	true	PROD_ESB_IS_V101_02 (integrationServer-default)	true
watt.server.oauth.requirePost	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	true	TEST_ESB_IS_V101_01 (integrationServer-default)	true	PREP_ESB_IS_V101_01 (integrationServer-default)	true	PROD_ESB_IS_V101_01 (integrationServer-default)	true
		DEV_ESB_IS_V101_02 (integrationServer-default)	true	TEST_ESB_IS_V101_02 (integrationServer-default)	true	PREP_ESB_IS_V101_02 (integrationServer-default)	true	PROD_ESB_IS_V101_02 (integrationServer-default)	true
watt.server.oauth.token.defaultExpirySeconds	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	3600	TEST_ESB_IS_V101_01 (integrationServer-default)	3600	PREP_ESB_IS_V101_01 (integrationServer-default)	3700	PROD_ESB_IS_V101_01 (integrationServer-default)	3600
		DEV_ESB_IS_V101_02 (integrationServer-default)	3600	TEST_ESB_IS_V101_02 (integrationServer-default)	3600	PREP_ESB_IS_V101_02 (integrationServer-default)	3600	PROD_ESB_IS_V101_02 (integrationServer-default)	3600
watt.server.optimize.jms.server.url	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)		TEST_ESB_IS_V101_01 (integrationServer-default)		PREP_ESB_IS_V101_01 (integrationServer-default)		PROD_ESB_IS_V101_01 (integrationServer-default)	
		DEV_ESB_IS_V101_02 (integrationServer-default)		TEST_ESB_IS_V101_02 (integrationServer-default)		PREP_ESB_IS_V101_02 (integrationServer-default)		PROD_ESB_IS_V101_02 (integrationServer-default)	
watt.server.optimize.monitoring	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	false	TEST_ESB_IS_V101_01 (integrationServer-default)	false	PREP_ESB_IS_V101_01 (integrationServer-default)	false	PROD_ESB_IS_V101_01 (integrationServer-default)	false
		DEV_ESB_IS_V101_02 (integrationServer-default)	false	TEST_ESB_IS_V101_02 (integrationServer-default)	false	PREP_ESB_IS_V101_02 (integrationServer-default)	false	PROD_ESB_IS_V101_02 (integrationServer-default)	false
watt.server.package.WSD.pre82WSDloadExternalResources	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	true	TEST_ESB_IS_V101_01 (integrationServer-default)	true	PREP_ESB_IS_V101_01 (integrationServer-default)	true	PROD_ESB_IS_V101_01 (integrationServer-default)	true
		DEV_ESB_IS_V101_02 (integrationServer-default)	true	TEST_ESB_IS_V101_02 (integrationServer-default)	true	PREP_ESB_IS_V101_02 (integrationServer-default)	true	PROD_ESB_IS_V101_02 (integrationServer-default)	true
watt.server.package.parallel.threads	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	6	TEST_ESB_IS_V101_01 (integrationServer-default)	6	PREP_ESB_IS_V101_01 (integrationServer-default)	6	PROD_ESB_IS_V101_01 (integrationServer-default)	6
		DEV_ESB_IS_V101_02 (integrationServer-default)	6	TEST_ESB_IS_V101_02 (integrationServer-default)	6	PREP_ESB_IS_V101_02 (integrationServer-default)	6	PROD_ESB_IS_V101_02 (integrationServer-default)	6
watt.server.partner	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)		TEST_ESB_IS_V101_01 (integrationServer-default)		PREP_ESB_IS_V101_01 (integrationServer-default)		PROD_ESB_IS_V101_01 (integrationServer-default)	
		DEV_ESB_IS_V101_02 (integrationServer-default)		TEST_ESB_IS_V101_02 (integrationServer-default)		PREP_ESB_IS_V101_02 (integrationServer-default)		PROD_ESB_IS_V101_02 (integrationServer-default)	

The above screen print illustrates the report format of a extended settings audit for a particular assertion group (ESB_IS_V101). Each assertion group would be within a separate tab.

- The left side column (A+B) lists the various properties (in this case extended setting).
- The columns C+D shows information about what values we have (for that property) in the Development environment. Each environment is made up of two columns:
    - Location (e.g. node name)
    - Property Value (e.g. fix version)
- A green property row indicates that all values for the property are identical
- A white cell indicates that the property is not available (but should be). In this case it refers to an extended setting that is not present.
- A red cell text indicates that there are differences (somewhere) for the property in one or many of the environments
- A gray cell indicates that there are no defined locations. E.g. one environment may have a three-node cluster and in another it is a stand-alone node (but configuration should be equal).

For each layered audit run, an Excel file will be generated in the output directory and whether to hide/show the equal values (for a single property) can be configured via the property file that

drives the layered audit. This way, only the differences will be displayed when opening the Excel file.

Complex Formatting

A more sophisticated colouring/highlighting of the Excel sheet can be done by tweaking the formatting configuration. Example:

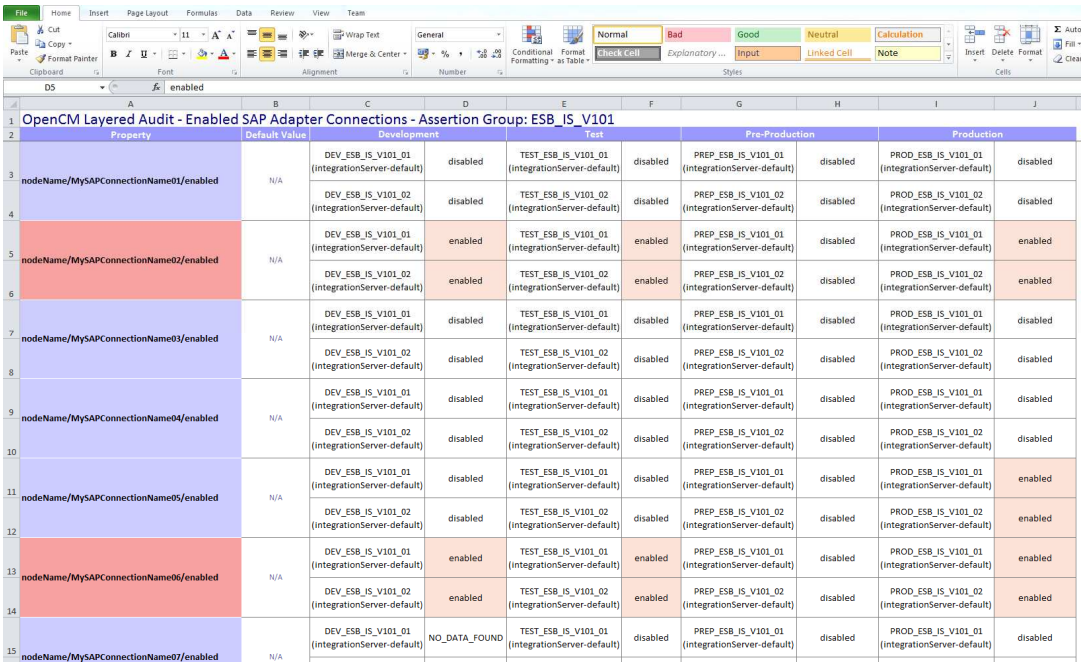
```
- rule:
AND(ODD(ROW())=ROW(),IF(IF(COUNTIFS(OFFSET(A1,0,3,2,1),"enabled")>0,1,0)+IF(COUNTIFS(OFFSET(A1,0,5,
2,1),"enabled")>0,1,0)+IF(COUNTIFS(OFFSET(A1,0,7,2,1),"enabled")>0,1,0)+IF(COUNTIFS(OFFSET(A1,0,9,2,1),"e
nabled")>0,1,0)>1,TRUE,FALSE))

 range: A1:A1048576
 bgcolor: "#F7A1A1"

- rule: A1="enabled"
 range: A1:XFD1048576
 bgcolor: "#FCE2D7"
```

In this particular example, we are highlighting SAP adapter connections that are enabled across multiple environments. I.e. if there are one or multiple connections with the same name that are enabled, then they would be shown in the color definied.

- The first rule highlights the name of the property (the name of the SAP Adapter Connection) – column A.
- The second rule simply highlights all individual property values in columns D – H.



OpenCM Layered Audit - Enabled SAP Adapter Connections - Assertion Group: ESB_IS_V101

Property	Default Value	Development		Test		Pre-Production		Production	
nodeName/MySAPConnectionName01/enabled	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	disabled	TEST_ESB_IS_V101_01 (integrationServer-default)	disabled	PREP_ESB_IS_V101_01 (integrationServer-default)	disabled	PROD_ESB_IS_V101_01 (integrationServer-default)	disabled
		DEV_ESB_IS_V101_02 (integrationServer-default)	disabled	TEST_ESB_IS_V101_02 (integrationServer-default)	disabled	PREP_ESB_IS_V101_02 (integrationServer-default)	disabled	PROD_ESB_IS_V101_02 (integrationServer-default)	disabled
nodeName/MySAPConnectionName02/enabled	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	enabled	TEST_ESB_IS_V101_01 (integrationServer-default)	enabled	PREP_ESB_IS_V101_01 (integrationServer-default)	disabled	PROD_ESB_IS_V101_01 (integrationServer-default)	enabled
		DEV_ESB_IS_V101_02 (integrationServer-default)	enabled	TEST_ESB_IS_V101_02 (integrationServer-default)	enabled	PREP_ESB_IS_V101_02 (integrationServer-default)	disabled	PROD_ESB_IS_V101_02 (integrationServer-default)	enabled
nodeName/MySAPConnectionName03/enabled	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	disabled	TEST_ESB_IS_V101_01 (integrationServer-default)	disabled	PREP_ESB_IS_V101_01 (integrationServer-default)	disabled	PROD_ESB_IS_V101_01 (integrationServer-default)	disabled
		DEV_ESB_IS_V101_02 (integrationServer-default)	disabled	TEST_ESB_IS_V101_02 (integrationServer-default)	disabled	PREP_ESB_IS_V101_02 (integrationServer-default)	disabled	PROD_ESB_IS_V101_02 (integrationServer-default)	disabled
nodeName/MySAPConnectionName04/enabled	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	disabled	TEST_ESB_IS_V101_01 (integrationServer-default)	disabled	PREP_ESB_IS_V101_01 (integrationServer-default)	disabled	PROD_ESB_IS_V101_01 (integrationServer-default)	disabled
		DEV_ESB_IS_V101_02 (integrationServer-default)	disabled	TEST_ESB_IS_V101_02 (integrationServer-default)	disabled	PREP_ESB_IS_V101_02 (integrationServer-default)	disabled	PROD_ESB_IS_V101_02 (integrationServer-default)	disabled
nodeName/MySAPConnectionName05/enabled	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	disabled	TEST_ESB_IS_V101_01 (integrationServer-default)	disabled	PREP_ESB_IS_V101_01 (integrationServer-default)	disabled	PROD_ESB_IS_V101_01 (integrationServer-default)	enabled
		DEV_ESB_IS_V101_02 (integrationServer-default)	disabled	TEST_ESB_IS_V101_02 (integrationServer-default)	disabled	PREP_ESB_IS_V101_02 (integrationServer-default)	disabled	PROD_ESB_IS_V101_02 (integrationServer-default)	enabled
nodeName/MySAPConnectionName06/enabled	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	enabled	TEST_ESB_IS_V101_01 (integrationServer-default)	enabled	PREP_ESB_IS_V101_01 (integrationServer-default)	disabled	PROD_ESB_IS_V101_01 (integrationServer-default)	enabled
		DEV_ESB_IS_V101_02 (integrationServer-default)	enabled	TEST_ESB_IS_V101_02 (integrationServer-default)	enabled	PREP_ESB_IS_V101_02 (integrationServer-default)	disabled	PROD_ESB_IS_V101_02 (integrationServer-default)	enabled
nodeName/MySAPConnectionName07/enabled	N/A	DEV_ESB_IS_V101_01 (integrationServer-default)	NO_DATA_FOUND	TEST_ESB_IS_V101_01 (integrationServer-default)	disabled	PREP_ESB_IS_V101_01 (integrationServer-default)	disabled	PROD_ESB_IS_V101_01 (integrationServer-default)	disabled

The main point here is that highlighting of individual property rows can be configured as per requirements, using whatever color desirable.

4.6 UI Configuration

Some helper services can be invoked from the OpenCM UI directly with the following functionality:

1. **Refresh Tree**

   This service refreshes the d3 tree information in the cmdata directories. This can useful when manually re-arranging the extracted data. For example, when adding a new server installation to an existing environment, only that installation can be extracted (specified in the extract.properties file). This extraction generates a new snapshot and when finished, one can move the complete server/node directories to a previous snapshot that contains all the rest of the nodes from that same environment.

   Once this move has been done, a refresh of the d3 tree information is needed so that the UI will show the new installation as part of the environment from the previous snapshot.

2. **Encrypt & Decrypt Credentials**

   These two services encrypts or decrypts the passwords in the nodes.properties file. These functions are in other words not applicable when using a password vault such as Keepass.

3. **Synchronization :: Send**

   Refer to below section “OpenCM Data Synchronization”.

4. **Command Central: Create All**

   This function removes all existing environment and node definitions currently defined on the Comman Central management component (as defined in the main opencm.properties).

   Based in the information in the OpenCM node store (either nodes.properties or Keepass), all environments are created and all nodes under each environment are created in Command Central.

   Note: in the event of having multiple Command Central installations, along with separate OpenCM installations responsible for different parts of the DBP landscape, there are environments and nodes defined in the OpenCM node store that are not considered “local”. Therefore, when creating nodes and environments in Command Central, only the locally extracted nodes are generated. This information is based on extracted data, and more specifically in the runtime server directory (properties.json). If the “extractAlias” is the same as the local OpenCM node alias, it would be considered a local installation and therefore created in CCE.

   This verification can be overridden by the “cce_mgmt_enforce_extraction_node” property in the main opencm.properties file. “True” means that this verification is enforced.

5. **Command Central: Add Node**

   This function adds the current node to the Command Central UI, and creates (if not present) the corresponding environment. There is no “verification” (as above) here to ensure that it really is a locally extracted node.

4.7 OpenCM – OpenCM Data Synchronization

In the event there are mutiple administrative domains, with isolated (restricted) environments in place, it is possible to have multiple OpenCM installations. Each OpenCM instance is responsible for extracting data within their respective domain and the extracted data can thereafter be synchronized with another OpenCM installation.

In order to instead perform the data integration automatically, the OpenCM comes with a synchronization feature, which involves zipping up the runtime server directories and performing an FTPS transfer to a target OpenCM location. This is done on a server by server basis from the runtime directory.

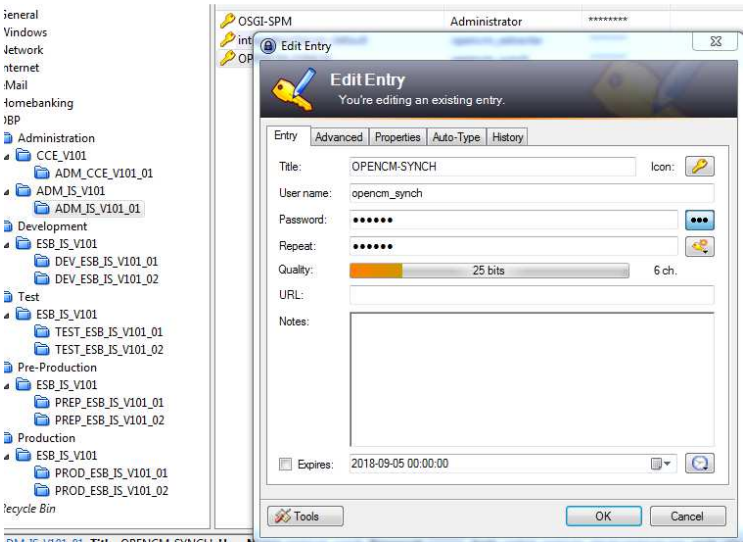
The source node, which will be sending information, will need to be configured to hold the FTPS endpoint settings to the target node. The “local” vs. “target” OpenCM node names are defined in the

main opencm.properties file and when sending, the “OPENCM_SYNCH” runtime component must be defined with the appropriate FTPS endpoint information (see below). Refer to section 4.1.3 for more information.

The “OPENCM_SYNCH” runtime component is defined as follows, which will then hold the appropriate information (e.g):

```
- node_name: "ADM_IS_V101_01"
  environment: "Administration"
  hostname: "central_opencm_server.opencm.org"
  assertion_group: "ADM_IS_V101"
  runtimeComponents:
  - name: "OPENCM-SYNCH"
    protocol: "ftps"
    port: "10021"
    username: "opencm_synch"
    password: "[ENCRYPTED::Re4FU/dKdWGBHrZAUDsveQ==:+orfXCDEooHuvF8DSIPyow==]"
```

The same can optionally be defined via Keepass:



In addition to above endpoint configuration, there is a timeout configuration that can be set, - configured from the opencm.properties:

```
# ------------------------------------------------------------
# Synchronization config (used by the send process)
# ------------------------------------------------------------
local_opencm_node: ADM_IS_V101_01
target_opencm_node: ADM_IS_V101_01
ftps_timeout_ms: "5000"
```

To enable the synchronization process, it is therefore needed to enable an FTPS port on the central IS hosting the OpenCM package, along with a user that can have sufficient access privilges to initiate an FTPS transfer process (in the example above, there is a separate opencm_synch user defined).

Note: the FTPS process is not making use of JSSE. When defining the FTPS port, user JSSE = no.

4.7.1 Main Send Service

The synchronization process is started by invoking the following service:

⇨ OpenCM.pub.synch:send

This service can also be manually invoked via OpenCM UI (Administration).

4.7.2 Main Receive Service

The above send service will compress an individual runtime server directory and perform an FTPS login to the target FTPS server. Thereafter, a „cd“ to the following namespace will be done:

/ns/OpenCM/pub/synch/receive

By „putting“ the zip file inthe this folder, the target OpenCM instance will therefore execute the following service:

⇨ OpenCM.pub.synch:receive

2. Appendix A – OpenCM Licenses

This section covers the OpenCM license as well as other third party libraries and utilities included into the OpenCM package, and their associated licenses. Each license file (text) can be found within the OpenCM package folder, under /pub/about/licenses, and are also shown from the about page of the OpenCM user interface.

2.1. OpenCM

The utility itself is licensed under the Apache 2.0 license and published on GitHub/SoftwareAG

Package	Licensed under	Source
OpenCM	Apache 2.0	https://www.apache.org/licenses/LICENSE-2.0

2.2. Java Scripts

Files	Licensed under	Source
d3.v3.min.js	BSD-3-Clause	https://github.com/d3/d3
jquery.json-view.js	MIT	https://github.com/yesmeck/jquery-jsonview
jquery-1.10.2.min.js	MIT	https://jquery.com/download/
opencm.js	MIT	Originated from http://bl.ocks.org/Thanaporn-sk/20a86f09c938b8dd0e4520a4c68c6315

2.3. css Files

css Files	Licensed under	Source
jquery.json-view.css	MIT	https://github.com/yesmeck/jquery-jsonview
opencm.css	MIT	Originated from http://bl.ocks.org/Thanaporn-sk/20a86f09c938b8dd0e4520a4c68c6315

2.4. Images and Icons

Image Files	Licensed under	Source
tab.png (Tab icon)	Creative Commons	https://icons8.com/icon/718/database-view https://creativecommons.org/licenses/by-nd/3.0/legalcode (license)
xlsx.ico (Excel icon)	Free for non-commercial use	http://www.iconhot.com/icon/file-icons-vs-2/xlsx-3.html Author: Jordan Michael

2.5. Java Libraries

Jar Files	Licensed under	Source
asm-1.0.2.jar	Apache 2.0	https://mvnrepository.com/artifact/net.minidev/asm/1.0.2
client-combined-3.14.0.jar	Apache 2.0	https://www.seleniumhq.org/download/
commons-collections4-4.2.jar	Apache 2.0	https://mvnrepository.com/artifact/org.apache.commons/commons-collections4/4.2
commons-io-2.6.jar	Apache 2.0	https://mvnrepository.com/artifact/commons-io/commons-io/2.6
commons-lang3-3.8.jar	Apache 2.0	https://mvnrepository.com/artifact/org.apache.commons/commons-lang3/3.8
commons-net-3.6.jar	Apache 2.0	https://mvnrepository.com/artifact/commons-net/commons-net/3.6
guava-26.0-jre.jar	Apache 2.0	https://mvnrepository.com/artifact/com.google.guava/guava/26.0-jre
guice-4.2.0.jar	Apache 2.0	https://mvnrepository.com/artifact/com.google.inject/guice/4.2.0
htmlunit-2.32.jar	Apache 2.0	https://mvnrepository.com/artifact/net.sourceforge.htmlunit/htmlunit/2.32
htmlunit-cssparser-1.0.0.jar	Apache 2.0	https://mvnrepository.com/artifact/net.sourceforge.htmlunit/htmlunit-cssparser/1.0.0
htmlunit-core-js-2.32.jar	MPL 2.0	https://mvnrepository.com/artifact/net.sourceforge.htmlunit/htmlunit-core-js/2.32
htmlunit-driver-2.32.1.jar	Apache 2.0	https://github.com/SeleniumHQ/htmlunit-driver/releases
httpclient-4.5.6.jar	Apache 2.0	https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient/4.5.6
httpmime-4.5.6.jar	Apache 2.0	https://mvnrepository.com/artifact/org.apache.httpcomponents/httpmime/4.5.6
jackson-annotations-2.9.6.jar	Apache 2.0	https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations/2.9.6
jackson-core-2.9.6.jar	Apache 2.0	https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core/2.9.6
jackson-databind-2.9.6.jar	Apache 2.0	https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind/2.9.6
jackson-dataformat-yaml-2.9.6.jar	Apache 2.0	https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-yaml/2.9.6
jcommander-1.72.jar	Apache 2.0	https://mvnrepository.com/artifact/com.beust/jcommander/1.72
json-20180813.jar	Json	https://mvnrepository.com/artifact/org.json/json/20180813
json-path-2.4.0.jar	Apache 2.0	https://mvnrepository.com/artifact/com.jayway.jsonpath/json-path/2.4.0
json-smart-2.3.jar	Apache 2.0	https://mvnrepository.com/artifact/net.minidev/json-smart/2.3
neko-htmlunit-2.32.jar	Apache 2.0	https://mvnrepository.com/artifact/net.sourceforge.htmlunit/neko-htmlunit/2.32

openkeepass-0.8.2-wd.jar	Apache 2.0	https://mvnrepository.com/artifact/de.slackspace/openkeepass/0.8.2
poi-3.17.jar	Apache 2.0	https://www.apache.org/dyn/closer.lua/poi/release/bin/poi-bin-3.17-20170915.zip
poi-ooxml-3.17.jar		
poi-ooxml-schemas-3.17.jar		
reportng-1.1.4.jar	Apache 2.0	https://mvnrepository.com/artifact/org.uncommons/reportng/1.1.4
testng-6.14.3.jar	Apache 2.0	http://mvnrepository.com/artifact/org.testng/testng

