OPENCM USER GUIDE

WEBMETHODS CONFIGURATION MANAGEMENT

Version 2.0.0 | 19.03.2019

VERSION

Version	Date	Author	Description
1.8.2	30.08.2018	Håkan Hansson	First public release, relating to implementation v1.8.2
1.8.9	20.10.2018	Håkan Hansson	Improved Inventory items, added to main.
1.9.0	10.12.2018	Håkan Hansson	Restructuring of inventory. As a consequence, many configuration files are updated to reflect the new structure.
2.0.0	19.03.2019	Håkan Hansson	Consolidation of auditing functionality. Auditing is now performed via a UI Wizard.

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:

- **UI for Inventory:** all installations and their associated meta information (e.g. which organisation, which department, what environment, etc.) an installation belongs to.
- **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
  - SAP Adapter connection information

  - WxConfig key-value pairs.
- **UI for Properties:** 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. All functions below are also possible to perform directly via the OpenCM UI.
- **Auditing of Properties** – Ability to compare any configuration property between different installations. The following types of auditing actions are available:
  - **Layered Auditing** – ability to compare differences between multiple locations. E.g. a fix level is compared across all installations for multiple environments.
  - **Baseline Auditing** – ability to compare differences between the baseline repository and either runtime or default repository, performed on a node by node basis.
  - **Reference Auditing** - ability to compare differences between a single "reference node" baseline repository and multiple other installations.
- **Bi-directional Runtime Data Synchronization** – allows for sending (via FTPS) extracted files from one OpenCM installation to another and vice versa. This is useful for domains that are physically separated (using multiple Command Central components for administration).
- **Command Central – Creating Nodes**: can create both CCE environments and installation nodes based on the OpenCM inventory.
- **Command Central** – **Fix Scripts:** based on an existing installation, can generate a Command Central CLI fix script for applying the same fixes (and the exact same version) in another installation.

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

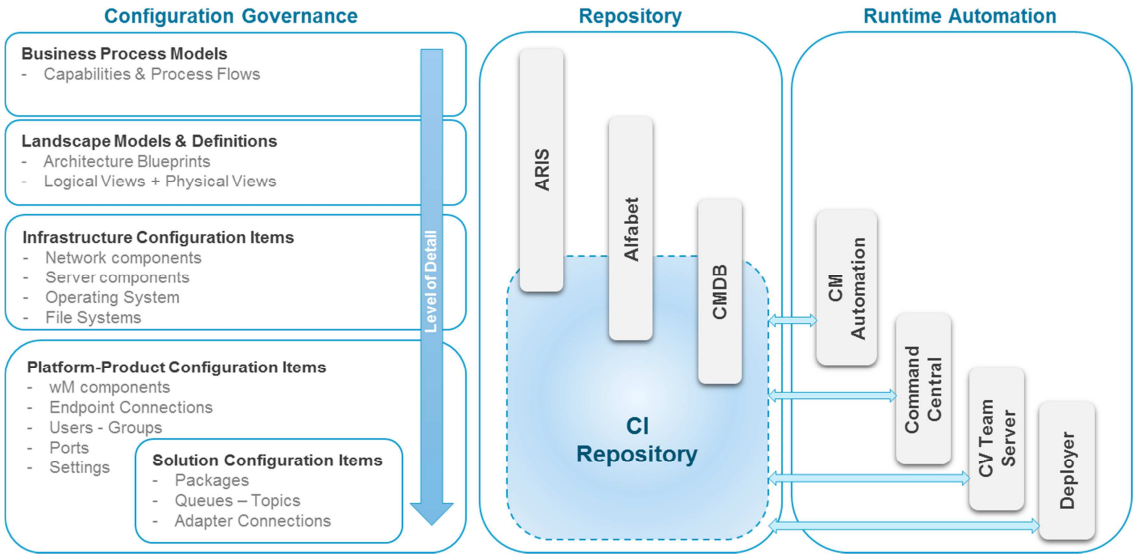
Somewhat more mature organisations will document the settings, the configurations and the properties of the DBP environments and will most likely use a more static repository such as Word documents, Visio diagrams, Excel spreadsheets or a Wiki-type location. The problems with static repositories are:

- 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.

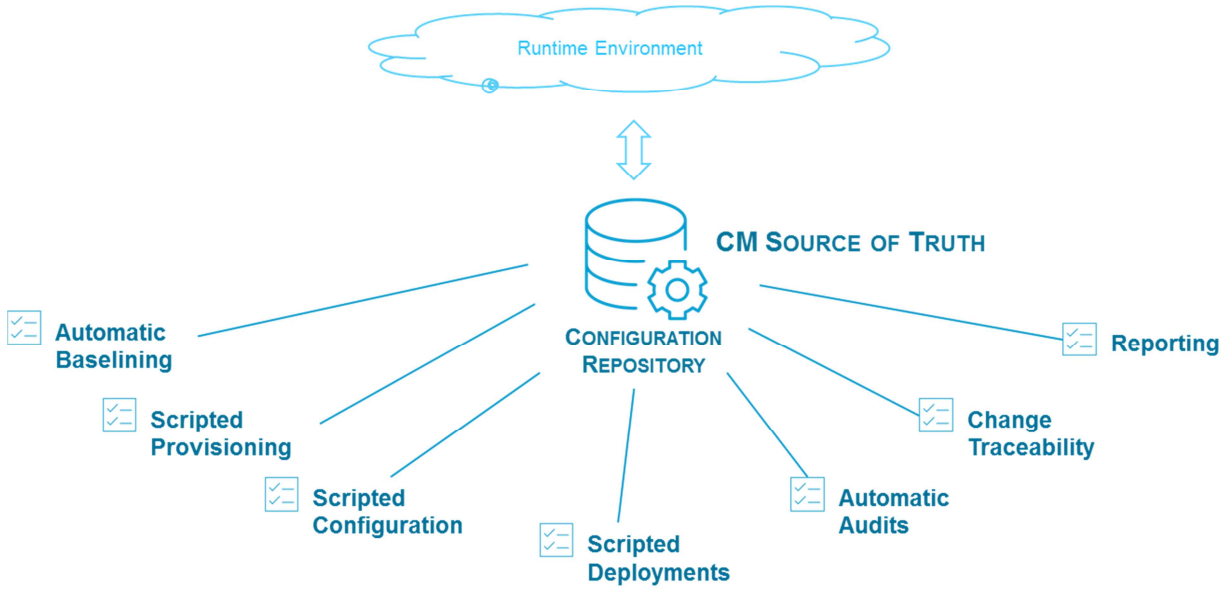
- In the middle of the diagram, are identified (potential) respositories of configuration information:
  - SoftwareAG ARIS provides a repository for the DBP blueprints and the physical architectural views can be extended to cover finer granularity of the assets. However, the volume of data that is stored 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

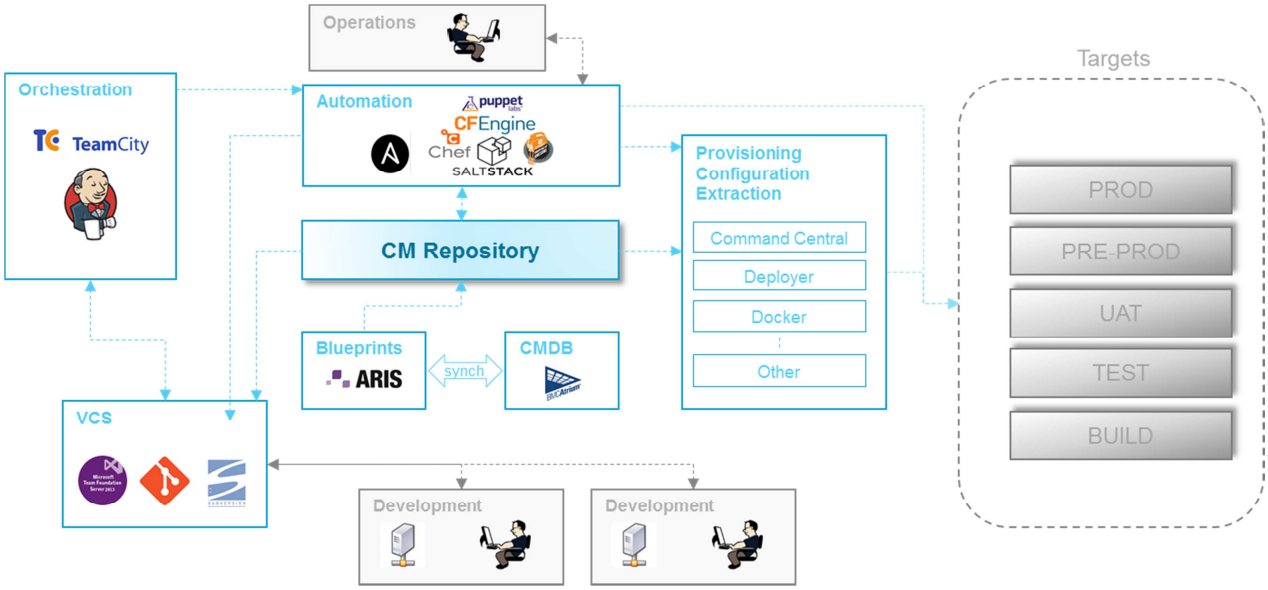
The CM strategy must support the ability to perform regular, automated environment audits as per content of the SoT. There could be various use cases and scope for the auditing, but the requirement is that the implementation should manage to perform retrieval of runtime information (in various environments) 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, 10.1 and 10.3.

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.
- ✓ If Keepass is used to define the nodes inventory items, the JCE cryptographic libraries (unlimited strength) is required.

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 external opencm directory. 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 opencm F:\SoftwareAG\opencm

      Linux:


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

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:
    - *inventory_config* refers to the type of endpoint definitions that are used. Refer to section 4.1.3 for more information.
    - *cce_mgmt_** refers to the Command Central management operations for defining environments and installation nodes directly in Command Central UI. Refer to section 4.6 for additiona information.
    - *synch_** elements refer to synchronization functionality. Refer to section 4.7 for more information.
    - *log_level* refers to the level of detail to log (see below).

1.5.2. Adjust as per local Environment

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

1. Define the inventory 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

All above activities is best performed by starting out with the template properties provided along with (and separate to) the IS package.

1.5.3. Logging

Any operation carried out will be logged according to the log level defined in the opencm.properties file. Logging output is in the IS wrapper.log, located in the profiles\IS_<instance>\logs directory. The log levels defined are:

- CRITICAL
- WARNING
- INFO
- DEBUG
- TRACE

4.0 Implementation

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

4.1 Visualization

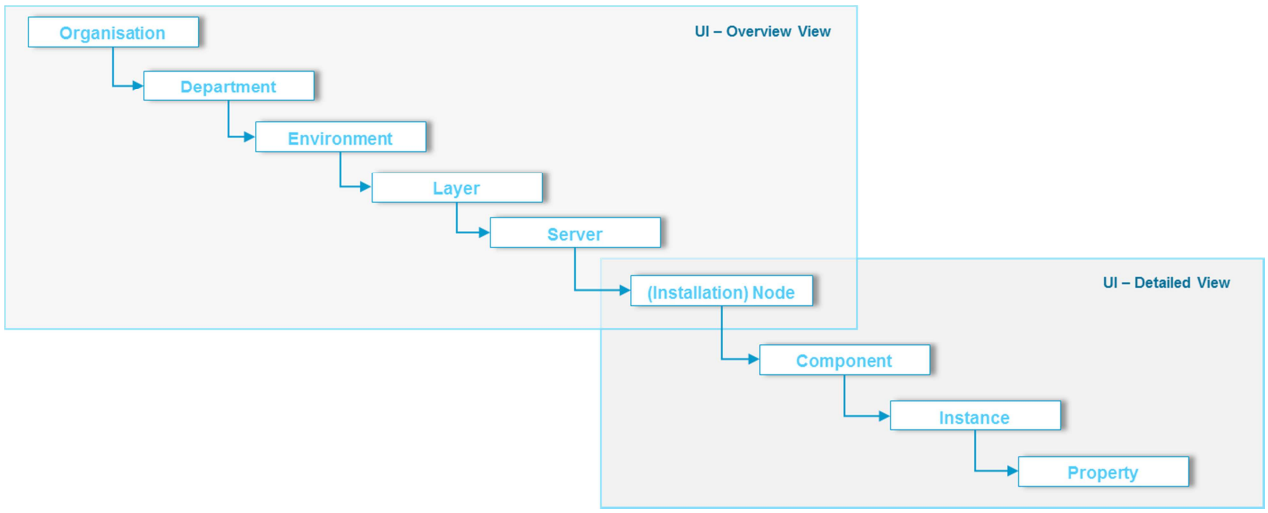
Visualization covers the OpenCM home page and the ability to view the baseline and runtime snaphot information. It also covers the ability to invoke certain functions from the home page via menues.

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).

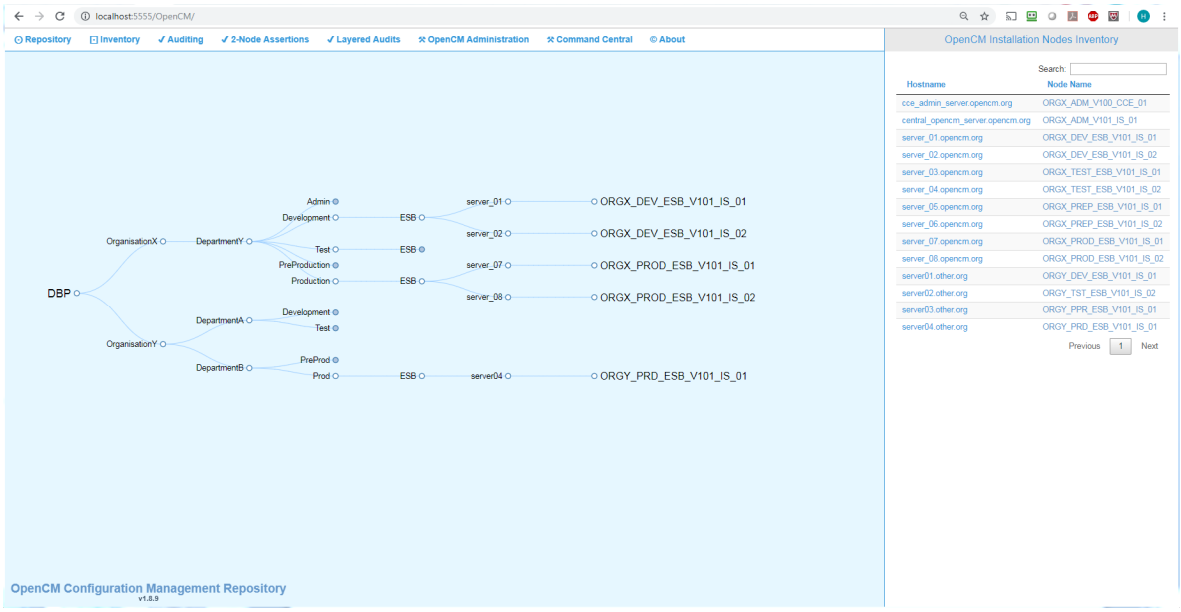
There are two separate views:

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

The structure is as follows:



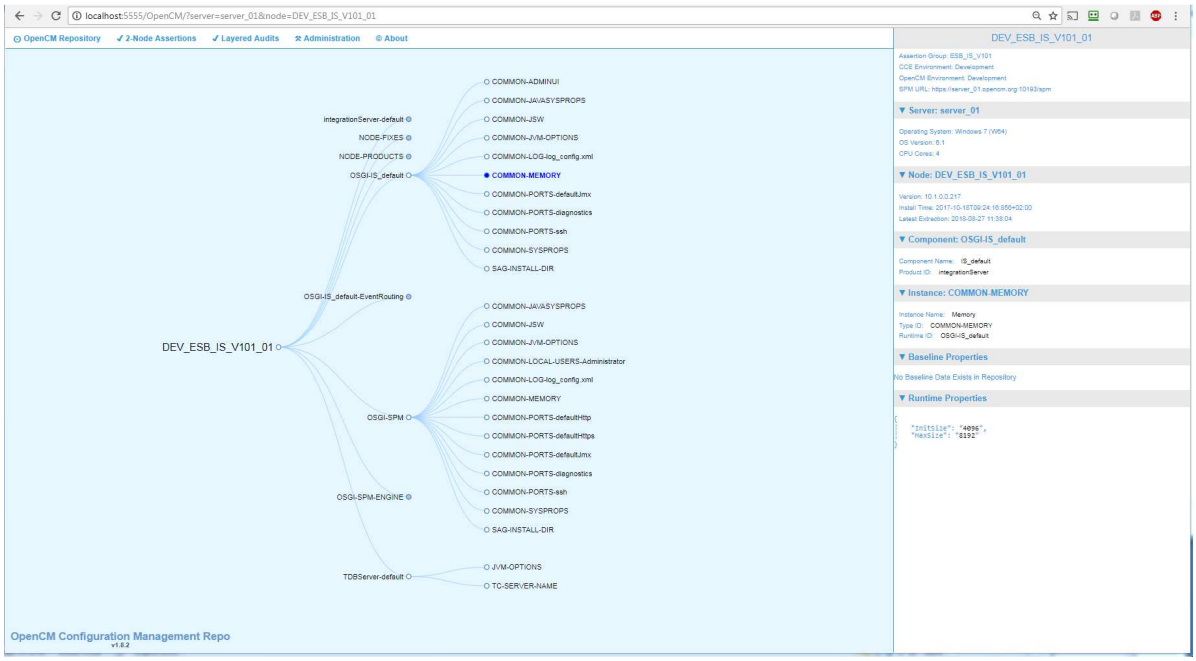
Overview: showing the landscape and relationships between environments, servers and installations:



The structure of the overview is based solely on the defined inventory, - a central store (e.g. a configuration file) describing the different servers and installations in the various environments. The central inventory store 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.

1.5.3.1. 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 Inventory

This is the central store containing information about what servers and installations are present within the various environments.

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 inventory containing the endpoint information:

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

Both approaches assumes a hierarchical definition of installations:

1. Organisation – the name of the organisation where the installation is located
2. Department – the operations group name that is responsible for managing the installation
3. Server – the physical server with a specific server host name hosting the installation
4. Installation – the logical name of the installation
5. Runtimes – within each installation, there can be multiple runtime components. These are actual runtime processes, exposing a port and can be accessed using the username/password combination.

Apart from above, there is additional “metadata” information tagged to two entities:

1. Server – a server can optionally hold the following attributes:
   a. description – free text information related to the server
   b. os – the type of operating system/version. Also optional free text
   c. type – again, optional free text item (e.g. “VM”, “physical”, etc.)

2. Installation – the following attributes can be added to an installation:
    a. environment – a textual name of the environment where the installation sits (e.g. “Dev”)
    b. layer – the logical name of the runtime layer. (e.g. “ESB”, “MFT”, “BPM”, “Caching”, etc.)
    c. sublayer – a logical sub-categorization of the installation (e.g. “Logistics”, “Europe”, “UM”, etc.)
    d. version – the (webMethods) version for the installation.

The metadata for the server is optional (used to visualize the inventory in the UI). However, the installation metadata is used to perform a number of OpenCM functions.

Note: OpenCM inventory and related functions using the inventory assumes and requires that all installation names are unique. There cannot be two installations with the same name.

inventory.properties

This is the 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.

```
inventory:
- name: "OrganisationX"
  departments:
  - name: "DepartmentA"
    servers:
    # ================================================
    #   DEV Environment
    # ================================================
    - name: "server_01.opencm.org"
      description: "Free Text"
      os: "Win2012 R2"
      type: "VM"
      installations:
      - name: "ORGX_DEV_ESB_V101_IS_01"
        environment: "Development"
        layer: "ESB"
        sublayer: "IS"
        version: "103"
        runtimes:
        - name: "OSGI-SPM"
          protocol: "https"
          port: "10093"
          username: "Administrator"
          password: “manage”
        - name: "integrationServer-default"
          protocol: "http"
          port: "5555"
          username: "opencm_extracter"
          password: "manage"
```

The above example shows a single node (installation) with 2 separate runtime components:

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

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.):


```
…
 - name: "ORGX_DEV_ESB_V101_IS_01"
   environment: "Development"
   layer: "ESB"
   sublayer: "IS"
   version: "103"
   runtimes:
   - name: "OSGI-SPM"
     protocol: "https"
     port: "10093"
     username: "Administrator"
     password: "[ENCRYPTED::MCgl9oFVIyVx0YCuXdaGHA==:3Q0dHiJo/PvLmpNQ6r4KRA==]"
   - name: "integrationServer-default"
     protocol: "http"
     port: "5555"
     username: "opencm_extracter"
     password: "[ENCRYPTED::gTtisif2W/C3hoyCRZ70AA==:Uy25UGNd7B+tahTrxNLkwA==]"
```

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

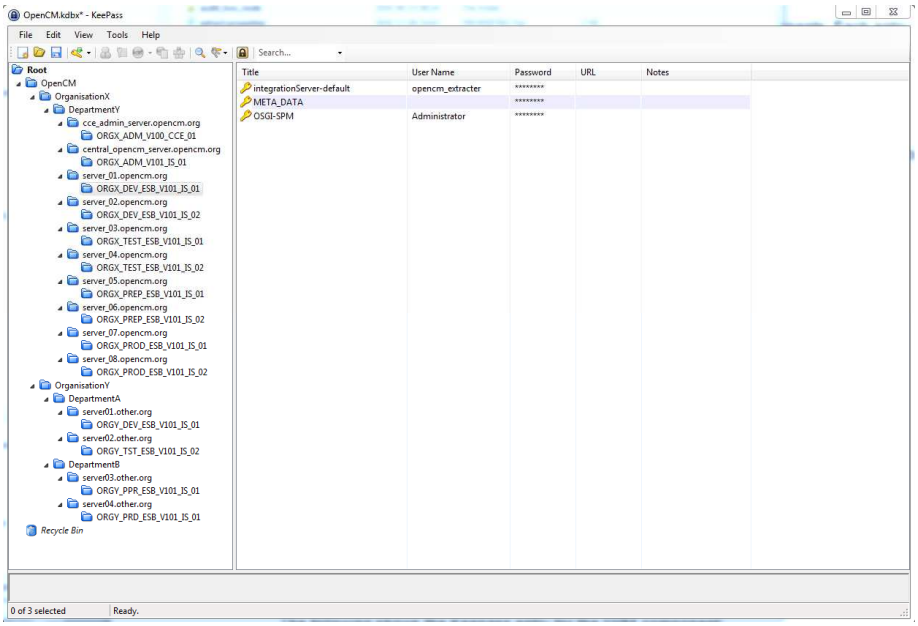
These services can also be invoked from the OpenCM home page, under the “Configuration” menu.

Keepass

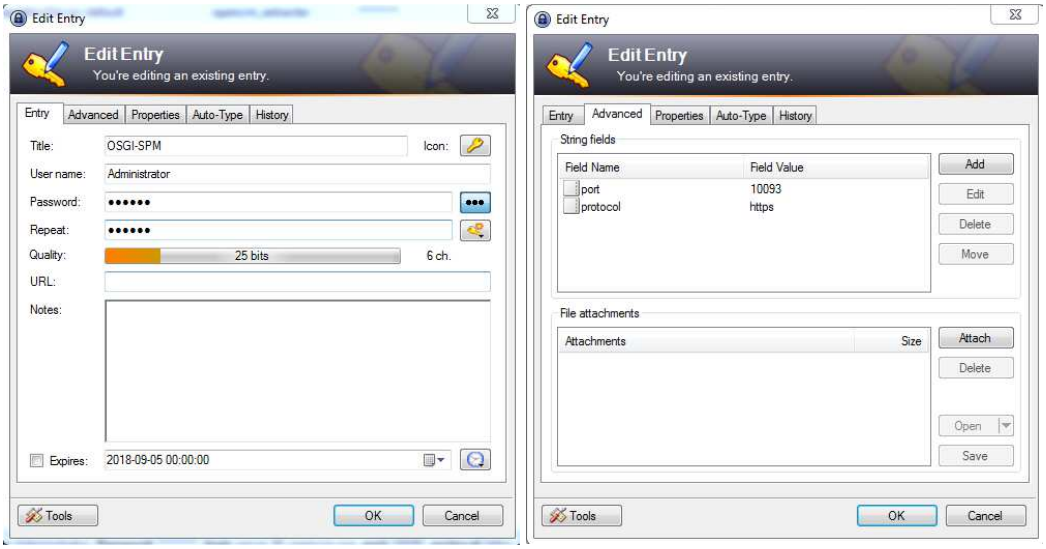
Another way to define the installation nodes is by using Keepass as the holder (or database) of the information. There are some important notes about this approach:

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 inventory.properties.

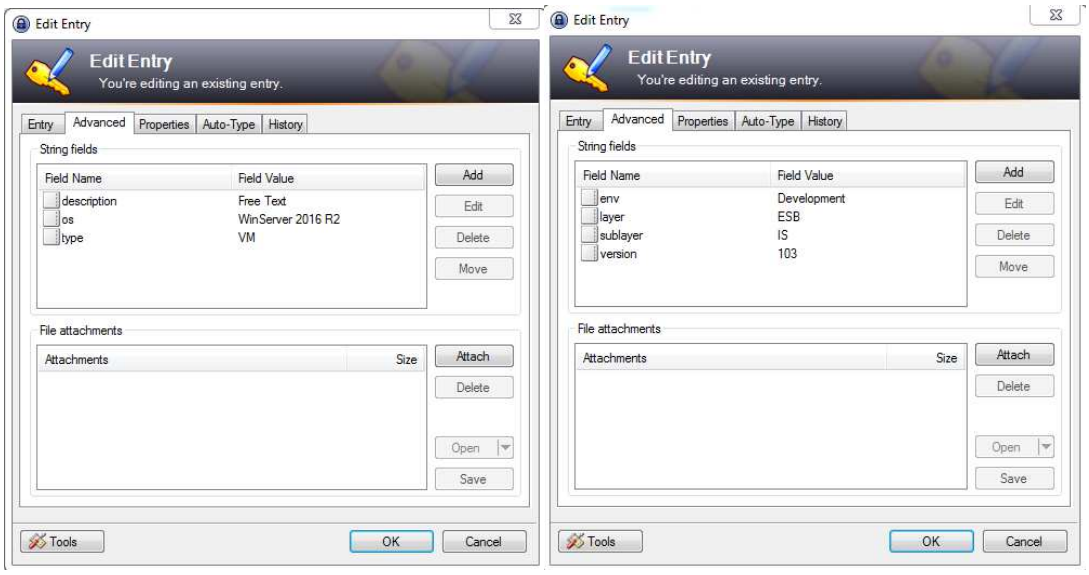


The following shows the Keepass entry for the SPM component:



Each runtime component (i.e. SPM and IntegrationServer) must contain the username/password as well as the additional attributes for port and protocol. This is how OpenCM knows how to connect to the external component.

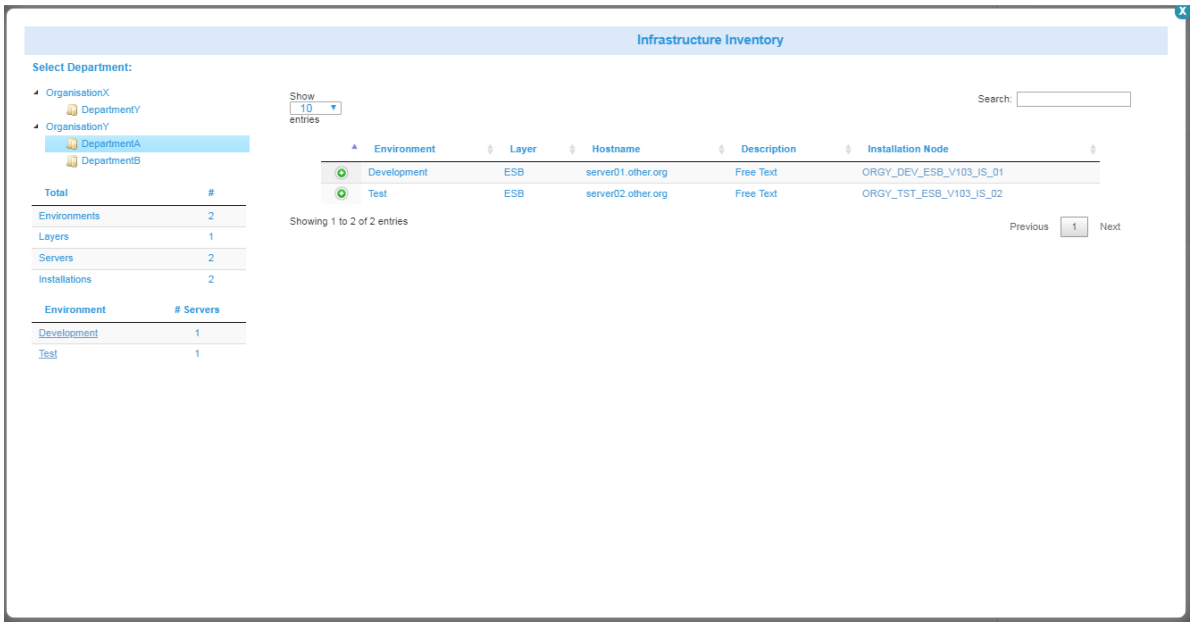
The additional metadata attributes to both server and installation is done by adding a separate item named “META_DATA”. See below for server and installation:



The top group can be any group within the Keepass database, and is references by the “top_group” property in opencm.properties.

4.2 Inventory UI

This function outlines all the opencm nodes defined in the nodes.properties (or other support stores). Use the table functions for searching or sorting the entries.



4.3 Extraction

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

4.3.1 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).

4.3.2 Extract Properties

Extraction is driven by a properties file that describes what to extract. The file is located under the config directory and named “extract.properties”.

The property file is made up of the follwowing (examples included):

```
# ----------------------------------------------------------------------------
# ----------------------------------------------------------------------------
# OpenCM Property file for driving the extraction process (yaml notation).
#
# ----------------------------------------------------------------------------
# Extracting configuration information from specific installations based on inventory
# If there are no children defined, it assumes that all nodes (installations) will be extracted from underneath
extract:
# ------------------------------------------
#  Ex: All nodes under all both orgs
# ------------------------------------------
- org: OrganisationX
- org: OrganisationY
#
# ------------------------------------------
#  Ex: Some from OrganisationX, all nodes under OrganisationY - DepartmentA
# ------------------------------------------
#- org: OrganisationX
#  departments:
#  - dep: DepartmentA
#    environments:
#    - env: Development
#      nodes:
#      - DEV_ESB_V101_IS_02
#    - env: Test
#    - env: Production
#- org: OrganisationY
#  departments:
#  - dep: DepartmentA
#
# ------------------------------------------
#  Ex: Only a single node (can also be done via UI)
# ------------------------------------------
#- org: OrganisationX
#  departments:
#  - dep: DepartmentA
#    environments:
#    - env: Development
#      nodes:
#      - ORGX_DEV_ESB_V101_IS_02
```


4.3.3 Retrieving Information

At this point in time, we have information about which nodes to collect information from (based on the extract and the inventory properties) and we also know how to connect to the remote components. Command Center is not involved at all during extractions, instead we collect 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.
- **IS - WxConfig Information** – Retrieves all key-value pairs from WxConfig (if present)

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)

4.3.4 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

4.4 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.

4.4.1 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.

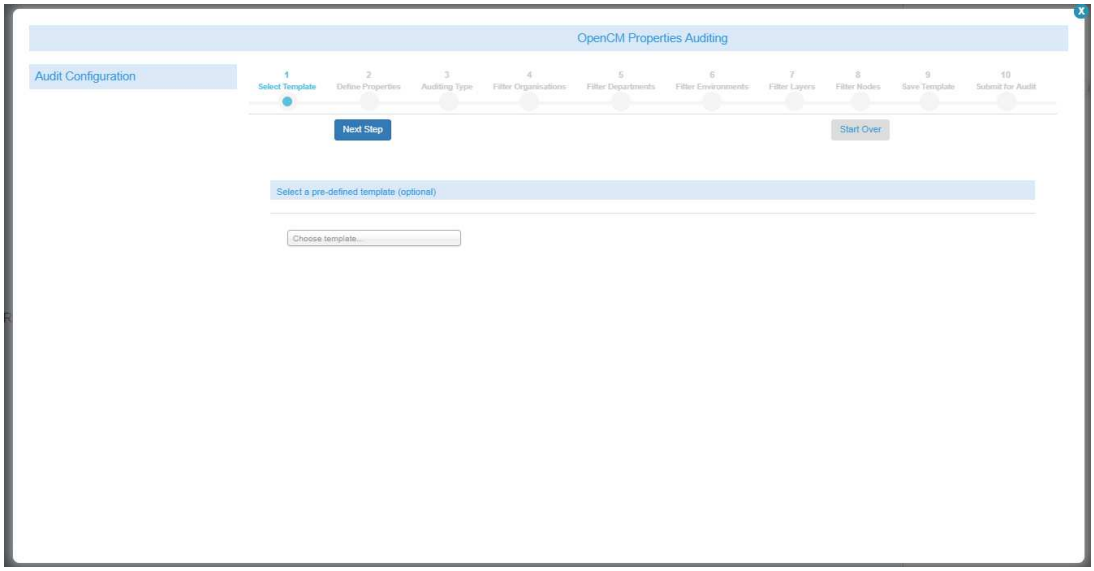
It takes one parameter:

- node (what node to promote)

4.5 Auditing

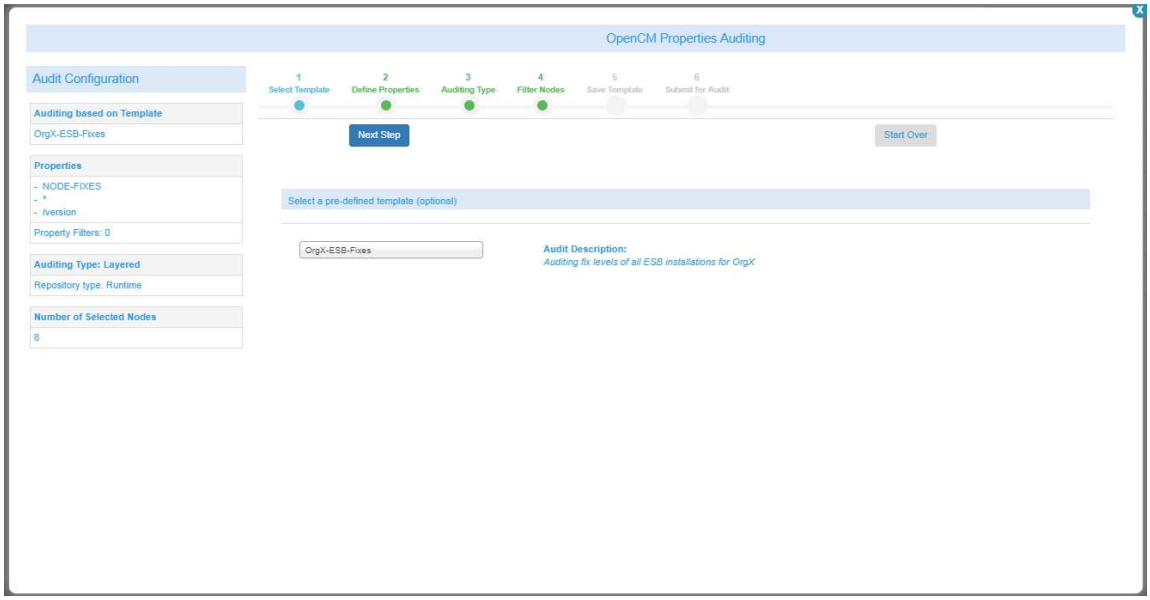
Auditing is the process of detecting differences between property values within the OpenCM repositories. It is performed via a wizard in the user interface where all the auditing configuration is defined and submitted. The result is displayed directly in the OpenCM UI.

1. **Step 1** – Import existing auditing properties.

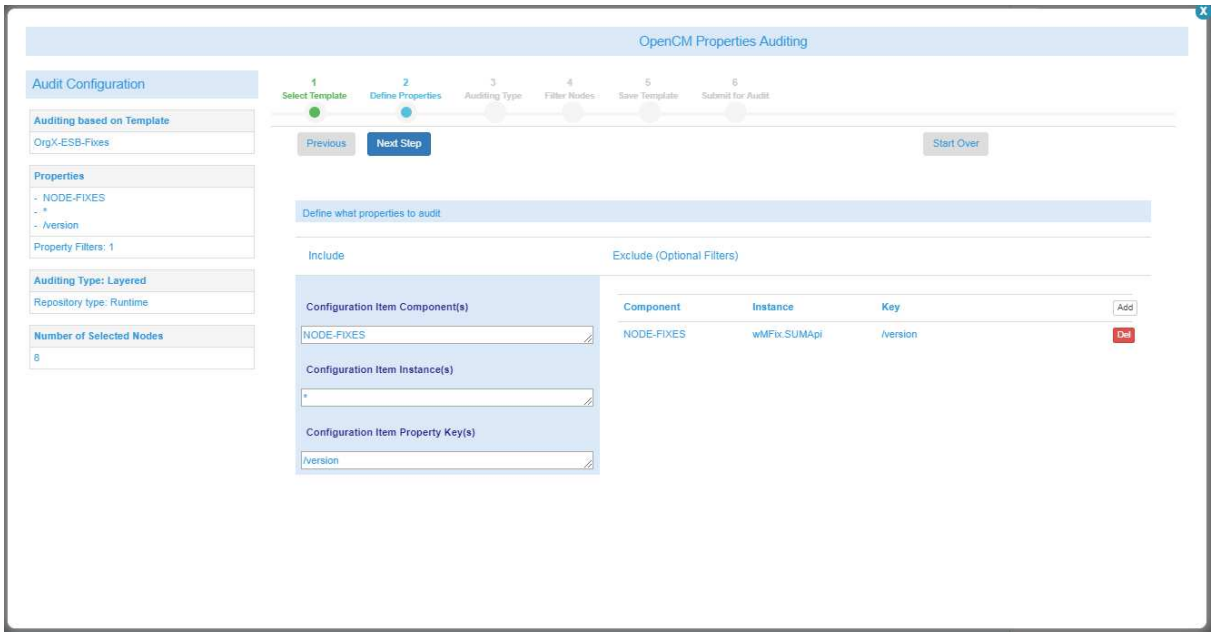


Auditing properties can be saved for later, and the auditing definitions are stored on the server within the opencm/config/audit directory. If selecting an existing auditing template definition, all values in the subsequent steps will be pre-populated.

E.g., selcting the ESB-Fixes template will result in auditing values being defined as follows:



2. **Step 2** – Define what properties to collect (and audit).



The above example follows the ESB-Fixes template. The properties to select are within the “Include” section and defines what components, instances and property keys to retrieve.

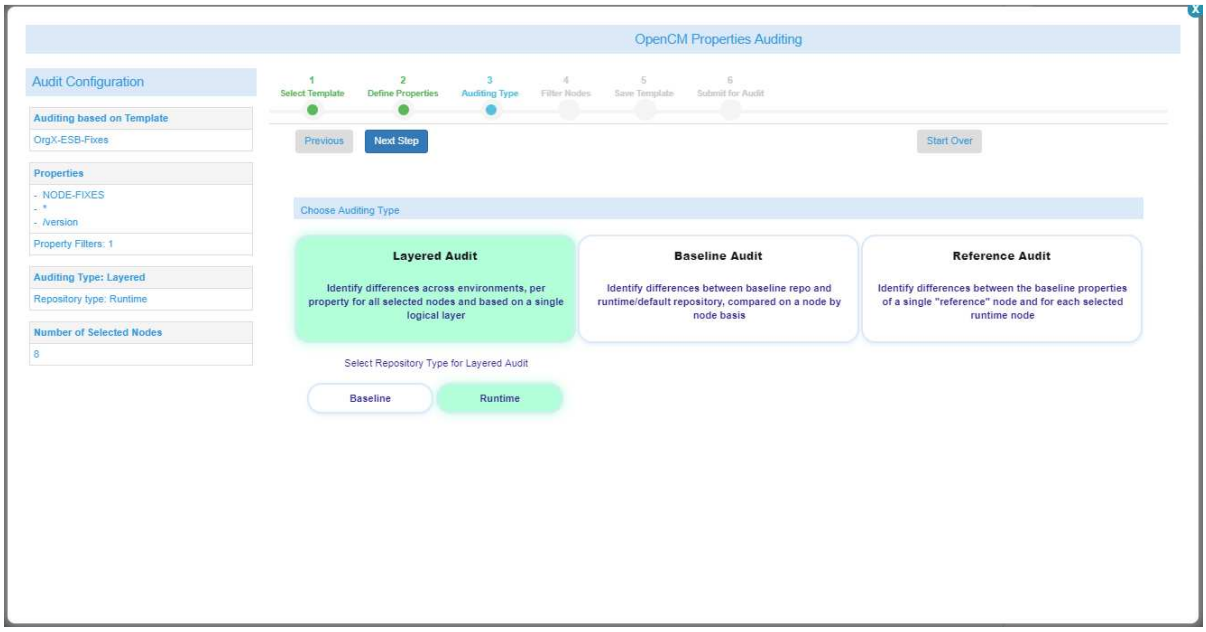
a. Component – refers to the component name under the installation to collect from. Can use wild-card character (“*”), e.g. “integrationServer-*”.

b. Instance – refers to the name of the component instance. Again, can take wild-card characters.

c. Property Key – name of the key to collect value information. In the above example, we are extracting the “/value” key from the property file.

In addition, it is possible to define filters to the properties collected, to be excluded from the result. E.g. it would be possible to exclude SUM Api Fixes from the result by defining the relevant component, instance and property key in the exclude section (as above).

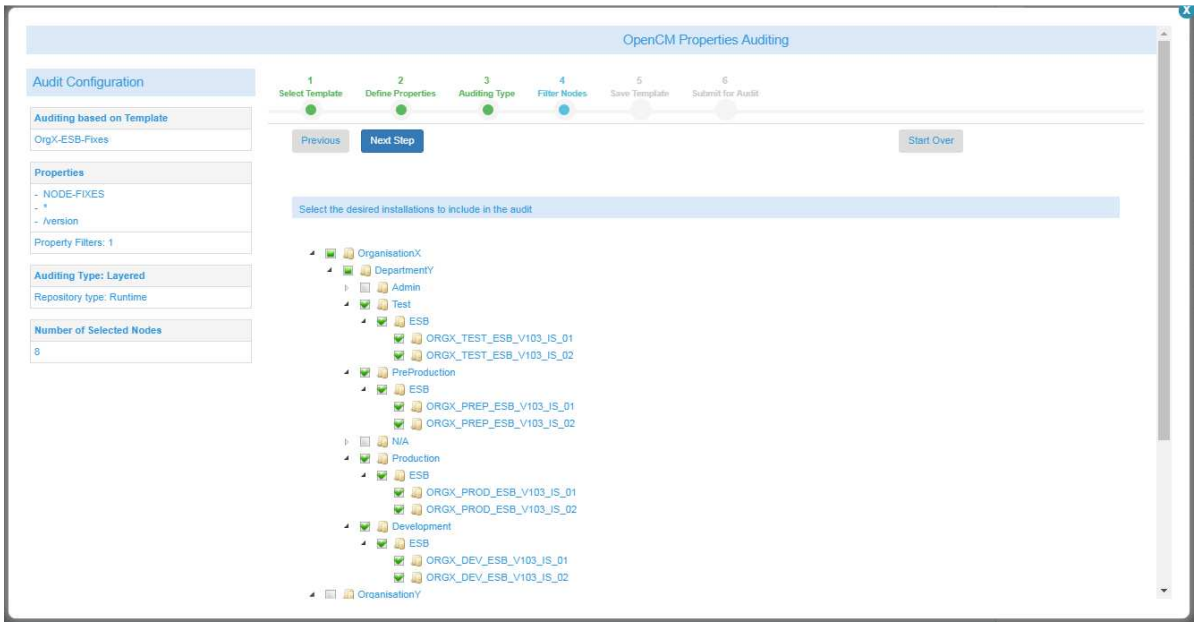
3. **Step 3** – Define what type of audit to perform



In the above example, we want to perform a so-called “layered audit”, which means that we want to compare property values across multiple installations and report differences where one or multiple values are different. E.g. if an Integration Server Core fix level is “3.2” in DEV, but “3.1” in TEST and PROD, then we will see that in the result.

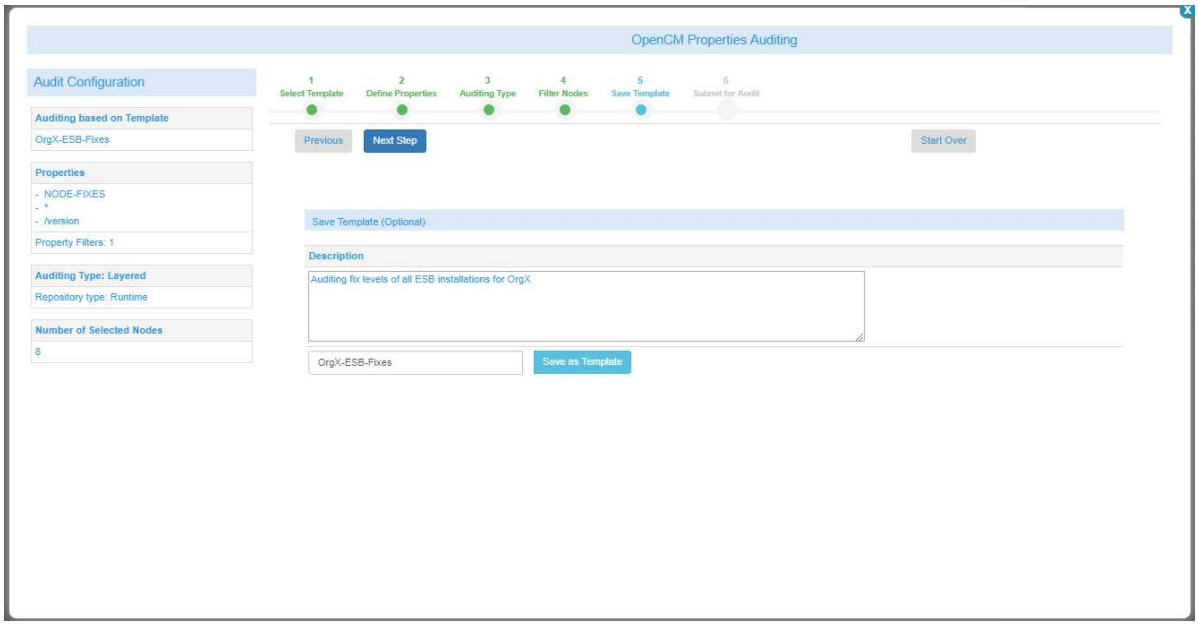
The second option for a layered audit is from which repository we want to look at. Either baseline or runtime repositories will be searched.

4. **Step 4** – Filter Nodes.



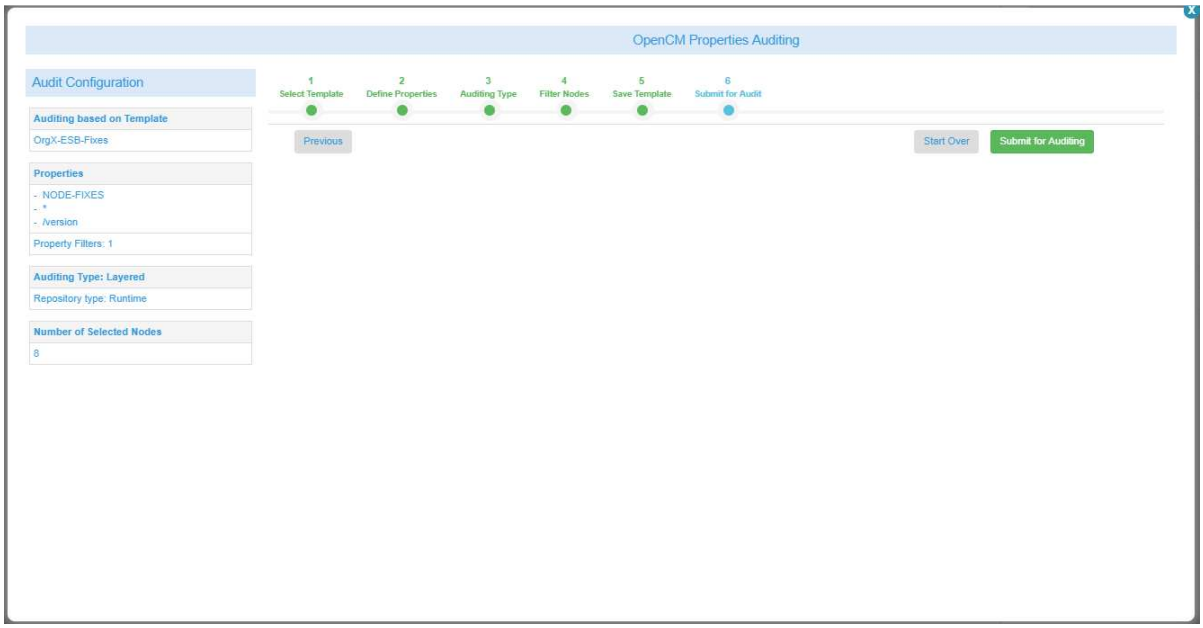
In this example, we will only perform the audit for “Organisation X” ESB nodes for 4 different environments, and not include any other installations in the audit.

5. **Step 5** – Option to save template



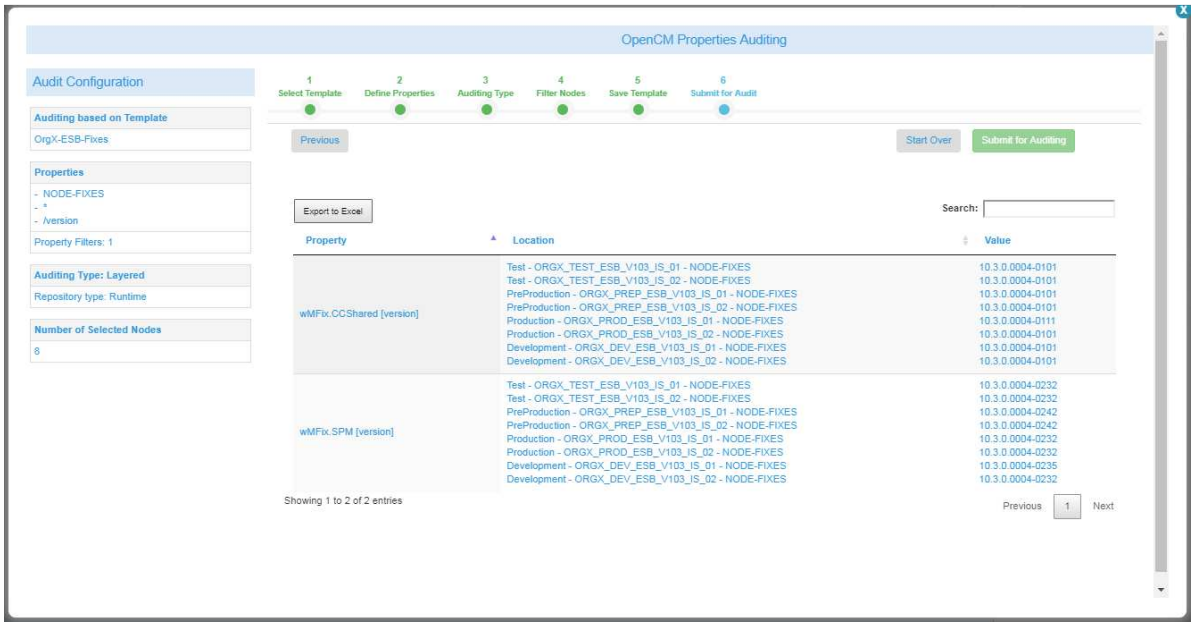
The previosuly selected auditing configuration can now be saved off to a template file, to be used for future auditing activities. A description and template name needs to be defined. If the template name is the same as an already saved template, it will be overwritten with the new values.

6. **Step 6** – Submit



It is now time to submit the audit definitions to the server and perform the work to detect differences.

Once completed, the result looks as follows:



The table has 3 columns:

a. Property – the name of the property

b. Location – where this property has been collected from

c. Value – the value of the property

Note that only differences are reported here. It is also possible to export the table to an Excel file

4.6 UI Configuration

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

4.6.1 OpenCM Administration

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 inventory.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”.

4.6.2 Command Central

1. **Create All Nodes**

   This function creates all environments and nodes based on the the information in the OpenCM node store (either nodes.properties or Keepass). If the environment/node already exists, it will be removed prior to creation. What to create is driven by opencm.properties configuration:

   - “cce_mgmt_create” – contains a list of installation nodes to create. All nodes defined under the particular organisation, department, environment, etc. will then be created in Command Central.

   Also, the name of the Command Central environment can be adjusted based on the following two opencm.properties:

   cce_mgmt_group_syntax:

   cce_mgmt_group_delim: " : "

   - “cce_mgmt_group_syntax” – will be used to construct the CCE group name. E.g., “ORG|DEP”|ENV” will create a group name consisting of those three levels. Another level that can be used is “LAYER”
   - “cce_mgmt_group_delim” – used to separate the above entities used in the group name E.g. “ : “ separates out organisation name, department name and environment as:

     OrganisationX : DevelopmentA : Development

2. **Add Node**

   This function adds the current node to the Command Central UI, and creates (if not present) the corresponding environment. This option is only available when navigating within an installation node.

3. **Create Fix Script**

This function creates a Windows based batch script, containing the operations for applying the fixes installed on a particular node to another node. This option is only available when navigating within an installation 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.

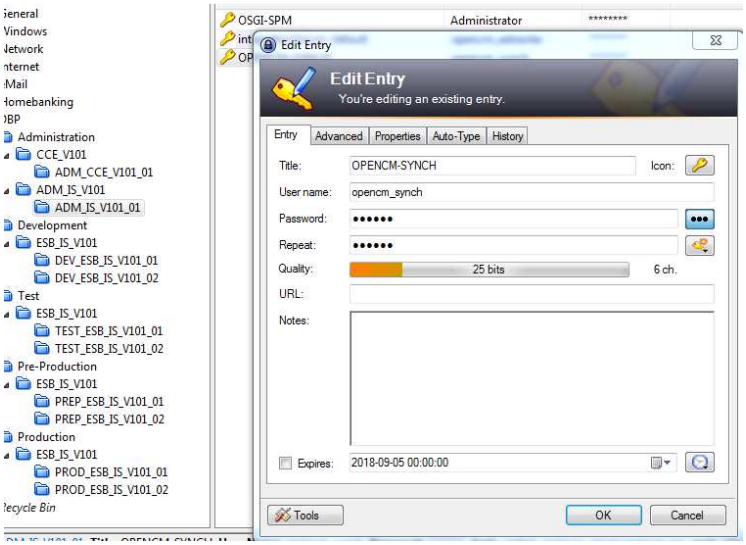
In order to instead perform the data integration automatically, the OpenCM comes with a synchronization feature, which involves zipping up the runtime server directories and performing an FTPS transfer to a target OpenCM location. This is done on a server by server basis from the runtime directory.

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:



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.

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"
```

What to send, is also defined within the opencm.properties file and based on the defined synch configuration:

```
# ------------------------------------------------------------
# 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"
synch:
- org: OrganisationX
  departments:
  - dep: DepartmentA
    environments:
    - env: Development
      nodes:
      - ORGX_DEV_ESB_V101_IS_02
```

(Above example would only send a single node)

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
bootstrap.min.js, bootstrap-waitingfor.min.js	MIT	https://getbootstrap.com/
datatables.min.js, buttons.html5.min.js, dataTables.buttons.min.js, jszip.min.js	MIT	https://datatables.net
chosen.jquery.min.js	MIT	https://harvesthq.github.io/chosen/
d3.v3.min.js	BSD-3-Clause	https://github.com/d3/d3
jquery.json-view.js	MIT	https://github.com/yesmeck/jquery-jsonview
jquery.min.js	MIT	https://jquery.com/download/
jquery.multi-select.js	WTFPL	http://loudev.com/
jquery. showLoading.min.js	MIT	https://www.jqueryscript.net/loading/Loading-Indicator-Plugin-jQuery-showLoading.html
jquery.smartWizard.min.js	MIT	http://techlaboratory.net/smartwizard
jstree.min.js	MIT	https://www.jstree.com
opencm.js	Apache 2.0	https://github.com/SoftwareAG/OpenCM

2.3. css Files

css Files	Licensed under	Source
bootstrap.min.css	MIT	https://getbootstrap.com/
buttons.dataTables.min.css, datatables.min.css	MIT	https://datatables.net
chosen.min.css	MIT	https://harvesthq.github.io/chosen/

jquery.json-view.css	MIT	https://github.com/yesmeck/jquery-jsonview
jstree.min.css	MIT	https://www.jstree.com
multi-select.css	WTFPL	http://loudev.com/
opencm.css	Apache 2.0	https://github.com/SoftwareAG/OpenCM
smart_wizard*.css	MIT	http://techlaboratory.net/smartwizard

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-net-3.6.jar	Apache 2.0	https://commons.apache.org/proper/commons-net/download_net.cgi
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
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
simple-xml-2.7.1.jar	Apache 2.0	https://mvnrepository.com/artifact/org.simpleframework/simple-xml/2.7.1

